INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

T H E S I S

To obtain the grade of

INPG DOCTOR

Speciality: « COMPUTER SYSTEMS AND COMMUNICATIONS »
(INFORMATIQUE : SYSTÈMES ET COMMUNICATION)

Prepared in the VERIMAG laboratory
Graduate school **« MATHEMATICS AND COMPUTER SCIENCE »**
(MATHÉMATIQUES, SCIENCES ET TECHNOLOGIES DE L'INFORMATION, INFORMATIQUE)

presented and defended publicly

by

Laurent Mazaré

on October, 11th 2006

Title:

Computational Soundness of Symbolic Models for Cryptographic Protocols

Supervised by

Yassine Lakhnech

JURY

Roger Mohr	President
Gilles Barthe	Reviewer
Emmanuel Bresson	Reviewer
Jean Goubault-Larrecq	Reviewer
Yassine Lakhnech	Director
Francis Klay	Examiner

Contents

Chapter 1

Introduction

Contents

1.1 Motivation

The recent emergence of electronic voting has created a new need for the verification of cryptographic systems. The main difficulty when analyzing such systems is that there are a wide variety of requirements to achieve security. For example, in the case of e-voting, the ballot value has to remain secret, the identities of the voters have to remain anonymous, and there are other needs which seem to be contradictory: the voters want to be able to test that their votes have been correctly taken into account but it should not be possible for them to prove to a third person the value of their ballots (this is required to avoid ballot selling). To achieve these properties, voting schemes use cryptographic primitives like symmetric encryption: given a message and a key, the encryption algorithm produces a cipher-text. Using that cipher-text, it is very hard to guess the message without knowing the key used by the encryption algorithm.

E-voting is an example of cryptographic protocol. A protocol can be seen as a list of rules that describes correct executions, these rules specify the emissions and receptions of messages by the actors of the protocols called the agents. The objective is to implement a functionality such as sharing a secret key between two agents or authenticating an agent to another one. These protocols are widely used today, from smart cards to wireless networks and typical functionalities are the transfer of a payment card number or the authentication of a user on a system.

1.2 Introducing Security Protocols

1.2.1 Some Elements of Cryptography

Security protocols use some cryptographic primitives to ensure their security. These algorithms are used to protect some secret (encryption schemes), to authenticate some data (digital signature schemes and symmetric encryption schemes), or to provide fresh values (random number generators). Details about these schemes, their uses and examples of implementation can be found in a wide variety of cryptographic surveys [BR03, MvOV96, Sch94, BG01]. In this document, we mainly consider four different types of cryptographic schemes: asymmetric encryption, symmetric encryption, digital signature and random number generation.

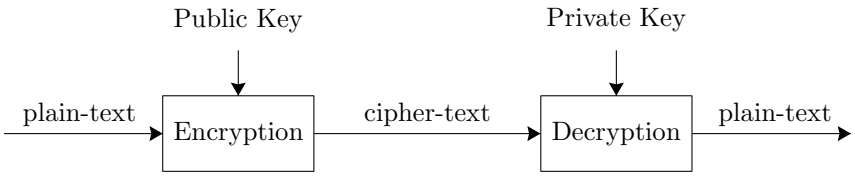
Asymmetric Encryption Schemes An *asymmetric encryption scheme* is composed of three algorithms: the key generation algorithm, the encryption algorithm and the decryption algorithm. As we consider asymmetric cryptography, the key generation algorithm produces a pair of keys containing a public key pk and the related secret key sk. The public key is used for encryption and can be disclosed to anyone whereas the secret key is used for decryption and must remain private. The encryption algorithm transforms a message m called *plain-text* into a message c called the *cipher-text*. The encryption of plain-text m using public key pk is denoted by:

$$c = \mathsf{enc}(m, pk) = \{m\}_{pk}$$

The decryption algorithm takes as input a cipher-text c and a private key sk and outputs the plain-text if the key used for encryption was pk. In order to show the link between pk and sk, the secret key related to public key pk can be denoted by pk^{-1}. pk^{-1} is the inverse key of pk. Then

$$\mathsf{dec}\big(\mathsf{enc}(m, pk), pk^{-1}\big) = m$$

The idea beyond asymmetric cryptography is that everyone can encrypt a message using the public key. This can be viewed as posting a letter in a box. But to decrypt a cipher-text, the secret key is required: to get the letter from the box, you must have its key.



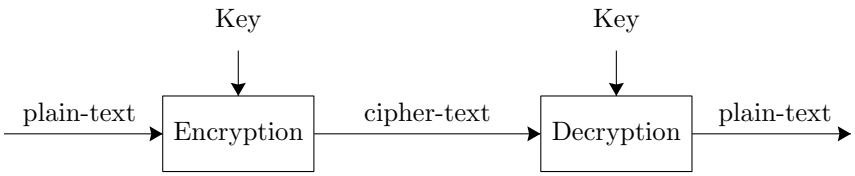
Symmetric Encryption Schemes Asymmetric encryption schemes have a major disadvantage: algorithms are in general very slow to apply. Symmetric encryption allows faster encryptions and decryptions but there is only a single shared encryption and decryption key. Therefore, the key has to be exchanged before using symmetric encryption. A typical use of asymmetric encryption consists in generating a fresh symmetric key and using the public key to encrypt it and send it securely. After that, encryption using the fresh symmetric keys can be used.

A *symmetric encryption scheme* is similar to an asymmetric encryption scheme. It is composed of three algorithms: the key generation algorithm, the encryption algorithm and the decryption algorithm. The difference with asymmetric cryptography is that the key generation algorithm only outputs a single key k instead of a key pair. This key is used by both encryption and decryption.

$$\mathsf{dec}\big(\mathsf{enc}(m, k), k\big) = m$$

Symmetric cryptography can be seen as asymmetric cryptography where the inverse of a key is itself:

$$k = k^{-1}$$



Digital Signature Schemes Digital signature is used to authenticate some information. It is the opposite of public key encryption: the roles of the keys are exchanged: the secret key is needed to transform a plain-text into a sign-text, the public key allows everyone to verify that a signature is valid. Hence digital signature can be seen as an asymmetric encryption scheme where the public key (for the encryption scheme) remains secret and the secret key (for the encryption scheme) is revealed to everyone.

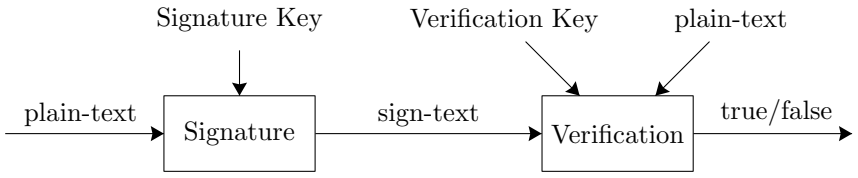
A *digital signature scheme* is composed of three algorithms: a key generation algorithm that outputs a signature key sik (or private key) and a verification key vk (or public key). As before, pairs of keys can be linked by the $\cdot^{-1}$ operator. The signature algorithm produces a sign-text using a plain-text and a signature key. The signature of plain-text m using signature key sik is the sign-text s given by:

$$s = \mathsf{sig}(m, sik) = \{m\}_{sik}$$

Note that it could be possible to guess the plain-text from the sign-text. In general, signature schemes do not guarantee any form of secrecy. Finally, there are two different flavors of verification algorithms.

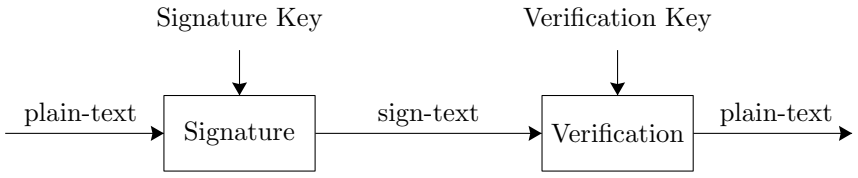
- Selective verification takes as argument the sign-text s, the plain-text m and the verification key vk and outputs $true$ if the sign-text has been produced from the plain-text using the matching signing key. $false$ is output in any other cases.

$$\mathsf{verif}\big(\mathsf{sig}(m, sik), m, sik^{-1}\big) = true$$



- Existential verification takes as argument the sign-text and the verification key and outputs the corresponding plain-text (if the signature and verification keys match).

$$\mathsf{verif}\big(\mathsf{sig}(m, sik), sik^{-1}\big) = m$$



The first type of verifier is more general as such a verifier can be simulated using a verifier of the second type.

Random Number Generators Random numbers are used to ensure freshness of a messages. These numbers are also called *nonces* (for numbers used once). They are generated using a random number generator. A generator is an algorithm that outputs such random numbers. As true randomness is difficult to achieve, pseudo-random number generators are commonly used instead.

1.2.2 Describing Security Protocols

Protocols describe the messages sent between honest participants during a *session*. A session is a single run of the protocol. Most protocols allow multiple concurrent sessions. Participant of the session are called *agents* and are usually denoted A (for Alice) and B (for Bob). A third participant C (for Charlie) represents the adversary which tries to break the protocol (for example by getting some secret information).

A simple way of describing protocols is to use message sequence charts [IT94]. Let us exemplify this on a part of the Needham-Schroeder public key protocol. This protocol involves two agents A and B.

$$\begin{array}{lllll} 1. & A \rightarrow B & : & \{A, N_A\}_{pkB} \\ 2. & B \rightarrow A & : & \{N_A, N_B\}_{pkA} \\ 3. & A \rightarrow B & : & \{N_B\}_{pkB} \end{array}$$

These lines describe a correct execution of one session of the protocol. Each line of the protocol corresponds to the emission of a message by an agent (A for the first line) and a reception of this message by another agent (B for the first line).

- In line 1, the agent A is the initiator of the session. Agent B is the responder. Agent A sends to B her identity and a freshly generated nonce N_A, both encrypted using the public key of B, pkB. Agent B receives the message, decrypts it using his secret key to obtain the identity of the initiator and the nonce N_A.
- In line 2, B sends back to A a message containing the nonce N_A that B just received and a freshly generated nonce N_B. Both are encrypted using the public key of A, pkA. The initiator A receives the message and decrypts it, A verifies that the first nonce corresponds to the nonce she sent to B in line 1 and obtains nonce N_B.
- In line 3, A sends to B the nonce N_B she just received encrypted with the public key of B. B receives the message and decrypts it. Then B checks that the received nonce corresponds to N_B.

The goal of this protocol is to provide authentication of A and B. When the session ends, agent A is sure that she was talking to B and agent B is sure that he was talking to A. To ensure this property, when A decodes the second message, she verifies that the person she is talking to correctly put N_A in it. As N_A was encrypted by the public key of B in the first message, only B could have deduced the value of N_A. When B decodes the third message, he verifies that the nonce is N_B. As N_B only circulated encrypted by the public key of A, A is the only agent that could have deduced N_B. For these two reasons, A thinks that she was talking to B and B thinks that she was talking to A.

1.2.3 Attacking Security Protocols

More than fifteen years after the first publication of the Needham-Schroeder protocol, Gavin Lowe found the first flaw of this protocol [Low95]. This flaw has become famous under the name "man in the middle attack", it works on the public key version of the protocol. This attack exists in the symbolic setting introduced by [DY83]. In this setting, messages are represented using algebraic terms. The adversary is also called the *intruder* in this model. He is able to intercept and view any

message that circulates on the network. He can forge new messages and send them to an honest agent while impersonating the identity of another agent. The messages that an intruder can forge are described later in this document. Basically, the idea is that if the intruder knows a cipher-text and the inverse of the encryption key, he can decrypt the message. He can also pair messages and perform projections. He can encrypt messages using the encryption keys he knows. This flaw uses two sessions: one between an honest agent A and the intruder C and another between A and another honest agent B. In this last session, C impersonates agent A and tries to make B believe that he is talking to A. A condition for this flaw is that an honest agent has to start a session with the intruder. The unfolding of this flaw is detailed in figure 1.1. In this attack, the intruder

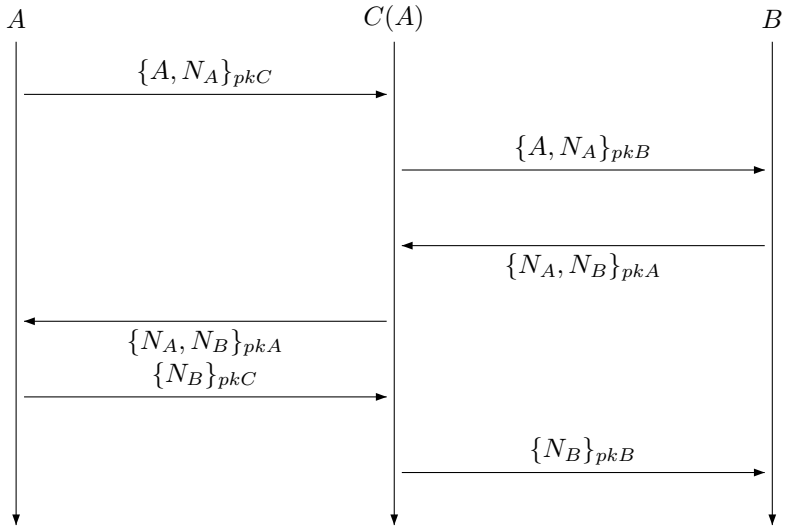


Figure 1.1: Man in the middle attack on the Needham-Schroeder protocol

uses the message from his session with A to make B think that he is A. When B receives message $\{A, N_A\}_{pkC}$, he uses his secret key to decrypt it and get A and N_A. Using the public key of B, the intruder can forge $\{A, N_A\}_{pkB}$. When the intruder receives the answer from B, he directly forwards the message to A as he cannot decrypt this message. A answers message $\{N_B\}_{pkC}$ and C uses his secret key to get the value of nonce N_B. Finally, the intruder forges message $\{N_B\}_{pkB}$ and sends it to B so that B believes that the execution has correctly terminated.

1.3 Verification of Protocols

Cryptographic protocols are used to ensure some security properties. Hence validation of cryptographic protocols is a deeply investigated domain. The objective of this research field is to prove formally that a given protocol verifies a given property. Let us first present which properties can be of interest for a protocol.

1.3.1 Properties

Secrecy The secrecy property concerns a message used by the protocol. This message is typically a nonce or a secret key that should not become public at the end of the protocol. The word "public" may have two different meanings: a message can be public if the adversary is able to show its value or it can be public as soon as the adversary is able to distinguish this message from a randomly generated message. Hence there are (at least) two distinct flavors for the secrecy property.

- *Weak Secrecy:* in this case, the adversary should not be able to produce the whole message. This approach is used in the Dolev-Yao model [DY83]. It can also be used in the computational setting even if strong secrecy makes more sense in this context.

- *Strong Secrecy:* here, the adversary should not be able to deduce any information on the secret. This definition of secrecy is used in the spi-calculus model [AG99]. It is also used in the computational world: for example, the SecNonce property implies that an adversary should not be able to distinguish an execution where a random bit-string value bs_0 is used from an execution where another random bit-string value bs_1 is used, even if the adversary knows or choses bs_0 and bs_1. SecNonce ensures that the adversary is not able to deduce any bit of information on the secret. The SecNonce property was originally introduced in [CW05].

Authentication There are several variants of authentication. A taxonomy of these have been proposed by Lowe in [Low97b]. This classification distinguishes four levels of authentication.

1. Aliveness of an agent A: this means that A has executed the protocol.

2. Weak Agreement of agent B with agent A: this signifies that A has executed a session of the protocol that involved B.

3. Non-injective Agreement of B with A on a set of messages M: this means that A has executed a session with agent B in which A and B agreed on the value of the set of messages M.

4. Injective Agreement of B with A on M: this signifies that there is a non-injective agreement of B with A on M and that each execution of the protocol for B corresponds to a unique execution for A.

Non-Repudiation Non-repudiation ensures that the author of a message cannot later claim not to be the author. There is a proof that the sender sent the message. This is an indispensable property for the electronic commerce protocols, the seller needing to prove to the bank that the client has really paid. This is often realized using digital signatures.

Fairness Fairness ensures that one of the parties cannot end the protocol part-way through and gain some unfair advantages over the other party. For example, a contract signing protocol where either each agent receives the signed contract, or neither receives any.

Anonymity Anonymity ensures that the identity of an agent is protected with respect to the message that he sent. For example, in a voting protocol the vote must not be linked back to the voter who cast it. In this case, the messages themselves do not have to be secret, only their association with a particular agent.

Information Flow Notions Information flows notions can be used to represent more complex properties. There are plenty of different properties that can be ensured using information flow. A taxonomy of these properties applied to the case of CCS is given in [FG94b, FG94a]. CCS is a process algebra introduced by Milner [Mil95] but the taxonomy also holds for various execution models.

One of the key concept in information flow is non-interference. The idea is that the set of actions is separated into two sub-sets: *low actions* and *high actions*. Low actions are public and can be seen by the adversary whereas high actions cannot. A process is safe for non-interference if it is impossible for the adversary to tell anything about high actions just by looking at low actions. Non-interference notions can be used to model other security properties such as anonymity or fairness.

1.3.2 The Two Approaches for Verification

When considering real implementations of security protocols, proofs are very hard to perform. Therefore the perfect cryptography hypothesis is commonly used to simplify these proofs. When making that hypothesis, the encryption schemes are supposed to be perfect. Hence it is impossible to deduce a plain-text from its cipher-text without knowing the decryption key. Even with this hypothesis, idealized protocols may contain some logical flaws. We already presented the best known example of such flaws which has been found on the Needham-Schroeder public key protocol. This authentication protocol was designed in 1978 [NS78]. It was commonly admitted that this protocol was secure. However Gavin Lowe found an attack on this protocol seventeen years later [Low95] by using a model-checker. Lowe also proposed a corrected version of this protocol in [Low96]. Following that example, there have been lots of works using formal methods to verify cryptographic protocols [Mea00, CS02]. This approach is called *formal verification* of cryptographic protocols (or symbolic verification). It originates from the work of Dolev and Yao [DY83]. The essential part of this approach is the perfect cryptography assumption that can be roughly summarized as follows: messages are represented as algebraic terms, fresh nonce creation is perfect, that is, nonces range over an infinite domain and freshness is absolute, the same holds for key creation. Moreover, there is no way to guess a nonce or a key and no information can be extracted from an encrypted message unless the inverse of the key used to encrypt the message is known. In this approach there is a single attacker that is modeled as an infinite process without bounds on its computational resources. Despite the strong assumptions concerning the cryptographic primitives, it is a difficult task to design correct protocols for symbolic verification. Some protocols where flaws have been found are listed in the Clark and Jacob survey [CJ97]. The good news about this approach is that a rich collection of automatic verification methods and tools have been developed [RT01, BLP03b, Son99, GL00, Bla01]. Automatic verification of protocols is in general an undecidable problem. However, some tools have been designed using either restrictions on protocols or abstract interpretation. In most of the case, these tools perform an efficient checking of security protocols.

The symbolic approach is opposed to the *computational approach*. In this last model, cryptographic primitives operate on strings of bits and their security is defined in terms of high complexity and weak probability of success [GM84, BKR] of any attacker. Protocols as well as attackers are randomized polynomial-time Turing machines. This computational approach is recognized as more realistic than the symbolic approach. Adversaries have the whole computing power of Turing machines and the cryptographic primitives are not idealized. The main drawback of this approach is its complexity. This makes it very difficult to design automatic verification tools, although this is still possible [Bla05]. Proofs are very hard to perform in this model, their complexities make it difficult even to check these proofs. For this reason, some errors can be discovered in computational proofs even years after their publications [CBH05].

Therefore, results of the type:

> If protocol Π uses the cryptographic schemes $S_1, \cdots, S_N$, if each schema S_i is correct with respect to the security notion C_i and if some additional syntactic conditions are satisfied by Π then the symbolic model is a safe abstraction of the computational, that is, correctness of the protocol established in the symbolic model implies its correctness in the computational one.

are of extreme importance for gaining confidence that a cryptographic protocol is secure.

1.3.3 Bridging the Gap

In the last years, attempts have been made to bridge the gap between these two approaches. The ultimate objective is to be able to prove security in the symbolic model, then to prove properties on the encryption scheme and with that to deduce security of the protocol in the computational model.

The Passive Case

Abadi and Rogaway were the first to link the symbolic and computational views of cryptography in [AR00]. They limit themselves to the case of a passive adversary (eavesdropper) that can only see a single message. Moreover, the only allowed cryptographic primitive is symmetric encryption. Abadi and Rogaway define a notion of *indistinguishability* for messages in the symbolic model. They start by introducing patterns which represent the information that an intruder can deduce from a message, the idea is to replace encryptions that cannot be opened (i.e. whose decryption key cannot be deduced) by the black box symbol $\square$. Two messages are indistinguishable if they have the same pattern (up to renaming). The main result of this paper is that if two messages M and N are indistinguishable and the encryption scheme used to compute these messages enjoys some security properties called type-0 security, then the distributions corresponding to computations of M and N cannot be distinguished by a polynomial time adversary. There is a limitation on M and N which should not contain any encryption cycles. Let us exemplify their result. The messages $\{K_1, K_2\}_K$ and $\{K_1, K_1\}_K$ have the same pattern $\square$. These messages are indistinguishable for a symbolic adversary. Therefore computational adversaries cannot distinguish bit-strings that represent the first message from bit-strings representing the second one. Type-0 security is a variant of semantic security against chosen plain-text attacks [GM84] (IND-CPA).

This initial work was extended the following year by Abadi and Jürjens in [AJ01]. Instead of only considering adversaries that observe a single message, they consider general eavesdroppers which observe the unfolding of a whole protocol. The idea here is that if the protocol is secure in the symbolic setting, then computational adversaries cannot perform any attack with non-negligible probability. Of course, the encryption scheme still has to verify some strong security notions and key cycles are still forbidden.

This work also introduces the notion of confusion-free encryption scheme, which is helpful in order to prove completeness of the Abadi-Rogaway logic. Completeness is the reciprocal of the main result stated in [AR00]: if a computational adversary cannot distinguish between the implementation of M and N, then M and N are equivalent in the symbolic setting. This result was proved by Micciancio and Warinschi in [MW04a]. Without adding the confusion-freeness hypothesis, the completeness result is false. For some specific secure encryption schemes, implementation of messages $\{K_1\}_{K_2}, K_2$ and $\{K_1\}_{K_2}, K_3$ cannot be distinguished by a computational adversary: the decryption of the first part of each message always succeeds and returns a key. However, the adversary cannot compare the result given by decryption with the value of K_1 as K_1 does not appear anywhere else in these messages. The confusion-freeness hypothesis states that decryption with a wrong key should fail most of the time. By adding this hypothesis, the completeness result can be proved.

The Abadi and Rogaway result was extended in several other directions. An interesting development is the case of weak passwords. Abadi and Warinschi considered the case of low entropy password in [AW05]. The number of possible passwords is low so the adversary can test all of them. However, the decryption scheme succeeds even when using a wrong password so the adversary cannot be sure that he deduced the right password. For example, if the adversary is given the encryption of some nonce N using a password W, he can decrypt the cipher-text with any password W'. The result is a nonce N'. If the adversary has another way to know the value of N, then he can try to decrypt the cipher-text with any possible password W' (as their number is limited) and whenever the resulting nonce N' is equal to N the adversary can deduce that W' is equal to W with non-negligible probability. Therefore, the adversary can deduce the value of W, this is called a *guessing attack*. The main idea is that guessing attacks are possible if there are two different ways of obtaining the value of some messages (the nonce N in the previous case). Guessing attacks are not fully considered in [AW05]. The main result of this paper is that if the symbolic adversary cannot deduce a password from a given message then the computational adversary cannot distinguish the password used in an implementation of this message from a randomly sampled password. This result has been extended to guessing attacks by Abadi, Baudet and Warinschi in [ABW06]. This work enhances the symbolic equivalence by taking into account new deductions for the adversary.

Dynamic Cases

Although interesting, the case of passive eavesdroppers cannot be used to represent real-world attacks. Therefore, there have been numerous efforts to extend the previous results to the case of a non-passive adversary. Once again, several directions have been explored.

First, it is possible to just prove computational security in the passive setting. Then using some compiler, one can derive a protocol that is secure in the active setting from the original protocol (by adding some authentication primitives). This approach has been introduced by Katz and Yung for group key exchange protocols in [KY03]. In this paper, they introduce a compiler which given a key exchange protocol that is secure in the passive setting outputs an authentified key exchange protocol secure in the active adversary model. The main drawback is that this cannot be used to analyze existing protocols. Moreover the compiler may add some unnecessary complexities to the protocol.

Micciancio and Panjwani considered the case of *adaptive adversaries* in [MP05]. The adaptive setting is a straightforward extension of the Abadi-Rogaway result. A computational adversary issues sequences of symbolic messages M_i, N_i such that $M_1, ..., M_i$ and $N_1, ..., N_i$ are equivalent. The adversary is given implementations of either M_i or N_i and has to decide which is the case. The interesting point is that the adversary behave in an adaptive way, i.e. after receiving the implementation of M_0 or N_0, he can choose messages for M_1 and N_1 that brings him better chances of success. These adaptive steps are repeated a polynomial number of times. The only cryptographic primitive allowed in messages is symmetric encryption. The main result states that if the encryption scheme is secure, then the adversary has negligible probability to win. This setting is exemplified on the case of group multicast for which it is particularly well suited.

A restriction of this work is that keys are required to be shared before being used only. Specifically, a key K cannot be sent (even encrypted) after it has been used to encrypt another message. The origin of the problem is that handling keys in a general way raises issues related to the selective decommitment problem for which no answer is known today [DNRS99]. This problem always comes when considering adaptive corruptions, thus to our knowledge, all the works considered here only use a static corruption model (a "simple" solution could be to consider non-committing encryption as introduced by Canetti et al. in [CFGN96] , however this has not been explored yet).

Instead of looking at adaptive adversaries, Bogdan Warinschi decided to consider protocol adversaries in the symbolic Dolev-Yao setting. His first result is a proof of computational safety for the Needham-Schroeder-Lowe protocol. This result only holds when considering semantic security against chosen-cipher-text attacks (IND-CCA). The originality of this proof, presented in [War03], is that computational safety is derived from safety in the symbolic setting. The idea is to build some adversaries $\mathcal{B}_i$ against IND-CCA from an adversary $\mathcal{A}$ against the protocol such that whenever $\mathcal{A}$ manages to produce a trace that is impossible in the symbolic setting, then one of the $\mathcal{B}_i$ can win its challenge. Therefore, $\mathcal{A}$'s behavior cannot differ from possible behaviors in the symbolic setting with non-negligible probability. This paper also proves that the IND-CPA hypothesis is not sufficient to ensure computational security of the Needham-Schroeder-Lowe protocol.

This computational soundness result has then been generalized to arbitrary protocols by Micciancio and Warinschi in [MW04c]. The authors consider protocols that only use public key encryption. Encryption nesting, message forwarding and sending of secret keys are disallowed. The hypothesis made on the public key encryption scheme used to implement the protocol is still IND-CCA. The main result is that if a protocol is secure in the symbolic world, then its implementation is also secure against computational adversaries. This is done for an authentication property but the result can be extended to handle any trace property. The proof technique used in this paper (which is derived from the proof of [War03]) has inspired several further works on the subject. In particular, section 6.2 of this document uses proofs that are closely related to this initial one. The principle is always the same: $\mathcal{A}$ is an adversary against the protocol, several adversaries can be built against the different cryptographic primitive (so against IND-CCA for encryption, selective forgery for signature...). These adversaries use $\mathcal{A}$ in such a way that whenever $\mathcal{A}$ produces a trace that cannot be mapped on a possible trace in the symbolic world, then one of

the new adversaries can win its challenge, hence one of the cryptographic primitive can be broken.

The work of Micciancio and Warinschi suffers from numerous limitations that prevent one from using it on most of the classical protocols as detailed in Clark and Jacob survey [CJ97] or as presented in the SPORE archive [1]. The limitations on encryption nesting and message forwarding were removed by Cortier and Warinschi in [CW05]. This paper also allows protocols that use digital signature and introduces message labels. These labels can be thought as the random coins used to perform an encryption or a signature: if two identical messages have the same label, then their implementations are exactly the same. Such notations are useful when one wants to sign an encryption and send the encryption along for example. Moreover, this work does not limit itself to trace properties, strong secrecy of nonces (SecNonce) is also considered. The main result is that if some protocols preserves secrecy of a nonce N in the symbolic world, then an adversary against the implementation of the protocol cannot, after executing the protocol, differentiate the value of N from a randomly sampled nonce value.

At the same time, we concurrently proved a similar result for protocols using asymmetric encryption and digital signature in [JLM05a]. Although we do not handle labels and we only consider trace properties (and not SecNonce), our main improvement with respect to [CW05] is to allow sending of secret keys. Our symbolic model is exactly the Dolev-Yao model, hence several automatic verification tools can be used to verify properties in the symbolic setting. After that, we added symmetric encryption (which can be thought of as authenticated encryption [BN00]) and hashing in [JLM05b]. Several problems arise when considering emission of secret keys as message cannot be entirely parsed when they are received, this is detailed in section 6.2. A solution to this is to ask new requirements on cryptographic primitive as Romain Janvier does in his PhD thesis [Jan06] (but these requirements are not linked – for now – to classical security requirements). Hashing raises similar problems, however to the best of our knowledge we are the only ones to consider hashing in the standard model and not in the Random Oracle Model (ROM).

The Reactive Approach

An impressive work has been achieved in a sequence of papers by Backes, Pflizmann et al [PSW00, BPW03a, BPW03b, BPW04]. Their different works are not based on the classical Dolev-Yao model but adopt a more realistic approach in the symbolic setting. Both their symbolic and computational models are built upon a *cryptographic library* which is called by the protocol participants each time they want to perform a cryptographic operation. Agents do not really have access to the bit-strings representing the different messages but use handles on these bit-strings in order to perform their operations. Handles and their values are stored within the cryptographic library which contains all the messages that are exchanged during the protocol execution.

In the symbolic setting, the library is centralized. Each message is linked to its type and to the agents that can access it. Whenever an agent wants to encrypt a message using some key, he submits a handle to the message and an handle to the key, the library checks that the agent can access both the message and the key. If this is the case, the library creates a new handle and stores the encryption at this position, the handle is returned to the agent. Operations are hence made symbolically. This model is closed to [THG99] where knowledge of each agent is considered instead of just considering the knowledge of the adversary (which is usually done in the Dolev-Yao model). In the computational setting, the library is distributed to each agent. Operations are performed on bit-strings even if the operations are the same as the one of the symbolic version of the library.

In both models, agents interact with the library but there is also an adversary, representing dishonest agents, that uses the library. Adversaries and agents are both polynomial time Turing machines and agents use the same symbolic operations in both setting. Adversaries can use the cryptographic library. In the computational setting, they can also access the bit-strings stored by the library and submit new bit-strings to replace them. Therefore such adversaries can use

[1] Security Protocols Open Repository http://www.lsv.ens-cachan.fr/spore

the whole computing power of polynomial-time Turing machine on the implementations of the different messages.

The main result is the following: for any honest agents H and any computational adversary $\mathcal{A}_c$, if the cryptographic primitives are secure for some criteria, then there exists a symbolic adversary $\mathcal{A}_s$ such that H cannot distinguish whether it uses the computational library with $\mathcal{A}_c$ or the symbolic library with $\mathcal{A}_s$. This means that if a protocol is secure with respect to symbolic adversaries using the symbolic library, then it is also secure when implemented with the computational library. These results were first given for a cryptographic library that only handles asymmetric cryptography and digital signatures, this was done in [BPW03a]. Then these results were extended to handle symmetric cryptography in [BP04] and to handle message authentication codes in [BPW03b]. Security requirements are similar to the ones required by other approaches, that is IND-CCA for asymmetric encryption, IND-CPA and existential forgery for symmetric encryption and existential forgery for digital signature and message authentication code.

The results of Backes et al. have several advantages compared to the approach proposed in this document, in [CW05] or in [Jan06].

1. First, they handle an unbounded number of sessions, this is also done in [CW05] for asymmetric encryption and digital signature. For symmetric encryption, results of chapter 8 should make the classical proof work but this has yet to be written.

2. They also have a weaker requirement for the digital signature scheme. To ensure that their simulator works, messages should be entirely parsed at reception time. If we also require messages to be parseable at reception time, we should also be able to consider this weaker requirement.

3. Message authentication code is considered as a cryptographic primitive but adding this to our model seems to be pretty straightforward.

4. Finally, their reactive frameworks makes it possible to consider more complex properties. Although opacity as detailed in chapter 9 also handles complex properties, its computational soundness has only been studied in the passive setting.

The disadvantages of these results with respect to ours is that some protocols cannot be considered, especially when a key is sent after being used to encrypt a message. The reactive simulatability approach cannot handle such late commitments. Moreover, their symbolic model is not exactly the Dolev-Yao model, hence it is not sure that it could be automatically verified. As we consider the Dolev-Yao model, we have access to a wide variety of tools that prove diverse properties on protocols. Finally, they do not consider hashing or modular exponentiation as cryptographic primitives. The case of hashing is even worse than that: Backes et al. also proved recently in [BPW06] that adding hashing is not possible in their framework without considering the random oracle model or without allowing only hashing of single nonces.

In [BP05b], Backes et al. consider new computational notions of secrecy specifically adapted to cryptographic keys and to "non-cryptographic" data like some English texts. A data is symbolically secret if the adversary does not know the handle that refers to that data in the cryptographic library. Computational indistinguishability implies that the data should, at the end of the protocol, be indistinguishable from a randomly sampled value. For nonces and non-cryptographic data, this notion is close to SecNonce as introduced in [CW05]. For symmetric cryptography, a key can only be secret if it has not been used to perform encryptions that are viewable by the intruder. As the encryption scheme might be key-revealing, it is easy for the adversary to distinguish the real key from a random key if he has access to some cipher-text. For this reason, we introduce the SecKey property in section 6.3 which states that a key can still be secure although some information on this key might be known by the adversary.

The works of Canetti [Can01, Can04] are close to those of Backes but add the notion of universal composability. This consists in defining an idealized version of the function that has to be implemented, and then in showing that it is impossible to distinguish the implementation from its idealized version. If this result holds for a single execution, then indistinguishability also

holds when considering multiple executions in any environment. The cryptographic primitives considered in [Can01] are only asymmetric cryptography and digital signature. As in our work, the security requirements on these primitives are respectively semantic security against chosen-cipher-text attacks and unforgeability. Symmetric cryptography and key sending were studied in [CK01]. In [Can04], Canetti corrects a proof of construction of the digital signature scheme. This illustrates once more the difficulties when reasoning in the computational model.

Compared to our work, Canetti's results have the advantage of composability: it is sufficient to prove security for one execution then it holds for multiple concurrent executions even if the number of such executions is polynomial. However the composability approach may make the property harder to satisfy than in our model.

Peter Laud [Lau04] defined a transformation technique on protocols which allows one to verify whether the protocol preserves secrecy or not. His requirement on asymmetric encryption schemes is still semantic security against chosen-cipher-text attacks. The main result is that if a protocol preserves secrecy in the symbolic setting, then it verifies strong secrecy in the computational world. In particular, he authorizes encryption cycles. His results only apply in the case of a bounded number of sessions and on secrecy properties.

1.4 Thesis Contributions

In this thesis, we propose to extend previous results originating from the work of Micciancio and Warinschi [MW04c]. The objective is here to handle a wider class of protocols than were first authorized and to remove some of the original limitations.

The contributions of this thesis focuses in two directions. First we extend the soundness result in order to be able to use various cryptographic primitives: asymmetric encryption, symmetric encryption, digital signature and hashing. Second, three possible extensions are proposed: addition of modular exponentiation, unbounded number of challenges (with adaptive corruptions) and handling of opacity properties.

1.4.1 Soundness of Formal Analysis

First, we focus on removing some limitations inherent to [MW04c]. For this purpose, we introduce a formalism in order to describe security criteria like IND-CCA. We then propose a partition theorem which allows, under some conditions, to break a criterion into multiple smaller criteria. Finally, using newly defined criteria, we prove a computational soundness result for protocols involving multiple different cryptographic primitives.

Criterion Formalism The first contribution of this thesis is to define formally what is a security criterion. This formalism can be used to represent most of the security criteria present in the literature like IND-CCA or selective forgery. Moreover, it is easy to build new criteria, either from scratch or by composing previously existing criteria. In particular, we define a new criterion for hashing (which is satisfied in the ROM by using the Random Oracle as a hash function). We also create new criteria that mix IND-CCA for asymmetric encryption, IND-CPA for symmetric encryption and selective forgery for digital signature and for symmetric encryption. This criterion represents joint security of the various primitives.

Criterion Partition Theorem By using patterns, we are able to define security criteria extending classical criteria. Instead of just allowing bit-strings in requests, patterns can be used to insert secret information under encryptions as soon as this does not create encryption cycles. In order to prove equivalence between these new criteria and classical ones, we propose two *criterion partition theorems*. These theorems can be used to prove joint security of multiple cryptographic primitives using security for each of the primitive. This can only be done under some acyclicity restrictions.

Linking Computational and Symbolic Security Finally, using our new criteria, we are able to prove computational soundness of security protocols involving asymmetric encryption, symmetric encryption and digital signature. The main result is that if a protocol is secure in the symbolic model, then it is also secure in the computational model. Properties considered here are trace properties but we also study nonce strong secrecy (SecNonce) and a variant adapted to keys called SecKey.

1.4.2 Extensions

We propose three possible extensions for the previous results. These extensions are orthogonal, hence we did not try to merge them.

Modular Exponentiation The first extension consists in adding modular exponentiation as a cryptographic primitive. Modular exponentiation is typically used as in a Diffie-Hellman key exchange scheme. The usual assumption concerning Diffie-Hellman is the Decisional Diffie-Hellman problem (DDH). We propose a dynamic polynomial extension of DDH called 3DH and prove it equivalent to DDH. Using this, we are able to prove computational soundness of a symbolic equivalence relation defined as an extension of the equivalence relation from [AR00].

Unbounded Challenges Previous chapters only involved a bounded number of sessions. We propose to extend this to the case of an unbounded number of sessions. This is done by defining new criteria holding a polynomial number of challenges and by giving a partition theorem for such criteria. In particular, a nice side effect of unbounded criteria is to directly handle adaptive corruptions without having to suppose non-commiting encryption as in [CFGN96].

Opacity Finally, we look at more complex properties than authentication or strong secrecy. Thus we define symbolic opacity in a very general framework and provide some decidability results. In particular, we specialize this notion in the case of an eavesdropper observing a protocol unfolding and prove that symbolic opacity implies computational indistinguishability, this can be seen as an extension of the original Abadi-Rogaway result to more complex properties than just equivalence.

1.5 Organization of this Document

The first two chapters introduce the basic definitions needed in the symbolic and computational settings.

In chapter 2, we present our symbolic model for protocols. This model is used to describe protocols that can be interpreted afterward with the symbolic or computational semantics. Symbolic semantics are also given in this chapter.

In chapter 3, we recall some preliminaries necessary for the computational setting. In particular, we define polynomial time probabilistic Turing machine and the different cryptographic primitives. We also give classical criteria for usual cryptographic primitives are given. Then we show how proofs are typically performed on these criteria. These proofs could be done more quickly by using the partition theorems introduced in the following.

The following two chapters properly define security criteria, state the partition theorems and show how these theorems can be applied. Then the following chapter uses these results to prove computational soundness of symbolic protocol analysis.

In chapter 4, we give a standard formalism that can be used to properly define classical security criteria. Examples of classical criteria are given, we also introduce new criteria: these criteria either allow encryption of secret information or represent joint security of multiple primitives. The objective of the next chapter is to prove that the new criteria are equivalent to classical ones.

In chapter 5, our two partition theorems are proven using an argument close to the commonly used hybrid argument. These theorems can be used to relate different criteria without having to design new adversaries as this is usually done. We apply our partition theorems to criteria defined

previously. Hence, we are able to prove that security of our combined criteria is implied by classical security assumptions.

In chapter 6, we make the link between the symbolic and computational analysis of protocols. First the computational semantics of security protocols are introduced. We then prove that computational adversaries cannot produce traces that are not possible in the symbolic setting with non negligible probability. We assume some restrictions over protocols and some security requirements for cryptographic primitives. Finally, we use the previous result in order to prove that if some properties are verified by a protocol in the symbolic setting, then computational version of these properties are verified in the computational setting. In particular, this is the case for all trace properties like authentication but also for secrecy of nonces and keys.

Finally, the last three chapters describe possible extensions of the previous link. Each of these chapters can be read independently from the other two.

In chapter 7, modular exponentiation is added as a cryptographic primitive. This allows us to extend our results to Diffie-Hellman like protocols. For this purpose, we introduce a new security criteria for modular exponentiation called 3DH and prove it equivalent to DDH. We believe that the 3DH criterion is interesting on its own as it can be used to verify protocols directly in the computational setting.

In chapter 8, previous definitions are generalized to the case of an unbounded number of challenges. As adversaries are polynomial time, only a polynomial number of challenges can be used. We introduce criteria generalizing classical criteria to an unbounded number of challenges and add oracles to handle adaptive corruption. Generalization of our partition theorems allows us to prove computational soundness results even within the adaptive corruption setting.

In chapter 9, we define opacity in the symbolic setting. Opacity is the symbolic version of indistinguishability: a predicate is opaque if an adversary cannot guess its value from some observations. We define various flavor of opacity and investigate their decidability. We also prove that in the passive setting, symbolic opacity and security of the cryptographic primitives imply computational indistinguishability.

Part I

Computational and Symbolic Aspects of Cryptography

Chapter 2

Symbolic Model for Security Protocols

Contents

In this chapter, we introduce the symbolic model for security protocols. This model has first been described by Dolev and Yao in [DY83] and has been well studied from then (see for example [Rya01], [CJM98], [CLC03], [CLC02] or [GL00]). Its main advantage is that it is possible to perform fully automatic verification of protocols [BLP03a, Low97a, DM00] within this model. However to allow this, the symbolic model abstracts many things like the underlying cryptographic primitives or the way random numbers are generated. For example, the "perfect cryptography hypothesis" implies that it is impossible from a cipher-text to deduce any information on the encoded plain-text without knowing the secret key.

Messages are a key concept in this modeling. They represent the information that circulates between the different agents during the protocol execution. Whereas in the real world (and in the computational model) messages are bit-strings, here messages are first order terms. Constants can be nonces, keys or agents identities. Functions are concatenation, asymmetric and symmetric encryption or digital signature. Chapter 7 also considers modular exponentiation.

During the protocol execution, honest agents exchange messages according to a protocol specification. The *intruder* is able to interfere with this exchange in order to gain some information,

authenticate as another agent or perform any other type of attack. The intruder is defined by a deduction relation which represents the messages that the intruder can deduce when he has observed some other messages. In this model, the intruder has a complete control over the network, he observes all the messages that are exchanged by honest agents: whenever an agent sends a new message, this message becomes part of his knowledge. Whenever he wants to, he can prevent a message from reaching its destination. He can also forge new messages (according to his knowledge and his deduction relation) and send them to an agent impersonating another agent.

A security protocol is composed of different *roles*. These roles are executed by different agents (although the same agent can play different roles simultaneously). Parallel composition of the roles detailed by a protocol is called a *session* of a protocol. In order to differentiate two occurrences of a same role, role have some parameters which are generally instantiated by identities or some keys. Multiple sessions of the same protocol can occur simultaneously. For example, if we consider a protocol with two roles R_1 and R_2, it is possible to execute in parallel a session where agent A has role R_1 and B has role R_2 with a session where A has role R_2 and C has role R_1.

2.1 Terms and Messages

In order to formally describe messages, we first introduce three disjoint sets of constants which are also called atomic messages. These three sets are infinite and countable.

1. $\mathcal{AN}$ contains *agent names*. These represent the identities of the different agents. Typical agent names are A_1, A_2...
2. $\mathcal{N}$ contains *nonces*. Nonces can be thought as random numbers. As it is impossible to guess the value of a nonce, it could be used to ensure freshness of a message. Nonces are usually denoted by N_1, N_2...
3. $\mathcal{K}$ contains *keys* which are used by the different cryptographic primitives. As we consider three different cryptographic primitives, there are several different sorts of keys. For the asymmetric encryption scheme, there are two sorts of keys: $PubK$ for public keys and $PrivK$ for private keys. For the symmetric encryption scheme, the only sort is $SymK$. For the digital signature scheme, there is a sort $SignK$ for secret signature keys and a sort $VerifK$ for public verification keys.

As we want to be able to associate a matching pair of keys, there is a bijection from $PubK$ to $PrivK$ and from $VerifK$ to $SignK$ associating each key k to its inverse k^{-1}. This bijection is extended from $PrivK$ to $PubK$ and from $SignK$ to $VerifK$ by taking $k^{-1^{-1}} = k$. It is also extended to $SymK$ by $k^{-1} = k$. Therefore, the inverse function is defined for any key.

The set of atomic messages $\mathcal{AN} \cup \mathcal{N} \cup \mathcal{K}$ is denoted by $\mathcal{AM}$. In order to define the set $\mathcal{T}$ of terms, we introduce some function symbols that can be used to compose atomic messages.

1. $\mathsf{enc}_a : \mathcal{T} \times PubK \to \mathcal{T}$
   The term $enc_a(t, k)$ is also denoted $\{t\}_k^a$, it represents the asymmetric encryption of term t using key k.
2. $\mathsf{enc}_s : \mathcal{T} \times SymK \to \mathcal{T}$
   The term $enc_s(t, k)$ is also denoted $\{t\}_k^s$, it represents the symmetric encryption of term t using key k.
3. $\mathsf{enc}_g : \mathcal{T} \times SignK \to \mathcal{T}$
   The term $enc_g(t, k)$ is also denoted $\{t\}_k^g$, it represents the digital signature of term t using signature key k.
4. $\mathsf{pair} : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$
   $\langle t_1, t_2 \rangle$ is used for $\mathsf{pair}(t_1, t_2)$, this denotes the concatenation of term t_1 and term t_2. The tuple $\langle t_1, \ldots, t_n \rangle$ is defined using the pair operator as term $\mathsf{pair}(t_1, \ldots, \mathsf{pair}(t_{n-1}, t_n))$

Let $\mathcal{X}$ be an infinite countable set of variables (disjoint from all the previously introduced sets). First order terms over the previous signature are referred to as *message terms*. Ground terms, i.e. variable free terms, are called *messages*.

We define the function *vars* over message terms such that $vars(t)$ returns all the variables that occur in t.

2.2 Description of Security Protocols

Protocols are usually described using a simple syntax that gives the sequence of messages exchanged in a single session. This way of specifying protocols has first been introduced in [BAN96] and then has been widely used in latter theoretical works about cryptographic protocols like the Clark and Jacob survey [CJ97]. For example, the classical Needham-Schroeder public key protocol [NS78] is described by:

$$\begin{array}{rcl} A \rightarrow B & : & \{\langle A, N_A \rangle\}_{pk_B} \\ B \rightarrow A & : & \{\langle N_A, N_B \rangle\}_{pk_A} \\ A \rightarrow B & : & \{N_B\}_{pk_B} \end{array}$$

This description does not make any difference between roles and identities. There are two roles A and B. The protocol has three steps: in the first one, A sends to B a message which is the encryption of its identity (A) and a nonce (N_A) by the public key of B (pk_B). In the second step, after receiving this message B answers with the encryption of the nonce sent by A and a fresh nonce (N_B) using key pk_A. Finally when A receives this last message, he answers nonce N_B encrypted using pk_B. The main advantage of this BAN syntax is that it is easy to read, however there are some drawbacks that makes it unsuitable for our purpose. For example as the protocol description does not use variables, assignment of the different parameters of a role is not clear: in the second step, B sends a message encoded by pk_A, does it always use that key or is it because B just received a message containing identity A ?

2.2.1 Roles

A role is the description of the actions performed by an agent in a protocol. During the protocol execution, an agent may receive some messages, accept them if they match a given pattern, send other messages and verify some signatures. We detail here a very simple syntax for roles that does neither include tests nor control points. The syntax is kept as restricted as possible so that further results given in this document are easier to understand and prove, however most of our results can be adapted when considering a more expressive role model.

Formally, a role is a finite sequence of actions. Actions and roles are defined by the following grammar:

$$\begin{array}{rcl} action & ::= & \textsc{Recv}(t) \mid \textsc{Send}(t) \mid \textsc{Veri}(t_1, t_2, t_3) \\ role & ::= & action.role \mid \epsilon \end{array}$$

Where t, t_1, t_2 and t_3 range over message terms. Each action has an intuitive signification.

1. $\textsc{Recv}(t)$: reception of a message that matches term t, this can affect the value of some variables from t for the rest of the role.

2. $\textsc{Send}(t)$: emission of a message t, all variables in t must have been assigned to before.

3. $\textsc{Veri}(t_1, t_2, t_3)$: this last action is related to signature, it tests that t_2 is a valid signature of t_1 using verification key t_3. t_1, t_2 and t_3 must have been assigned to before.

Usually roles depend on several arguments (e.g. the identity of the agent that plays the role in the protocol, its public and secret keys). Thus, instead of using directly roles, we use functions from tuples of messages to roles. These functions are called *parametrized roles*. A role can be seen as a parametrized role that takes no arguments.

Example 2.1 *The Needham-Schroeder protocol given above contains two parametrized roles, R_1 (played by agent A in the previous description) and R_2 (played by B in the previous description):*

$$R_1(x_A, x_{pkA}, x_B, x_{pkB}) : \quad \text{SEND}(\{x_A, N_A\}_{x_{pkB}}) \,.\, \text{RECV}(\{N_A, y\}_{x_{pkA}}) \,.\, \text{SEND}(\{y\}_{x_{pkB}})$$
$$R_2(x_B, x_{pkB}, x_A, x_{pkA}) : \quad \text{RECV}(\{x_A, z\}_{x_{pkB}}) \,.\, \text{SEND}(\{z, N_B\}_{x_{pkA}}) \,.\, \text{RECV}(\{N_B\}_{x_{pkB}})$$

2.2.2 Protocols and Scenarios

A protocol definition contains a list of roles, but this is not sufficient. The protocol also has to make precise which roles have to be executed simultaneously and how their parameters have to be initialized. This is done via scenarios. Once more, the syntax given here is quite restricted as it only allows parallel composition. More expressive syntaxes are commonly used when analysing symbolic security of protocols (including for example sequentialisation). However, we stick to our simplified model. Scenarios are given by the following grammar:

$$scenario ::= prole(m_1, ..., m_n) \| scenario \mid \epsilon$$

Where *prole* is a parametrized role that takes n arguments, therefore $prole(m_1, ..., m_n)$ is a role. Note that it is also possible to use a role instead of a parametrized role, this role will have no arguments in the scenario's description. A single scenario can contain multiple sessions executed simultaneously as the same role can occur more than once in the scenario.

Example 2.2 *We still consider the Needham-Schroeder protocol. The famous "man-in-the-middle" attack [Low95, Low96] uses two parallel sessions, one involving an honest agent A and the intruder C, the other one involving A and another honest agent B. Therefore, the related scenario is:*

$$S = R_1(A, pk_A, C, pk_C) \| R_1(A, pk_A, B, pk_B) \| R_2(B, pk_B, A, pk_A)$$

Notice that the role R_2 instantiated by the intruder C is not present in this scenario as the intruder has total control over the network (details on this point appear in section 2.4). Moreover role R_1 from the session between A and B is useless in this attack thus it is possible to define a simpler scenario on which the attack can still be performed.

$$S = R_1(A, pk_A, C, pk_C) \| R_2(B, pk_B, A, pk_A)$$

A scenario defines how the different roles are instantiated during the protocol execution. However, it does not tell us which information are known by the intruder and which are not. For example, an intruder may share a key with an honest agent in a session and have no information on a key used by two honest agents in another session. Therefore, a protocol description contains a scenario and a finite set of atoms from that scenario. This set represents the initial knowledge of the intruder. First, we introduce the function *atoms* defined on message terms, roles and scenarios that returns the atoms that the intruder may know in a message term, in a role or in a scenario. This function returns all the atoms that appear in its argument or whose inverses appear in its argument.

Let t be a message term, the set $atoms(t)$ contains all the closed atoms a (identities, nonces and keys) from t such that either a or its inverse a^{-1} (in the case of keys) appears in t.

The set $atoms(R)$ is inductively defined on roles by:

$$\begin{aligned} atoms(\epsilon) &= \emptyset \\ atoms(\text{RECV}(t).R) &= atoms(t) \cup atoms(R) \\ atoms(\text{SEND}(t).R) &= atoms(t) \cup atoms(R) \\ atoms(\text{VERI}(t_1, t_2, t_3).R) &= atoms(t_1) \cup atoms(t_2) \cup atoms(t_3) \cup atoms(R) \end{aligned}$$

The set $atoms(S)$ is inductively defined on scenarios by:

$$\begin{aligned} atoms(\epsilon) &= \emptyset \\ atoms(R(m_1,...,m_n)\|S) &= \Big(\bigcup_{1\leq i\leq n} atoms(m_i)\Big) \cup atoms(R) \cup atoms(S) \end{aligned}$$

Then a protocol is composed of a scenario S and a subset IK of $atoms(S)$. The set IK represents the initial knowledge of the intruder. It usually contains all the identities and public keys (for asymmetric encryption and digital signature) from scenario S, IK also usually contains secret keys related to dishonest agents and symmetric keys that are shared with a dishonest agent.

Example 2.3 *It is now possible to define formally the Needham-Schroeder protocol. Notice that we do not really define the exact Needham-Schroeder protocol but rather a protocol that involves two sessions of Needham-Schroeder. This protocol* Π *is defined by* $\Pi = (S, IK)$ *where* S *has already been defined in example 2.2 and* IK *contains identities* A*,* B *and* C*, public keys* pk_A*,* pk_B *and* pk_C *and the secret key of the intruder* $sk_C = pk_C^{-1}$*.*

2.3 Protocol Semantics

In the symbolic setting, there are two different semantics for security protocols. Semantics for a "normal" execution of the protocol are described in this section whereas semantics for execution of the protocol opposed to an active intruder are detailed in section 2.4. Before introducing the semantics, we first define what kind of protocols we consider as there are protocols that are not executable.

2.3.1 Executable Protocol

A role is said to be *executable* if it fulfills some simple conditions. First any variable that is used in the protocol has to be defined (by a reception or as a parameter of the role) before being used in any emission or signature verification. Second, signature verification can only have a very specific form. Then a protocol is said executable if it only contains executable roles. Let us formulate these restrictions in more details. Π is executable if for any parametrized role $R(x_1,...,x_n)$ in Π that has the form $inst_1 \cdot inst_2 \cdots inst_n$,

1. If $inst_i = \text{SEND}(t_1)$ and a variable x occurs in $vars(t_1)$, then there exists $j < i$ such that $inst_j = \text{RECV}(t_2)$ and x occurs in $vars(t_2)$ or there exists k such that $x = x_k$ (i.e. x is a parameter of the role).
2. If $inst_i = \text{VERI}(t_1, t_2, t_3)$ and a variable x occurs in $vars(t_1)$ or in $vars(t_2)$ or in $vars(t_3)$, then there exists $j < i$ such that $inst_j = \text{RECV}(t_4)$ and x occurs in $vars(t_4)$ or there exists k such that $x = x_k$.
3. If $inst_i = \text{VERI}(t_1, t_2, t_3)$, then t_2 is a variable and t_3 is either a variable or a signature verification key.

The first two restrictions are classical: in order to be able to execute a program, variables must be properly defined before sending them, thus any variable that occurs in an emission or a verification statement must have been initialized before by a reception statement. The last restriction is intuitive, there is no use in testing that a message is a signature if this message has just been forged by the same agent, therefore the second parameter must be a variable. As the last parameter stands for the verification key, it has to be either a variable (if the agent received in a previous message the verification key) or a constant verification key.

We only put restrictions on variables, however we do not ensure that an agent has the necessary knowledge in order to execute its role. For example, to execute $\text{RECV}(\{x\}^a_{pk})$ an agent has to be

able to deduce the inverse key of pk. We are not interested in these conditions as we do not ask protocols to be implementable in the real life but rather want to ensure that simulation of the protocol is possible.

2.3.2 Executing a Protocol

The execution of a protocol produces a trace. As this is principally what an outside observer can see from the execution, a trace is a role that contains only emissions and receptions of messages, signature verifications cannot be observed by an outside observer.

Definition 2.1 *Traces are given by the following grammar where m ranges over (closed) messages.*

$$\begin{array}{rcl} trace & ::= & \epsilon \\ & | & \textsc{Send}(m).trace \\ & | & \textsc{Recv}(m).trace \end{array}$$

Therefore, a trace is a finite list of actions which can either be emission or reception of a (closed) message.

A protocol has multiple possible traces depending on which interleaving of the different role is chosen. The initial knowledge does not play any role in defining possible traces as we only consider passive observers. Semantics are defined by a rewriting systems between scenarios. Rewriting $S \longrightarrow \langle t \rangle S'$ denotes that the trace t is possible for scenario S and leads to scenario S'

In order to keep the semantics simple, the parallel composition is assumed to be an associative and commutative operator in the following three rules. The first rule allows an agent to send a message, even if this message is not received by any other agent.

$$\overline{\textsc{Send}(m).R \| S \longrightarrow \langle \textsc{Send}(m) \rangle R \| S}$$

The second rule connects a send action to a receive action. The emission of the message has to occur before the reception, emission and reception are not synchronized. Let t be a message term and m be a message. Let σ be the most general unifier of t and m, then σ is applied to the rest of the role where the reception is performed (variables are not shared between the different roles executed in parallel).

$$\frac{S \longrightarrow \langle \textsc{Send}(m) \rangle \textsc{Recv}(t).R \| S'}{S \longrightarrow \langle \textsc{Send}(m) \textsc{Recv}(m) \rangle R\sigma \| S'}$$

The last rule defines the behavior of signature verification. Every message has to be instantiated and of course we ask the second message to be a signature of the first one. The signature key has to be the inverse of the third message. In this case, no action is output in the trace.

$$\overline{\textsc{Veri}(m, \{m\}_k^g, k^{-1}).R \| S \longrightarrow \langle \epsilon \rangle R \| S}$$

Then an execution of a scenario S is a sequence of rewriting:

$$S \longrightarrow \langle t_1 \rangle \cdots \longrightarrow \langle t_n \rangle S'$$

The corresponding trace is $t_1 \cdots t_n$. It is now easy to define the set of possible traces for a protocol Π whose attached scenario is S.

Definition 2.2 *Let Π be a protocol (S, IK). Then the set of possible traces corresponding to correct executions (i.e. no intruder except a passive eavesdropper) $ctraces(\Pi)$ is defined by:*

$$ctraces(\Pi) = \left\{ t_1 \cdots t_n \middle| \exists S_1, \cdots, S_n \text{ such that } S \longrightarrow \langle t_1 \rangle S_1 \cdots \longrightarrow \langle t_n \rangle S_n \right\}$$

Note that possible traces for a protocol do not depend on the initial knowledge of this protocol. Hence, we also use notation $ctraces(S)$ instead of $ctraces(S, IK)$.

These semantics authorize that a role sends a message to itself. For example, a possible trace for scenario $\text{SEND}(a).\text{RECV}(x)$ is $\text{SEND}(a).\text{RECV}(a)$. The only other possible trace is $\text{SEND}(a)$.

Example 2.4 *Consider the following scenario S where roles R_1 and R_2 have been properly defined in previous examples and represent participants to the Needham-Schroeder protocol.*

$$S = R_1(A, pk_A, B, pk_B) \| R_2(B, pk_B, A, pk_A)$$

This scenario represents the correct behavior of the protocol involving two honest agents A and B. Then a possible trace is:

$$\begin{aligned}
&\text{SEND}(\{A, N_A\}_{pk_B}).\text{RECV}(\{A, N_A\}_{pk_B}).\\
&\text{SEND}(\{N_A, N_B\}_{pk_A}).\text{RECV}(\{N_A, N_B\}_{pk_A}).\\
&\text{SEND}(\{N_B\}_{pk_B}).\text{RECV}(\{N_B\}_{pk_B})
\end{aligned}$$

2.3.3 Interleavings and Simplified Protocols

The main complexity of the semantics introduced in the previous section comes from the number of possible interleavings. Thus if we consider a scenario that is composed of a single role its semantics can be expressed in a much simpler way. We first define possible substitutions for a role R. Intuitively, a substitution σ is possible if the trace obtained by applying σ to R and deleting verifications is correct.

As multiple receives in a sequence may block the execution of a role, we only consider *alternated* roles. Let R be a role $a_1 \cdots a_n$. Then R is alternated if each reception corresponds to exactly one emission. Formally for any i between 1 and n where a_i is a reception, let j be the maximal index lower than i such that a_j is an emission, then for all the indexes strictly between j and i the corresponding action is a signature verification. Moreover, we ask the first reception to occur after an emission and the last emission to occur before a reception. Hence role R has the form

$$\text{VERI}()^*.\text{SEND}(m_1).\text{VERI}()^*.\text{RECV}(n_1).\text{VERI}()^*.\text{SEND}(m_2)\ldots\text{RECV}(n_u).\text{VERI}()^*$$

Then a substitution σ is possible for a length k if for any $i \leq k$:

1. If $a_i = \text{VERI}(t_1, t_2, t_3)$, then $t_2\sigma = \{t_1\sigma\}^g_{t_3\sigma}$.
2. If $a_i = \text{RECV}(t)$, let j be the greatest integer lower than i such that $a_j = \text{SEND}(t')$, then $t'\sigma = t\sigma$. Moreover, there does not exist any k between j and i such that a_k is a receive action.

Then the set $ctraces'$ of possible traces is obtained by projection of $a_1\sigma \cdots a_k\sigma$ on emissions and receptions, i.e. one has to cut the role to length k, apply σ and delete the VERI() statements.

Proposition 2.1 *For any alternated role R, the two sets of possible traces $ctraces(R)$ and $ctraces'(R)$ are equal.*

Proof: This proof is performed using an induction on the number u of emissions/receptions of R.

1. If $u = 0$, R only contains signature verifications, as the protocol is executable R does not contain any variable. Therefore $ctraces'(R)$ contains the possible prefixes of R. Moreover rewritings R can only produce the prefixes of R, hence $ctraces(R)$ is equal to $ctraces'(R)$.
2. Else, R has the following form:

   $$R'.\text{SEND}(m_u).\text{VERI}()^*.\text{RECV}(n_u).\text{VERI}()^*$$

   where role R' involves $u - 1$ emissions/receptions. The induction hypothesis applies to R', thus $ctraces(R')$ and $ctraces'(R')$ are equal. In both semantics, new traces can be obtained

by adding SEND(m) where m is the only possible instantiation of σ (there is only one such instantiation as R is executable) and by adding SEND(m).RECV(m) if all the signature verifications hold. Hence we get that $ctraces(R)$ and $ctraces'(R)$ are equal.

■

The number of possible traces for a role R can be bounded by the length of R.

Proposition 2.2 *Let $R = a_1 \cdots a_n$ be a role, then*

$$|ctraces'(R)| \leq n + 1$$

Proof: This follows directly from the definition of $ctraces'$. ■

Let S be a scenario $R_1 \| \cdots \| R_n$ where each R_i is the role $a_1^i \cdots a_{n_i}^i$. Then a possible (but maybe incomplete) interleaving for S is a role defined by a function ψ from $[1, k]$ to $[1, n]$ such that k is lower than $n_1 + \cdots + n_n$. This function tells which role of S is supposed to be executed at the j^{th} step ($\psi(j)$). The corresponding role is denoted by $IL(S, k, \psi)$. The IL function can be recursively defined by:

$$\begin{aligned} IL(R_1 \| \cdots \| R_n, 0, \psi) &= \epsilon \\ IL(R_1 \| \cdots \| R_n, k, \psi) &= a_1^{\psi(i)}.IL\Big(R_1 \| \cdots \| a_2^{\psi(i)} \cdots a_{n_{\psi(i)}}^{\psi(i)} \| \cdots \| R_n, k-1, x \to \psi(x-1)\Big) \end{aligned}$$

The set of possible interleavings for a scenario S is denoted by $IL(S)$, it contains every $IL(S, k, \psi)$ for any possible values of k and ψ. Knowing the set of possible interleavings of a scenario, the set of possible traces for any protocol with this scenario can be obtained as the union of the possible traces for each interleaving.

Proposition 2.3 *Let S be a scenario, then*

$$ctraces(S) = \bigcup_{R \in IL(S)} ctraces'(R)$$

This last proposition gives us the idea that in some cases, it is possible to consider protocols with a single role without loss of generality. Namely if one wants to prove that a property ϕ holds on any possible traces of a scenario that verifies ϕ', then one just have to verify that this is true for single role protocols and that if ϕ' is true for a scenario, it is also true for any of its possible interleavings.

Protocols with a single role are called *linear protocols*. We proved here that semantics in the case of an eavesdropper can be related to semantics of the possible interleavings. This can be generalized to the semantics for an active intruder. In order to prove that, we first have to introduce the active intruder model.

2.4 The Intruder Model

In this section, we describe the classical model of intruder introduced by Dolev and Yao in [DY83]. In the symbolic model, there is a single adversary that attacks the protocol. This adversary is called the intruder. It has total control over the network, therefore it can intercept any message sent over the network and prevent it from reaching its destination. It can also forge new messages using its deduction relation and send these messages to some agent using another agent's identity. There are no restrictions on the time used by the intruder and on the size of the messages he can produce. However as cryptography is perfect, the intruder cannot deduce the content of an encrypted message without knowing the decryption key and cannot guess any nonce.

2.4.1 The Deduction Relation

The intruder is represented by the set of messages he knows which is denoted by E. We do not suppose here that E is finite as an intruder can generate an unbounded number of fresh nonces. At the beginning of the protocol execution, the intruder has an initial knowledge (for example identities and public keys). Whenever a message is sent by an agent, this message is added to the intruder's knowledge. Whenever an agent waits for a message, the intruder can give any deducible message.

The deduction relation defines the messages an intruder can produce from the set of messages E, it is denoted using the entailment relation $\vdash$. Let m be a message, $E \vdash m$ reads as message m is deducible from E. This relation is inductively defined using the following rules:

1. If $m \in E$, then $E \vdash m$.
2. If $E \vdash m_1$ and $E \vdash m_2$, then $E \vdash \langle m_1, m_2 \rangle$.
3. If $E \vdash m$ and $E \vdash pk$ where $pk \in PubK$, then $E \vdash \{m\}^a_{pk}$.
4. If $E \vdash m$ and $E \vdash k$ where $k \in SymK$, then $E \vdash \{m\}^s_k$.
5. If $E \vdash m$ and $E \vdash sik$ where $sik \in SigK$, then $E \vdash \{m\}^g_{sik}$.
6. If $E \vdash \langle m_1, m_2 \rangle$, then $E \vdash m_1$ and $E \vdash m_2$.
7. If $E \vdash \{m\}^a_{pk}$ and $E \vdash pk^{-1}$, then $E \vdash m$.
8. If $E \vdash \{m\}^s_k$ and $E \vdash k$, then $E \vdash m$.
9. If $E \vdash \{m\}^g_{sik}$, then $E \vdash m$.

The intruder can deduce any message that appears in his knowledge, he can compose messages using pairing, encryption or signature. He can also perform projections on messages or decrypt them if he knows the related secret key (for asymmetric encryption) or the encryption key (for symmetric encryption). Finally, as usual algorithms for digital signature do not ensure secrecy of the signed message, we consider signature with message recovery: the intruder can deduce from a signature which message was signed.

2.4.2 Semantics with an Intruder

The second possible symbolic semantics for a protocol is semantics where an intruder can interact with the protocol. This semantics with an intruder defines a new set of possible traces denoted by $traces(\Pi)$ for a protocol Π. Note that, whereas the set of honest traces does not depend on the initial knowledge set of Π, this set of traces heavily depends on the initial knowledge.

Parallel composition is still assumed to be an associative and commutative operator in the following. The semantics is described by a new rewriting system $\longrightarrow$. This rewriting system is defined over protocols as the knowledge of the intruder is modified during the execution: in a protocol (S, K), S represents the current scenario of the protocol (which is the state of honest agents) and K represents the knowledge of the intruder (i.e. his state). The interpretation of $(S, K) \longrightarrow \langle t \rangle (S, K')$ is that from scenario S, if the intruder has knowledge K, he can make the scenario evolve to S' and his new knowledge will be K', this modification produces a trace t.

The first rule allows an agent to send a message, even if this message is not received by any other agent. We consider that the intruder intercepts this message, that is why this message is appended to the knowledge of the intruder.

$$\overline{\big(\textsc{Send}(m).R \| S, K\big) \longrightarrow \langle \textsc{Send}(m) \rangle \big(R \| S, K \cup \{m\}\big)}$$

The second rule describes the behavior of the receive statement. The received message is forged by the intruder using its deduction relation. If t and t' are two message terms that can be unified, let $mgu(t,t')$ be their most general unifier.

$$\frac{K \vdash m \qquad \sigma = mgu(m,t)}{\big(\textsc{Recv}(t).R \| S, K\big) \longrightarrow \langle \textsc{Recv}(m)\rangle\big(R\sigma \| S, K\big)}$$

The last rule defines the behavior of signature verification, it is the same rule as for "normal executions", i.e. every message has to be instantiated and the second message must be a signature of the first one with the inverse of the third message. In this case, no action is output in the trace and the intruder knowledge is not modified.

$$\frac{}{\big(\textsc{Veri}(m, \{m\}_k^g, k^{-1}).R \| S, K\big) \longrightarrow \langle \epsilon \rangle \big(R \| S, K\big)}$$

Then an execution of a scenario S is a sequence of rewriting:

$$S \longrightarrow \langle t_1 \rangle \cdots \longrightarrow \langle t_n \rangle S'$$

The corresponding trace is $t_1 \cdots t_n$. It is now easy to define the set of possible traces for a protocol Π.

Definition 2.3 *Let Π be a protocol (S, IK). Then the set of possible traces corresponding to executions with an intruder $traces(\Pi)$ is defined by:*

$$traces(\Pi) = \big\{ t_1 \cdots t_n \big| \exists (S_1, K_1), \cdots, (S_n, K_n) \text{ such that } \Pi \longrightarrow \langle t_1 \rangle (S_1, K_1) \cdots \longrightarrow \langle t_n \rangle (S_n, K_n) \big\}$$

Example 2.5 (Needham-Schroeder) *We exemplify the semantics by detailing the well-known attack on the Needham-Schroeder protocol. We consider two honest agents A and B, the intruder is denoted by C. To perform the attack, we assume that there are in parallel a session between A and C and another session between A and B, role R_1 in the second session is not described here as it is useless for the attack. Scenario S is defined by:*

$$S = R_1(A, pkA, C, pkC) \| R_2(B, pkB, A, pkA)$$

The instantiated versions of roles R_1 and R_2 are:

$$\begin{aligned} R_1(A, pkA, C, pkC): &\quad \textsc{Send}(\{A, N_A\}_{pkC}) \,.\, \textsc{Recv}(\{N_A, y\}_{pkA}) \,.\, \textsc{Send}(\{y\}_{pkC}) \\ R_2(B, pkB, A, pkA): &\quad \textsc{Recv}(\{A, z\}_{pkB}) \,.\, \textsc{Send}(\{z, N_B\}_{pkA}) \end{aligned}$$

Then the attack on the Needham-Schroeder protocol can be obtained using the previous semantics.

Simple Properties of the Semantics First, there is a simple link between the "normal" semantics and the semantics with an intruder. The intruder can decide to correctly forward the messages. In this case, the produced traces are also valid traces for "normal" semantics.

Proposition 2.4 *For any protocol Π, the set of traces for "normal" executions is included in the set of traces for execution with intruder, i.e.:*

$$ctraces(\Pi) \subseteq traces(\Pi)$$

Proof: This comes directly because if m appears in K, then m is deducible from K. ∎

As before, it is sufficient to consider the different possible interleavings of the scenario:

Proposition 2.5 *Let S be a scenario and IK be an initial knowledge, then*

$$traces(S, IK) = \bigcup_{R \in IL(S)} traces(R, IK)$$

This last proposition also leads us to think that it is sufficient to consider linear protocols. In the rest of this document except in the remaining of this chapter, we only consider linear protocols. However we strongly believe that our results can be generalized to protocols using parallel composition. This restriction allows us in particular to keep the computational semantics as simple as possible.

2.5 Protocol Properties

Here, we present some properties over protocols that can be formulated and verified within our model. Some of them can be verified by fully automatic tools as we only consider a bounded number of roles. These properties correspond to the intuitive definition of security for a protocol. What does it mean that a protocol is secure ? Depending on the objective of the protocol, it may mean that an intruder can never get some information (for example a secret key), this is secrecy. It may also mean that whenever an agent reaches some point in the execution of the protocol, this agent can be sure that he is talking to another honest agent, this property is called authentication. More complicated properties based on *opacity* are detailed in chapter 9.

2.5.1 Secrecy

Let Π be a protocol composed of a scenario S and a role R. A message m is kept secret by Π if it is not possible that after executing the protocol, the intruder can deduce message m. An important remark is that if any of the atoms of m does not occur neither in S nor in IK, then m is not deducible by the intruder. Hence we only considers message m such that every atom of m occur in S or in IK.

Definition 2.4 (Secrecy) *A protocol $\Pi = (S, IK)$ preserves secrecy of message m if for all $act_1(m_1) \cdots act_n(m_n)$ in traces(Π), it is not possible to deduce m from the initial knowledge and the exchanged messages:*

$$IK, m_1, \ldots m_n \not\vdash m$$

The intuitive meaning of secrecy is that a protocol does not preserve secrecy of message m if there exists a way for the intruder to run the protocol in order to deduce m. The intruder chooses which messages to exchange, he can also choose the interleaving.

Proposition 2.6 *Let $\Pi = (S, IK)$ be a protocol that does not preserve secrecy of a message m, then there exists R in $IL(S)$ such that protocol (R, IK) does not preserve secrecy of m. The reciprocal is also true: if a protocol has an interleaving that does not preserve secrecy, then the whole protocol does not preserve secrecy.*

Proof: This is a direct consequence of proposition 2.5. ∎

A classical result is that for bounded protocols (which is the case here), secrecy is a decidable problem. This result has been given in [RT01] and in [AL00]. The following proposition is a consequence of the more general results given in section 2.6.

Proposition 2.7 *Let Π be a protocol and m be a message. Determining whether Π preserves secrecy of m is a decidable (and co-NP complete) problem.*

Extensions of this result have been developed recently. The main idea is to generalize this result when adding equational theories like exclusive-or (associativity, commutativity, neutral element, nilpotency) [CKRT03a, CLS03].

2.5.2 A Trace Property: Authentication

The symbolic model presented here does not handle control points, hence authentication cannot be expressed directly. Thus we model authentication as a trace property: if Π is a protocol, then we consider a set of traces for Π that represents executions where authentication is verified.

Then the authentication property is verified for Π in the active setting if there are no traces in *traces*(Π) that does not appear in the previous set of traces.

2.6 Dolev-Yao Constraints

A possible way to prove decidability of secrecy (proposition 2.7) is to consider constraints based on the deduction relation [MS01, Maz04a]. The $\vdash$ operator when used in $E \vdash m$ means that the message m is deducible from knowledge E. Then, a Dolev-Yao constraint is a conjunction of atomic relations involving the $\vdash$ operator. The question of how to build constraints from protocol specifications has widely been discussed before and can be found in [MS01], [CLS03] or in [AL00]. The idea is to test any possible interleavings of the different sessions (which are in finite number as the number of sessions is itself finite). For each interleaving, secrecy is equivalent to a constraint. The main point is that constraints built from a protocol are always in a very specific class of constraints called well-formed constraints. Thus, satisfiability of well-formed constraints has quickly been proven NP-complete but there are no such results for more general constraints.

In this section, we prove that satisfiability remains NP-complete when considering any possible constraint. When verifying a property on a protocol, three distinct constraints appear: the first one specifies the initial state of the intruder (or of any agent), the second describes the unfolding of the protocol and the last one gives the property that we want to prove. The second constraint is well-formed so previous studies omitted the first constraint and forced the last constraint to be a unique $E \vdash m$ constraint where E was the final knowledge of the intruder. Our results allow us to use general constraints for the first and the third constraint. Thus, we can verify properties that cannot be expressed using well-formed constraints only and this allows verification of a larger class of properties.

For the sake of simplicity, we only consider here the case of symmetric encryption. Encryption of m using k is denoted by $\{m\}_k$ in order to shorten notations.

2.6.1 Constraints and their Verification

Constraints are equations where the deduction relation is allowed and denoted by $\Vdash$. The logical and operator is also used to build conjunctions. For example, $pk, a \Vdash x \wedge pk, a, x \Vdash y$ denotes a constraint where x is deducible from pk and a and y is deducible from pk, a and x.

Definition 2.5 (Constraints) *The set DYC of Dolev-Yao constraints is defined by C in the following grammar where T is a finite set of message terms and t is a message term.*

$$\begin{array}{rcl} C & ::= & C \wedge C \mid C \vee C \mid \top \mid \bot \mid C_A \\ C_A & ::= & T \Vdash t \end{array}$$

Given a constraint C, a model of C is a substitution σ that defines any free variable of C such that $C\sigma$ is a true predicate. Note that, in our case we are limited to the atomic key model: keys are atoms and cannot be composed of messages.

Definition 2.6 (Models) *A substitution σ is said to be a* model *for a constraint C iff $C\sigma$ is closed and $\sigma \models C$ where $\models$ is defined using the usual inference rules extended by:*

$$\frac{T\sigma \vdash t\sigma}{\sigma \models T \Vdash t}$$

A model σ is said valid for a constraint C if for any message m in $C\sigma$, m is correctly typed.

Another important definition is well-formed constraints. These constraints are well-known in the protocol analysis community because secrecy for a protocol with a bounded number of sessions is equivalent to satisfiability of a well-formed constraint. To introduce these constraints, two hypotheses need to be made whereas there are no assumption on constraints in DYC. More precisely, a constraint C in DYC is said to be *well formed* iff none of the environment is empty and for any conjunction Co composing C in disjunctive normal form, the two following conditions hold:

- If $T \Vdash t$ and $T' \Vdash t'$ are in Co, then $T \subseteq T'$ or $T' \subseteq T$ (Environment Inclusion).

- If $T \Vdash t \in Co$ and $x \in var(T)$, then there exists $T' \Vdash t' \in Co$ such that $x \in var(t')$ and $T' \subseteq T$ (Variable Introduction).

If constraint C has at least one valid model, constraint C is said to be *satisfiable*. In the rest of this section, we prove that the satisfiability problem is decidable and NP-complete. However, this result is already well-known for well-formed constraints even in the case of composed keys, see for example [AL00, RT01] and even with the XOR operator [CKRT03a, CLS03] or modular exponentiation [CKRT03b]. Satisfiability of these constraints is exactly equivalent to the secrecy problem. The next theorem is a generalisation of this result to general constraints.

Theorem 2.1 *Satisfiability for constraints in DYC is decidable and is a NP-complete problem.*

2.6.2 Proving Decidability

The goal of this section is to prove theorem 2.1. A first result is that if a constraint C is satisfiable, then it has a model whose atoms all occur in C. Let $keys(C)$ denote the keys (atoms and variables) used by a constraint C (the *keys* operation is easily defined on messages and extended to constraints).

Proposition 2.8 *Let C be a constraint in DYC such that $atoms(C)$ is not empty and C is satisfiable, then there exists a model σ of C such that:*

$$\bigcup_{x \in var(C)} atoms(x\sigma) \subseteq atoms(C)$$

So,in particular,

$$\bigcup_{x \in var(C)} keys(x\sigma) \subseteq atoms(C)$$

Proof: As C is satisfiable and $atoms(C)$ is not empty, let σ' be a model of C and a an element of $atoms(C)$. Then let σ'' be the substitution that associates a to each atom of $atoms(C\sigma) \backslash atoms(C)$ and $\sigma = \sigma'\sigma''$. Hence, $atoms(x\sigma) \subseteq atoms(C)$. Moreover, $T\sigma' \vdash m\sigma'$ implies $T\sigma \vdash m\sigma$. ■

The condition that $atoms(C)$ must not be empty is not very restrictive as satisfiability of C is equivalent to satisfiability of $C \wedge a \Vdash a$ for some fresh atom a.

Moreover, as we only use atomic keys, checking satisfiability of a constraint in DYC is equivalent to checking satisfiability of some constraints C in DYC such that $keys(C)$ does not contain any variable.

Proposition 2.9 *Let C be a constraint in DYC such that $atoms(C)$ is not empty, then there exists a subset Γ of DYC such that:*

- *For all C' in Γ, $keys(C') \subseteq \mathcal{AM}$ and there exists σ that associates atoms to $keys(C) \cap \mathcal{X}$ such that $C' = C\sigma$.*
- *$|\Gamma| \leq |card(atoms(C))|^n$ where $n = |keys(C) \cap \mathcal{X}|$.*
- *C is satisfiable iff there exists C' in Γ such that C' is satisfiable.*

Proof: This proof can be made by using an induction on the value of n.

- If $n = 0$, then $\Gamma = \{C\}$ verifies the proposition.
- Else if $n > 0$, let x be an element of $keys(C) \cap X$. Consider Γ' defined by

  $$\Gamma' = \{C[x \backslash a_1], ..., C[x \backslash a_m]\}$$

  Where a_1 to a_m are the atoms occurring in C. Then C is satisfiable iff a constraint $C' \in \Gamma'$ is satisfiable. Hence, the induction hypothesis can be applied to any element in Γ' and this gives the awaited result by taking the union of the constraints sets.

■

Moreover, given a constraint C, the set Γ can be computed. So without loss of generality we can consider only constraints C such that $keys(C) \subseteq \mathcal{A}$, i.e. these constraints only use atoms as keys.

To check satisfiability, our first step is to test all possible unifications. Then we test all the possible orders in which the keys can be compromised. For this purpose, we introduce the notation $T \vdash m[U]$ which means that m is deducible from T using keys in U.

Definition 2.7 *Let T be a finite set of closed messages, m be a closed message, and U be a finite set of atoms. Then $T \vdash m[U]$ is defined using classical Dolev-Yao inferences where the decode rule is replaced by:*

$$\frac{T \vdash \{m\}_u[U] \quad u \in U}{T \vdash m[U]}$$

The set DYC is trivially extended by adding the atomic constraint $T \Vdash m[U]$ where T and m are potentially not closed but U is still a set of atoms.

The quantification upon the order in which keys are compromised is expressed in the following property.

Proposition 2.10 *Let $T \Vdash m$ be an atomic constraint, then we have that the next predicate is a tautology.*

$$T \Vdash m \Leftrightarrow \bigcup_{\{a_1,\ldots,a_n\} \subseteq K} T \Vdash a_1[] \wedge T \Vdash a_2[a_1] \wedge \ldots \wedge T \Vdash a_n[a_1, \ldots, a_{n-1}] \wedge T \Vdash m[a_1, \ldots, a_n]$$

Where K is the set of all keys in T and m, namely $keys(T) \cup keys(m)$.

Proof: $\Leftarrow$: trivial.
$\Rightarrow$: let σ be a substitution such that $T\sigma \vdash m\sigma$. Then there exists a minimal (for height) proof of $T\sigma \vdash m\sigma$ that only uses keys from K. The last step is to consider the order among keys $\prec$ such that $k_1 \prec k_2$ means that k_1 was deduced before k_2 in our minimal proof. As the proof is minimal, $\prec$ does not have any cycle, hence it is possible to define a total order $<$ among keys compatible with $\prec$. This order gives a_1 (minimal for $<$) to a_n. ■

Thus, using the former property, satisfiability of a constraint in DYC is equivalent to satisfiability of a constraint using only atomic predicates of the form $T \Vdash m[U]$. Such constraints are called quantified constraints. In order to check satisfiability of such constraints, we introduce a rewriting system between constraints.

Definition 2.8 *The rewriting system $\hookrightarrow$ is defined over quantified constraints by:*

$$\begin{aligned} T \Vdash \langle m, n \rangle[U] &\hookrightarrow T \Vdash m[U] \wedge T \Vdash n[U] \\ T \Vdash \{m\}_n[U] &\hookrightarrow T \Vdash m[U] \text{ if } n \in U \end{aligned}$$

Normal forms for the former rewriting system use only atomic constraints like $T \Vdash a[U]$ where a is an atom, $T \Vdash x[U]$ where x is a variable or $T \Vdash \{m\}_k[U]$ where k is an atom that does not appear in U.

Proposition 2.11 *The rewriting system $\hookrightarrow$ over quantified constraints is correct, complete and terminates. So every constraint is equivalent to a constraint in normal form.*

After rewriting the constraint to its normal form, each atomic constraint is decomposed by looking at which atoms, variables and encryptions can be obtained given an environment T and a set of keys U and by looking at which are needed given a message m and a set of keys U.

Definition 2.9 (Split) *The split function is recursively defined over closed messages by the following lines where U is a finite set of atoms.*

$$\begin{aligned} split(a, U) &= \{a\} \\ split(\langle m, n\rangle, U) &= split(m, U) \cup split(n, U) \\ split(\{m\}_u, U) &= split(m, U) \text{ if } u \in U \\ split(\{m\}_v, U) &= \{\{m\}_v\} \text{ if } v \notin U \end{aligned}$$

This function can also be used on sets of messages. Applying split to a set of messages T returns the union of sets obtained by applying split on elements t of T.

$$split(T, U) = \bigcup_{t \in U} split(t, U)$$

Then, the following equivalences are tautologies. And thus, constraints can be transformed to constraints involving $\in$, $\subseteq$ and *split* (as these are constraints, *split* can be applied to message terms).

$$T \Vdash a[U] \Leftrightarrow \Big(a \in S \vee \bigvee_{x \in split(T,U)} a \in split(x, U)\Big)$$

$$T \Vdash \{m\}_n[U] \Leftrightarrow \Big(\{m\}_n \in S \vee \bigvee_{x \in split(T,U)} \{m\}_n \in split(x, U)\Big)$$

$$T \Vdash x[U] \Leftrightarrow \Big(split(x, U) \subseteq S \cup \bigcup_{y \in split(T,U)} split(y, U)\Big)$$

Where S contains all the atoms and messages from $split(T, U)$ (i.e. S is obtained by removing variables from $split(T, U)$). It is now easy to prove a restricted version of the first theorem.

Proposition 2.12 *Let C be a constraint in DYC. If for all atomic constraint $T \Vdash m$ occurring in C, m is a closed message, then satisfiability of C is decidable.*

Proof: Checking satisfiability of constraint C is equivalent to checking satisfiability of a finite number of conjunctions involving atomic constraints of two forms: $a \in split(x, U)$ and $\{m\}_n \in split(x, U)$ where $\{m\}_n$ is a closed message. If there is a non-empty conjunction, it is clearly satisfiable by using for any variable x the pairing of all the atoms and encryptions that occur in the conjunction.

$$x\sigma = \langle a_1, ..., a_\alpha, \{m_1\}_{n_1}, ..., \{m_\beta\}_{n_\beta}\rangle$$

For example if the non-empty conjunction is:

$$a \in split(x, U) \wedge b \in split(x, U) \wedge b \in split(y, U) \wedge \{m\}_n \in split(y, U)$$

This conjunction is satisfied by:

$$\begin{aligned} x\sigma &= \langle a, b, \{m\}_n\rangle \\ y\sigma &= \langle a, b, \{m\}_n\rangle \end{aligned}$$

Hence satisfiability is decidable. ∎

In the rest of this section, we prove theorem 2.1. This proof uses a finite model argument. First, let us introduce the norm which allows us to bound the size of the minimal model. Let us call C the constraint (in normal form) we study and C' be the equivalent constraint using the *split* notation.

Definition 2.10 *Let $\{u\}_v$ be a message, σ be a substitution, then $\sigma[\{u\}_v\backslash u]$ is the substitution defined by:*

$$x(\sigma[\{u\}_v\backslash u]) = (x\sigma)[\{u\}_v\backslash u]$$

I.e. every occurrence of $\{u\}_v$ in $x\sigma$ is replaced by u.

Note that, in general, the following property is false.

$$m(\sigma[\{u\}_v\backslash u]) = (m\sigma)[\{u\}_v\backslash u]$$

This can be noticed by taking $m = \{u\}_v$ and $\sigma = Id$. However, when considering only messages satisfying a specific property, this becomes true. This is expressed in the following two properties.

Proposition 2.13 *Let m, n and $\{u\}_v$ be three messages, U be a finite set of atoms such that $m \in split(n, U)$ and $m \neq \{u\}_v$, then:*

$$m[\{u\}_v\backslash u] \in split(n[\{u\}_v\backslash u], U)$$

Proof: This proof can easily be achieved using an induction on the definition of $split(n, U)$.

- If n is an atom a, then $split(n, U) = \{a\}$, hence $m = a$. Moreover $n[\{u\}_v\backslash u]$ and $m[\{u\}_v\backslash u]$ are both equal to a so we have that:
$$m[\{u\}_v\backslash u] \in split(n[\{u\}_v\backslash u], U)$$
- If n is a pair $\langle n_1, n_2\rangle$, then $split(n, U) = split(n_1, U) \cup split(n_2, U)$. Let us consider that m appears in $split(n_1, U)$ (the other case is symmetrical), then using our induction, we get that:
$$m[\{u\}_v\backslash u] \in split(n_1[\{u\}_v\backslash u], U)$$
As $n[\{u\}_v\backslash u] = \langle n_1[\{u\}_v\backslash u], n_2[\{u\}_v\backslash u]\rangle$ we obtain:
$$m[\{u\}_v\backslash u] \in split(n[\{u\}_v\backslash u], U)$$
- If n is an encryption $\{n_1\}_k$ where k appears in U, $split(n, U) = split(n_1, U)$. We have that:
$$m[\{u\}_v\backslash u] \in split(n_1[\{u\}_v\backslash u], U)$$
If n is equal to $\{u\}_v$, then $n_1[\{u\}_v\backslash u]$ is equal to n_1, hence
$$m[\{u\}_v\backslash u] \in split(n_1, U) = split(n[\{u\}_v\backslash u], U)$$
Else $n[\{u\}_v\backslash u] = \{n_1[\{u\}_v\backslash u]\}_k$ and so,
$$m[\{u\}_v\backslash u] \in split(n[\{u\}_v\backslash u], U)$$
- If n is an encryption $\{n_1\}_k$ where k does not appear in U, $split(n, U) = \{n\}$ so m and n are equal. Hence $m[\{u\}_v\backslash u]$ and $n[\{u\}_v\backslash u]$ are also equal. As m is different from $\{u\}_v$, $n[\{u\}_v\backslash u]$ is equal to $\{n_1[\{u\}_v\backslash u]\}_k$ hence $split(n[\{u\}_v\backslash u], U)$ contains only element $n[\{u\}_v\backslash u]$, so we finally get that
$$m[\{u\}_v\backslash u] \in split(n[\{u\}_v\backslash u], U)$$

■

Proposition 2.14 *Let m, u and u' be three messages, U be a finite set of atoms and σ be a substitution such that there does not exist any sub-term n of m satisfying $n\sigma = u$. Then, the $[\cdot]$ operator is associative, i.e.*

$$\big(m\sigma\big)[u\backslash u'] = m\big(\sigma[u\backslash u']\big)$$

Proof: This can be achieved using an induction on the structure of m. ∎

This allows us to remove the unused encryptions in a model. For this purpose, we introduce a norm over models that is minimal when considering models that have only the necessary encryptions.

Definition 2.11 *The split norm $|\cdot|_s$ is defined over messages and extended to substitutions by the two following equalities for any message m and substitution σ.*

$$|m|_s = \sum_{n \in split(m,\emptyset)} height(n) \qquad |\sigma|_s = \sum_{x \in dom(\sigma)} |x\sigma|_s$$

We can now express our main property. It states that a minimal (for the former norm) solution is composed of sub-terms of the initial constraint. Knowing that, the main theorem is easy to deduce by trying to pair any sub-terms and checking whether the constraint is satisfied or not (this is of course decidable). As we only have conditions upon split (for the "smallest" variable), we only have to try with one copy of each sub-terms and this can be achieved in a finite time.

Proposition 2.15 *Let σ be a model of C' such that $|\sigma|_s$ is minimal. Then for any variable x of C' and for any message $\{u\}_v$ in $split(x\sigma, \emptyset)$, there exists a sub-term b of C' such that $\{u\}_v = b\sigma$.*

Proof: Suppose that the proposition is not verified. Then there exists a variable z from C' such that $\{u\}_v$ appears in $split(z\sigma, \emptyset)$ and for all b sub-message of C', $\{u\}_v \neq b\sigma$. We consider a new substitution σ' defined by $\sigma' = \sigma[\{u\}_v \backslash u]$. To shorten notations, $[\{u\}_v \backslash u]$ will be denoted using [..] in the following.

A first step is to verify that σ' is a model of C'. For this purpose, we consider the three possible cases of atomic constraints.

- $a \in split(x\sigma, U)$: using proposition 2.13, we have $a \in split\big((x\sigma)[..], U\big)$ and by definition, $x\sigma' = (x\sigma)[..]$. Thus $a \in split(x\sigma', U)$.
- $\{m\}_n\sigma \in split(x\sigma, U)$: proposition 2.13 gives $(\{m\}_n\sigma)[..] \in split\big((x\sigma)[..], U\big)$. However, proposition 2.14 can be applied as there are no sub-term of $\{m\}_n$ that is unified with $\{u\}_v$ using σ.
- $split(x\sigma', U) \subseteq S\sigma \cup \bigcup_i split(x_i\sigma, U)$: let m' be a message from $split(x\sigma', U)$. Then if there exists a message m such that $m[..] = m'$ and $m \in split(x\sigma, U)$, there are two cases:
  - If $m \in split(x_i\sigma, U)$, we have $m' \in split(x_i\sigma', U)$.
  - Else, $m = s\sigma$ for some $s \in S$. Moreover, $\{u\}_v \neq s'\sigma$ for any s' sub-message of s and $m' = (s\sigma)[..]$. Hence $m' = c\sigma'$ and so $m' \in S\sigma'$.

  If a such message m does not exist, then u is an encryption of message m' using an arbitrary number of keys from U, u_1 to u_γ. We can deduce that $\{u\}_v \in split(x\sigma, U)$ and so $\{u\}_v \in split(x_i\sigma, U)$ for some i ($\{u\}_v$ cannot be in $S\sigma$). Hence $m' \in split(x_i\sigma', U)$.

The conclusion is that σ' is a model of C' and $|\sigma'|_s < |\sigma|_s$, thus there is a contradiction with the minimality of σ. ∎

2.6.3 NP-completeness

We now discuss the complexity of our approach. First, satisfiability of well-formed constraints is NP-hard (see for example [CKRT03a] or [RT01]). This is the case for well-formed constraints that only involve $\Vdash$. And as all of these constraints are in the set of constraints that we are studying, our satisfiability problem is NP-hard. To show that this satisfiability problem is in NP and thus NP-complete, we rely on the results presented in [RT01]. The authors of this paper proved that given a satisfiable well-formed constraint, there exists a model whose size is polynomial in the size of the constraint. The size used here is the number of different sub-terms. The same result holds for our method. In the former part, we proved that minimal models are made by pairing sub-terms

of the initial constraint. Thus, the number of distinct sub-terms for any $x\sigma$ is clearly bounded. That is why, the DAG size (number of distinct sub-terms) of our model remains polynomial in the size of the initial problem. The last thing to notice is that checking whether a closed constraint is satisfied or not is PTIME. This well-known property comes from the locality theorem stated by McAllester [McA93] and satisfiability of Horn clauses. And so, we can conclude that satisfiability of our constraints is NP-complete. A direct consequence of this result is that secrecy in the symbolic setting is a NP-complete problem when considering a bounded number of sessions.

Chapter 3

Preliminaries for the Computational Model

Contents

In this whole document, η is the security parameter used by the different cryptographic schemes. The higher η is, the more secure the cryptographic primitives are. For example, the length of keys is generally linear in η, thus brute force attacks (by trying any possible value for the key) have a complexity that is exponential in η. Nonces which are implemented by random numbers in the computational world are also assumed to have a size that is linear in η. Hence the complexity of guessing a nonce by trying every possibility is also exponential in η.

3.1 Negligible Functions

A cryptographic scheme is said to be secure if any adversary has a low probability to break it. In this context, low means that the probability has to be negligible in η.

Definition 3.1 (Negligible and p-Negligible Functions) *A function $g : \mathbb{R} \to \mathbb{R}$ is* negligible*, if it is ultimately bounded by x^{-c}, for each positive $c \in \mathbb{N}$, i.e., for all $c \geq 0$ there exists N_c such that $|g(x)| < x^{-c}$, for any $x > N_c$.*

A function $g : \mathbb{R} \to \mathbb{R}$ is p-negligible*, if for all $c > 0$ there exists N_c such that $g(x) < x^{-c}$, for all $x > N_c$.*

As these two notions are equivalent for positive functions, we use indifferently the two terms when considering functions that are always positive, for example probabilities. A typical use of negligible function is to measure the probability that an adversary breaks a cryptographic scheme. More precisely, as an adversary acting randomly may have a chance to break the scheme, we measure the difference between the probability for the adversary to win and the probability for the best random acting adversary to win. This difference is called the *advantage* of the adversary. As it might be possible to design adversaries that always lose, the advantage of such adversaries can be negative and non-negligible. For this reason, we introduce p-negligible functions. Our security requirement is then that the advantage of any adversary has to be p-negligible.

3.1.1 Basic Properties of Negligible Functions

When computing the probability for an adversary to win, negligible functions can be combined in various ways. Thus it is important to verify that such combinations give as results negligible functions. The main results concerning the set of negligible functions are this set is stable under multiplication, exponentiation by a strictly positive constant and linear combination.

Proposition 3.1 *Let f and g be two negligible functions, then*

1. *$f.g$ is negligible.*
2. *For any $k > 0$, f^k is negligible.*
3. *For any λ, μ in $\mathbb{R}$, $\lambda.f + \mu.g$ is negligible.*

The same kind of properties hold for p-negligible functions. The only difference lies in linear combination: as subtraction may produce a negative function, the multiplicative factors have to be positive.

Proposition 3.2 *Let f and g be two p-negligible functions, then*

1. *$f.g$ is p-negligible.*
2. *For any $k > 0$, f^k is p-negligible.*
3. *For any λ, μ in $\mathbb{R}^+$, $\lambda.f + \mu.g$ is p-negligible.*

Moreover, composing a negligible function with a positive polynomial produces a negligible function. The same hold for p-negligible functions.

Proposition 3.3 *Let f be a negligible function, g be a p-negligible function and P be a polynomial that is ultimately positive. Then $f \circ P$ is a negligible function and $g \circ P$ is a p-negligible function.*

3.2 Cryptographic Schemes

Security protocols can use several different encryption schemes. Classical schemes include encryption which is used to ensure secrecy of a message and signature which is used to ensure authenticity of a message.

We briefly introduce these schemes in the introduction. In this section, we formalize the definition of such cryptographic schemes.

An *asymmetric encryption scheme* $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is defined by three algorithms. The key generation algorithm $\mathcal{KG}$ is a randomized function which given a security parameter η outputs a pair of keys (pk, sk), where pk is a public key and sk the associated secret key. The encryption algorithm $\mathcal{E}$ is also a randomized function which given a message and a public key outputs the encryption of the message by the public key. Finally the decryption algorithm $\mathcal{D}$ takes as input a secret key and a cipher-text and outputs the corresponding plain-text, i.e., $\mathcal{D}(\mathcal{E}(m, pk), sk) = m$ if the pair (pk, sk) has been produced by $\mathcal{KG}$. If the message is not a correct encryption or if the keys

do not match, the decryption algorithm outputs $\perp$. The execution time of the three algorithms is assumed polynomially bounded by the security parameter η.

A *symmetric encryption scheme* $\mathcal{SE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is defined by three algorithms. The key generation algorithm $\mathcal{KG}$ is a randomized function which given a security parameter η outputs a key k. The encryption algorithm $\mathcal{E}$ is also a randomized function which given a message and a key outputs the encryption of the message by this key. Finally the decryption algorithm $\mathcal{D}$ takes as input a key and a cipher-text and outputs the corresponding plain-text, i.e., $\mathcal{D}(\mathcal{E}(m,k),k) = m$. If the message is not a correct encryption, the decryption algorithm outputs $\perp$. The execution time of the three algorithms is also assumed polynomially bounded by η.

A *signature scheme* $\mathcal{SS} = (\mathcal{KG}, \mathcal{S}, \mathcal{V})$ is also defined by three algorithms. The key generation algorithm randomly generates pairs of keys (sik, vk), where sik is the signature key and vk is the verification key. The signature algorithm $\mathcal{S}$ randomly produces a signature of a given message by a given signature key. The verification algorithm $\mathcal{V}$ is given a message m, a signature σ and a verification key vk and tests if σ is a signature of m with the signature key corresponding to vk. Hence, $\mathcal{V}(m, \mathcal{S}(m, sik), vk)$ returns true for any message m and any pair of keys (sik, vk) generated by $\mathcal{KG}$. We say that σ is a valid signature under sik if there exists m such that $\mathcal{V}(m, \sigma, vk)$ returns true. We still assume that the algorithms have a polynomial complexity.

Finally, a *cryptographic library* $\mathcal{CL}$ is a finite collection of cryptographic schemes (asymmetric encryption, symmetric encryption and digital signature). Hence security of the cryptographic library is defined as joint security of its components.

Security protocols also use nonces. Nonces are not really defined by a cryptographic scheme. However, the algorithm generating nonces depends on η and has to fulfill a basic no-collision restriction, therefore nonce generation is formulated as a cryptographic scheme that is defined by a single algorithm $\mathcal{G}$. This algorithm takes as argument the security parameter η and returns a bit-string (usually of size η). As the restriction on nonce generation scheme does not use any adversary, it is possible to formulate it now. Let $\mathcal{G}$ be a nonce generation algorithm, this algorithm is said to be resistant against collisions iff the probability p to find a collision is negligible where p is properly defined by:

$$p = Pr[bs_1 := \mathcal{G}(\eta) \ ; \ bs_2 := \mathcal{G}(\eta) \ ; \ bs_1 = bs_2]$$

Various notions of security have been introduced for pseudo-random number generators. However, in the following we consider that nonce generation is perfect: the generation scheme is implemented by an algorithm $\mathcal{G}$ such that $\mathcal{G}(\eta)$ randomly generates a bit-string of length η with uniform probability. This scheme is trivially resistant against collisions.

3.2.1 On Encryption Size

An encryption scheme is designed to preserve secrecy. However for most of them the cipher-text leaks some information on the size of the embedded plain-text. The usual solution consists in padding the message up to a fixed size but this cannot be done for arbitrary message size. As we want to consider potentially unbounded messages, this leak cannot be avoided, therefore it might be possible to deduce the size of a bit-string bs from its encryption. However, we add some restrictions on the cryptographic schemes considered here as we do not want the size of the cipher-text to leak other information on the plain-text than its size. Our hypothesis on encryption schemes is called *fixed encryption size hypothesis*. It can be stated as follows:

Let ℓ be the function on bit-strings such that $\ell(bs)$ is the length (in bits) of bs. Let $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an encryption scheme (either symmetric or asymmetric). Then for any pair of bit-strings bs_0, bs_1 of same length, i.e. $\ell(bs_0) = \ell(bs_1)$ their encryption also have the same length(for a fixed value of η): let k and k' be two keys generated using $\mathcal{KG}$ (either public keys or symmetric keys), then $\ell(\mathcal{E}(bs_0, k)) = \ell(\mathcal{E}(bs_1, k'))$. In this document, we principally consider encryption schemes that are non-deterministic, however we always assume that the function associating to a bit-string the length of its encryption is a deterministic function. Moreover, we ask the length of public keys and the length of private keys to be constant, for a fixed value of η (of course these lengths change whenever η changes).

In the following, we assume that all the encryption schemes verify the fixed encryption size hypothesis.

3.2.2 Cyclic Groups

Cyclic groups are often used by protocols which perform Diffie-Hellman like key exchanges.

Definition 3.2 (Group) *A* group $(G, *)$ *is composed of a set* G *and a binary operator* $*$ *on* G *which satisfy the three following axioms:*

$$\begin{aligned} \forall a, b, c \in G,\ a * (b * c) = (a * b) * c \quad & \textit{Associativity} \\ \exists e \in G,\ \forall a \in G,\ e * a = a * e = a \quad & \textit{Neutral Element} \\ \forall a \in G,\ \exists b \in G,\ a * b = b * a = e \quad & \textit{Inverse Element} \end{aligned}$$

In the last equation, b *is called the inverse of* a *and is denoted by* a^{-1}*.*

In the following, the binary operator $*$ is omitted from the group definition and we use the multiplicative notation to represent this operator.

A group G is cyclic if it can be generated by a single element: there exists an element g of G such that every element of the group is a power of g.

Definition 3.3 (Cyclic Group) *A group* G *is* cyclic *if* G *is finite and there exists an element* g *of* G *such that:*

$$\forall a \in G,\ \exists n \in \mathbb{N},\ a = g^n$$

Element g *is called a* generator *of group* G*.*

Note that we only consider finite cyclic groups. The order of a cyclic group G is its number of elements q. Alternatively, a cyclic group of order q can be defined as a group which is isomorphic to $(\mathbb{Z}_q, +)$.

A family of cyclic groups $\mathbb{G}$ is a function that associates to each security parameter η a cyclic group $\mathbb{G}(\eta)$ of order $q(\eta)$. We often ask $q(\eta)$ to be large (i.e. exponential in η). By abuse of notation, we also use $\mathbb{G}$ to denote $\mathbb{G}(\eta)$. Moreover, we make no difference between the group and its computational implementation.

3.3 Probabilistic Turing Machines

Adversaries are a central notion in the computational model. As we want the adversary to be realistic, it is modeled as a probabilistic Turing machine. Therefore, the adversary is able to perform any computation he wants, for example brute-force attack on an encryption whereas in the Dolev-Yao model, only a few operations were allowed. Probabilistic Turing machines are an extension of classical Turing machines.

3.3.1 Turing Machines

To simplify the notations, Turing machines are never represented formally in this document (i.e. we do not describe tapes or transition systems) but we rather use pseudo-code to describe a Turing machine. Of course, execution of a pseudo-code can be simulated using a Turing machine.

Our pseudo-code syntax is inspired by usual iterative programming languages. Semantics are not given formally here, they are kept intuitive to preserve simplicity. The basic statements are specified thereafter.

1. $x := e$ denotes the assignment of expression e to variable x.
2. **if** e **then** br_1 **else** br_2, if e is evaluated to true, then br_1 is executed, else br_2 is executed.

3. **while** e **do** br **done**, pseudo-code br is executed in loop while e is evaluated to true.
4. br_1 br_2 denotes the sequence of pseudo-code br_1 and pseudo-code br_2. br_1 is executed first then br_2 is executed.
5. **return** e, expression e is evaluated, the result is returned by the Turing machine.

Our Turing Machines are used to manipulate computational substitutions. A *computational substitution* is a mapping that links some names to bit-string values. Typical implementations of such substitutions can be done using associative lists. There are three ways to use such a substitution that corresponds to three new notations in our pseudo-syntax.

1. $\theta := []$, this statement creates a new substitution θ, θ is initialized to the empty substitution.
2. $x\theta := e$, this statement evaluates expression e and x is linked to this value in θ. If x was previously linked in θ, the old value is erased and replaced by the result of the evaluation of e. This statement does not change the value of variables other than x in θ.
3. In an expression, $x\theta$ denotes the value of x in θ. This statement raises an error if x is not defined in θ.

Moreover $sup(\theta)$ denotes the set of variables that are defined in θ.

Finally, we also allow the use of the classical lambda notation to denote functions: an algorithm that takes as input x and returns $f(x)$ can be denoted by $\lambda x.f(x)$.

3.3.2 Probabilistic Turing Machines

In order to enhance the capabilities of adversaries, we want them to be able to generate random numbers. For this purpose, we add a new instruction $x \xleftarrow{R} [1, n]$, this instruction randomly generates a natural number between 1 and n with uniform probability (the probability to obtain any i in $[1, n]$ is $1/n$).

As an adversary can only generate a polynomial number of random integers, probabilities here hold on a finite space. For this reason, it is easy to represent probabilities as an additional argument (or an additional tape) of a Turing machine.

Let us first consider Turing machines such that the only allowed probabilistic instruction is $b \xleftarrow{R} [0, 1]$ (i.e. b is a randomly sampled bit). In this situation, a probabilistic Turing machine is a Turing machine that takes an additional input representing randomness. As we only consider polynomial machines, the length of this last input is bounded. Let p be a probabilistic polynomial Turing machine that has a single input x. Let α be the bound on the execution of p, hence p cannot sample more than α different bits. The deterministic implementation of p is denoted by p_d, it is also denoted by p when no confusion is possible. This implementation takes as argument the same argument x as p (x) and a vector of random bits $\vec{r}$. The length of $\vec{r}$ is α. Then to ensure determinism in p_d, the probabilistic actions $b \xleftarrow{R} [0, 1]$ are interpreted by b takes as value the i^{th} component of $\vec{r}$ and i is incremented where i designates a global counter.

Then the probability that machine p applied to x returns o is given by:

$$Pr[p(x) = o] = \frac{\left|\left\{\vec{r} \in [0,1]^{\alpha} / p_d(x, \vec{r}) = o\right\}\right|}{2^{\alpha}}$$

The distribution of probability of outputs of p is the function that associates to each possible output o the probability $Pr[p(x) = o]$. Two machines p_1 and p_2 have similar behavior if for any argument x, the distribution of outputs of $p_1(x)$ is exactly the same as the distribution of outputs of $p_2(x)$.

Integer α is defined as a bound on the execution of p. It is not important to choose the smallest bound because the probability with two different bounds is the same: let α and α' be two bounds on the execution of p, then

$$\frac{\left|\left\{\vec{r} \in [0,1]^{\alpha} / p_d(x, \vec{r}) = o\right\}\right|}{2^{\alpha}} = \frac{\left|\left\{\vec{r} \in [0,1]^{\alpha'} / p_d(x, \vec{r}) = o\right\}\right|}{2^{\alpha'}}$$

Formalizing Probabilistic Turing Machine In this paragraph, we give a quick glimpse at how probabilistic Turing machines can be described formally. First, as the Turing machine formalism is complicated and too "low-level", we use a Turing powerful language: lambda calculus [Bar84]. A Turing machine corresponds to a lambda term, thus a probabilistic Turing machine also corresponds to a lambda term. The general form of this term is:

$$p = \lambda arg_1 ... \lambda arg_n . \lambda r . t$$

Then for any lambda terms a_1 to a_n (representing parameters) and any lambda term o (representing the result), the probabilistic Turing machine corresponding to p has the following probability to output o when using a_1 to a_n as inputs.

$$Pr[p(a_1, ..., a_n) \rightarrow o] = \frac{\left|\left\{\vec{r} \in [0,1]^\alpha / p\ a_1\ ...\ a_n\ \vec{r} \xrightarrow{\beta^*} o\right\}\right|}{2^\alpha}$$

Where $\vec{r}$ designates a vector of α bits, hence $\vec{r}$ can be represented by a number between 0 and $2^{\alpha-1}$ and can be converted to a lambda term using the classical Church coding.

Authorizing $x \xleftarrow{R} [1,q]$ Statements For this point, the interested reader is referred to section 7.4 of [Sho99]. If q is a power of 2, 2^k then the k bits from x can be obtained by generating random bits. The difficulty occurs when q does not have the form 2^k, simulation of such statement is impossible when using only $x \xleftarrow{R} [0,1]$ statements. However, it is possible to "correctly approximate" the $x \xleftarrow{R} [1,q]$ statement by using only a polynomial number of random coins. A simple idea to achieve this is to consider the integer k such that $2^{k-1} \leq q < 2^k$ then we use the following algorithm:

do
  $x \xleftarrow{R} [1, 2^k]$
while $x > q$

This algorithm only terminates in constant time on average.

3.4 Security of Cryptographic Schemes

Safety for a cryptographic scheme is defined as weak probability for a reasonable time adversary to break this scheme. Brute force attacks like testing any possible key may be possible, that is why we usually do not ask a scheme to be secure for unbounded time adversaries. Weak probability means that the probability of success has to be negligible in the parameter of security η. Reasonable time means that there is a bound on the execution time of the adversary that is polynomial in η. Therefore adversaries are represented by polynomial time random Turing machines (PRTM).

3.4.1 Asymmetric Encryption, IND-CPA

In this section, we recall the classical notion of indistinguishability against chosen plain-text attacks (IND-CPA, which was called polynomial security by Micali and Goldwasser in [GM84]). Let $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme. This encryption scheme is secure for IND-CPA if any adversary (i.e. PRTM) has a negligible probability to win in the following game: first a bit b is chosen randomly; a pair of keys (pk, sk) is generated using $\mathcal{KG}$; the adversary receives pk and has to produce two bit-strings bs_0 and bs_1 (these two bit-strings have to have the same size); bit-string bs_b is encrypted using pk. The resulting bit-string is returned to the adversary which has to deduce what the value of b is. As guessing the value of b at random leads to non-negligible probability of success, the adversary has to have a better probability of success than when answering randomly. An adversary that outputs a random bit has a probability of 1/2 to guess the challenge bit.

Formally, an adversary is constituted of two parts $\mathcal{A}_1$ and $\mathcal{A}_2$ which are two PRTM. A long term store denoted by *mem* is used by the adversary as a shared memory between $\mathcal{A}_1$ and $\mathcal{A}_2$, *mem* is given by $\mathcal{A}_1$ to $\mathcal{A}_2$. Then the game $G^b_{\mathcal{A}_1,\mathcal{A}_2}$ is defined as following:

Game $\mathbf{G}^b_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}(\eta)$
$bs_0, bs_1, mem := \mathcal{A}_1(\eta, pk)$
$bs := \mathcal{E}(bs_b, pk)$
return $\mathcal{A}_2(mem, bs)$

Note that the previous game depends on the challenge bit b, on the security parameter η, on the adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and on the encryption scheme $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$. All these dependencies are not made explicit so that notations are kept as simple as possible. The advantage of an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ measures how much the adversary wins the game compared to how much it loses it. The advantage is given by:

$$\mathbf{Adv}^{IND-CPA}_{(\mathcal{A}_1,\mathcal{A}_2)} = Pr[\mathbf{G}^1_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1] - Pr[\mathbf{G}^0_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1]$$

Definition 3.4 (IND-CPA) *An asymmetric encryption scheme is said to be* secure against IND-CPA *if for any adversary* $(\mathcal{A}_1, \mathcal{A}_2)$, $\mathbf{Adv}^{IND-CPA}_{(\mathcal{A}_1,\mathcal{A}_2)}$ *is a negligible function in* η.

There are example of algorithms that have been proven secure against IND-CPA under the hypothesis that some computational problem is hard to solve. For example, in [FOPS01], algorithm RSA-OAEP is proven secure against IND-CCA if the RSA problem is supposed hard.

Sample Attacks

Let us illustrate how an adversary can try to attack IND-CPA. Suppose that the encryption scheme used here is deterministic. Then consider the adversary such that $\mathcal{A}_1$ returns $0, 1, pk$. $\mathcal{A}_2$ receives as argument his memory pk and the bit-string $\mathcal{E}(b, pk)$. $\mathcal{A}_2$ just has to forge $\mathcal{E}(0, pk)$ and $\mathcal{E}(1, pk)$ and as the encryption scheme is supposed deterministic, he can deduce the value of bit b. Formally, adversary $\mathcal{A}_2$ proceeds as follows:

Adversary $\mathcal{A}_2(mem, bs)$:
$pk := mem$
$bs_0 := \mathcal{E}(0, pk)$
$bs_1 := \mathcal{E}(1, pk)$
if $bs = bs_0$ **then return** 0
else return 1

As $\mathcal{E}(0, pk)$ is different from $\mathcal{E}(1, pk)$, we have $\mathbf{Adv}^{IND-CPA}_{(\mathcal{A}_1,\mathcal{A}_2)} = 1$. Hence an algorithm secure against IND-CPA cannot be deterministic. The number of possible encryptions for a given bit-string and a given public key cannot even be polynomially bounded in η.

Extensions

The notion of IND-CPA as introduced here can be extended in several ways. First, the adversary can be allowed to submit multiple pairs bs_0, bs_1 and be returned the encryption of bs_b. These pairs can even be chosen adaptively, i.e. after receiving the encryption of bs_b, the adversary chooses which pair bs_0, bs_1 to submit next. This can be done by giving to the adversary access to an oracle that takes as argument a pair bs_0, bs_1 and returns $\mathcal{E}(bs_b, pk)$. This oracle is called the *left-right encryption oracle* and is commonly used in indistinguishability criteria.

Another extension is the *multi-user setting*. In this situation, N pairs of keys are generated, the adversary has access to all the public keys and to N oracles $\lambda(bs_0, bs_1).\mathcal{E}(bs_b, pk_i)$. This security criterion is called N-IND-CPA. A classical result in provable cryptography is the equivalence

between IND-CPA (or 1-IND-CPA where the "left-right oracle" is called only once) and N-IND-CPA. This result appears in [BBM00] and the proof given in this paper uses the famous hybrid argument.

Proposition 3.4 *Let N be an integer. An asymmetric encryption scheme is secure for N-IND-CPA iff it is secure for IND-CPA.*

Proof: See [BBM00]. ■ All these notions of IND-CPA security are equivalent hence in the following IND-CPA is used as a shorthand for 1-IND-CPA, that is in IND-CPA the adversary is allowed to perform multiple calls to the left-right encryption oracle.

Finally a last extension is indistinguishability against chosen cipher-text attacks (IND-CCA). In this case, a single key pair is generated, the adversary has access to the public key and to the left-right encryption oracle but he also has access to a decryption oracle $\lambda bs.\mathcal{D}(bs, sk)$. However, as we do not want the game to be too easy, the decryption oracle cannot be called on any bit-string that has been output by the left-right oracle. This game is easier to win than the classical IND-CPA game.

Proposition 3.5 *If an asymmetric encryption scheme is secure against IND-CCA, then it is also secure against IND-CPA.*

However, the converse of the previous proposition is false if we suppose the existence of an algorithm that is secure against IND-CPA.

Proposition 3.6 *If there exists an asymmetric encryption scheme secure against IND-CPA, then there exists an asymmetric encryption scheme that is secure against IND-CPA but not secure against IND-CCA.*

Proofs for the previous propositions cannot be achieved here as adversaries using oracles have not been defined properly. Thus these proofs are delayed to section 5.4.1.

3.4.2 Digital Signature, UNF

A digital signature scheme has to guarantee an unforgeability property. However there are several variants of this property, some of them are presented in [GMR88]. The main idea is that it should be hard for any adversary to produce a valid signature using in some cases only the verification key and in the other cases other signatures. Therefore, there are two kinds of attacks: *key only attacks* where the adversary only has access to the verification key and *message attacks* where the adversary has access to the verification key and to signatures of some messages.

Of course, there are several different flavors of message attacks. In *known message attacks* the adversary knows the signature of some bit-strings $bs_1, \cdots, bs_n$ as well as the bit-strings themselves, in *directed chosen message attacks*, the adversary has access to the verification key, he chooses n bit-strings and receives their signatures. Finally in *adaptive chosen message attacks*, the adversary can choose each bit-string after seeing the signature of the previous bit-string he has chosen.

In order to win his game, the adversary has to satisfy *existential forgery*, he has to be able to produce a fresh signature even if he does not know which bit-string is signed.

Here, we are interested in adaptive chosen message attacks, this is denoted by UNF in the following. Let $(\mathcal{KG}, \mathcal{S}, \mathcal{V})$ be a digital signature scheme. Let $\mathcal{A}$ be an adversary against UNF, then the game involving $\mathcal{A}$ is defined by:

Game $\mathbf{G}^b_{\mathcal{A}}(\eta)$:
$(sik, vk) := \mathcal{KG}(\eta)$
$s := 0$
$mem := vk, \eta$
while $s = 0$ **do**
$s, bs_{sig}, mem := \mathcal{A}(mem)$
$mem := \mathcal{S}(bs_{sig}, sik), mem$

return s

The game proceeds as follows: first a signature key pair (sik, vk) is randomly generated using the key generation algorithm $\mathcal{KG}$. Then the initial memory of the adversary mem is created. It contains the security parameter η and the verification key vk. The adversary is executed and can ask for signature of a bit-string bs_{sig}. The resulting signature is appended to his memory before the next call of the adversary. Finally, the adversary outputs a bit-string s different from 0 which should be a valid signature for verification key vk. The condition that $\mathcal{A}$'s execution time has to be polynomially bounded is not sufficient to ensure termination of this experiment. Thus, the only adversaries considered here are those for which the execution of the whole game is polynomially bounded in η.

Then the advantage of $\mathcal{A}$ is defined by the probability for $\mathcal{A}$ to produce a valid signature at the end of game $\mathbf{G}$ such that this signature has not been produced by the $\mathcal{S}$ algorithm used in game $\mathbf{G}$:

$$\mathbf{Adv}_{\mathcal{A}}^{UNF} = Pr[bs := \mathbf{G}_{\mathcal{A}}^{b}(\eta) \text{ where } bs \text{ is a valid signature for } vk \text{ not produced in } \mathbf{G}]$$

Definition 3.5 (UNF) *A digital signature scheme is said to be* secure against UNF *if for any adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{UNF}$ is a negligible function in η.*

There exist some signature schemes that are strongly believed to be secure against UNF, see [GMR88] for example.

3.4.3 Symmetric Encryption

In order to be secure, a symmetric encryption scheme has to preserve secrecy of the encrypted messages in a similar way as IND-CPA but it also has to prevent production of new encryptions as UNF. Therefore, security of a symmetric encryption scheme is a mix between IND-CPA and UNF. In this case, the term authenticated encryption can be used instead of symmetric encryption. Such a mix of unforgeability and indistinguishability has been proposed under the name IND-CPA∧INT-CTXT in [BN00].

Criterion IND-CPA has to be modified as there is only one key in symmetric encryption and this key has to remain secret. The adversary only has access to the left-right encryption oracle and has to guess the value of the challenge bit b. Criterion UNF also has to be modified, the adversary only has access to a signature oracle, this oracle is implemented using the encryption algorithm.

In the following, we are interested in proving the security of a whole cryptographic library that contains at least an asymmetric encryption scheme, a symmetric encryption scheme and a digital signature scheme. Security of symmetric encryption has introduced the need for mixing security criteria. This is generalized in the following chapter but in order to do this properly, we first introduce a formal definition for security criterion.

3.5 Relating Security of Different Criteria to IND-CPA

In this section, we recall classical results comparing different criteria for asymmetric encryption. The objective is to show that these proofs are often complicated, hard to understand and difficult to check. This has also been underlined by Victor Shoup in the introduction of [Sho04]: security proofs in the computational world often "become so messy, complicated, and subtle as to be nearly impossible to understand". Since the main difficulty lies in the design of new adversaries, we introduce in chapter 5 a theorem allowing one to compare different criteria without having to describe such new adversaries.

Criteria introduced in this section are commonly used in provable cryptography. The first criterion is real-or-random security [BDJR97]: is it impossible for an intruder to distinguish the

encryption of a text he has chosen from the encryption of a random text. The second criterion, more complex, is called non-malleability [BDPR98].

3.5.1 Real-or-Random Security

Let $\mathcal{AE}$ be an asymmetric encryption scheme $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$. Safety of this encryption scheme can be characterized by the IND-CPA criterion. Another possible way of describing safety of asymmetric encryption schemes is to consider Real-or-Random Attacks (RRA). An encryption scheme is secure for RRA if any adversary has a negligible probability to win in the following game: first a key pair (pk, sk) is generated, then a first part of the adversary, $\mathcal{A}_1$, has to compute a bit-string bs, $\mathcal{A}_1$ has access to the public key pk and the security parameter η. At this point, there are two possible cases, either bs is encoded using pk, or a random bit-string (but whose length is the same as the length of bs) is encoded using pk. The resulting cipher-text is given to the second part of the adversary, $\mathcal{A}_2$. Finally, $\mathcal{A}_2$ has to return 1 if he thinks bs was encoded or 0 if he thinks a random bit-string was encoded in order to win the challenge.

Formally, an adversary is composed of a first part $\mathcal{A}_1$ and a second part $\mathcal{A}_2$. The adversary can be faced to two different games. In the first one, the bit-string output by $\mathcal{A}_1$ is really used in the encryption whereas a random bit-string is used in the second case. The length of bit-string bs is still denoted by $\ell(bs)$.

Game $\mathbf{G}^{Real}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}$
$bs, mem := \mathcal{A}_1(\eta, pk)$
$bs' := \mathcal{E}(bs, pk)$
return $\mathcal{A}_2(mem, bs')$

Game $\mathbf{G}^{Rand}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}$
$bs, mem := \mathcal{A}_1(\eta, pk)$
$r \xleftarrow{R} [0,1]^{\ell(bs)}$
$bs' := \mathcal{E}(r, pk)$
return $\mathcal{A}_2(mem, bs')$

Then the advantage of an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ is defined by:

$$\mathbf{Adv}^{RR}_{(\mathcal{A}_1,\mathcal{A}_2)}(\eta) = Pr[\mathbf{G}^{Real}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1] - Pr[\mathbf{G}^{Rand}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1]$$

Definition 3.6 (Real-or-Random Security) *An encryption scheme $\mathcal{AE}$ is said secure against Real-or-Random Attacks (RRA) if the advantage of any pair of adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ is p-negligible in η.*

The main result concerning real-or-random security is that it is equivalent to IND-CPA security. This result and its proof appear in [BDJR97].

Proposition 3.7 *An asymmetric encryption scheme $\mathcal{AE}$ is secure against IND-CPA if and only if it is secure against RRA.*

Proof:

IND-CPA $\Rightarrow$ RRA We start by proving that an encryption scheme secure against IND-CPA is secure against RRA. Let $\mathcal{AE}$ be an encryption scheme secure against IND-CPA and $(\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against RRA. Then we build an adversary $(\mathcal{A}'_1, \mathcal{A}_2)$ against IND-CPA. The second component of this adversary is unchanged whereas the first one is modified in order to generate a random bit-string:

Adversary $\mathcal{A}'_1(\eta, pk)$:
$bs, mem := \mathcal{A}_1(\eta, pk)$
$bs' \xleftarrow{R} [0,1]^{\ell(bs)}$
return $(bs', bs), mem$

Then after unfolding the definition of $\mathcal{A}'_1$, it is easy to note that games $\mathbf{G}^{Real}_{\mathcal{A}_1,\mathcal{A}_2}$ and $\mathbf{G}^{1}_{\mathcal{A}'_1,\mathcal{A}_2}$ are semantically the same:

Game $\mathbf{G}^{Real}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}$
$bs, mem := \mathcal{A}_1(\eta, pk)$
$bs' := \mathcal{E}(bs, pk)$
return $\mathcal{A}_2(mem, bs')$

Game $\mathbf{G}^{1}_{\mathcal{A}'_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}$
$bs, mem := \mathcal{A}_1(\eta, pk)$
$bs' \xleftarrow{R} [0,1]^{\ell(bs)}$
$bs'' := \mathcal{E}(bs, pk)$
return $\mathcal{A}_2(mem, bs'')$

The only difference is that a random bit-strin bs' is generated in game $\mathbf{G}^1$ but is not used. This game uses more random coins but we get that the probability of success is the same, i.e.:

$$Pr[\mathbf{G}^{Real}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1] = Pr[\mathbf{G}^{1}_{\mathcal{A}'_1,\mathcal{A}_2}(\eta) = 1]$$

The same thing can be done to relate games $\mathbf{G}^{Rand}_{\mathcal{A}'_1,\mathcal{A}_2}$ and $\mathbf{G}^{0}_{\mathcal{A}'_1,\mathcal{A}_2}$ and this leads to:

$$Pr[\mathbf{G}^{Rand}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1] = Pr[\mathbf{G}^{0}_{\mathcal{A}'_1,\mathcal{A}_2}(\eta) = 1]$$

Therefore the advantage of $(\mathcal{A}_1, \mathcal{A}_2)$ against RRA is equal to the advantage of $(\mathcal{A}'_1, \mathcal{A}_2)$ against IND-CPA. As the encryption scheme is secure against IND-CPA, the advantage of $(\mathcal{A}'_1, \mathcal{A}_2)$ is p-negligible and so the advantage of $(\mathcal{A}_1, \mathcal{A}_2)$ is also p-negligible. The encryption scheme is also secure against RRA.

RRA $\Rightarrow$ IND-CPA Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against IND-CPA. Then the adversary $(\mathcal{A}'_1, \mathcal{A}'_2)$ against RRA is built from this adversary by randomly selecting one of the two bit-strings output by $\mathcal{A}_1$:

Adversary $\mathcal{A}'_1(\eta, pk)$:
$b \xleftarrow{R} [0,1]$
$bs_0, bs_1, mem := \mathcal{A}_1(\eta, pk)$
return $(bs_b, b.mem)$

The second component $\mathcal{A}'_2$ verifies that $\mathcal{A}_2$ correctly deduced the value of b. If this is the case, it outputs 1 (as the adversary assumes that this correct guess is linked to the Real oracle), otherwise, it outputs 0.

Adversary $\mathcal{A}'_2(b.mem, bs)$:
$b' := \mathcal{A}_2(mem, bs)$
return $b' = b$

Once more, the result is obtained by comparing the two games.

Game $\mathbf{G}^{Real}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta)$:
$(pk, sk) := \mathcal{KG}$
$b \xleftarrow{R} [0,1]$
$bs_0, bs_1, mem := \mathcal{A}_1(\eta, pk)$
$bs' := \mathcal{E}(bs_b, pk)$
$b' := \mathcal{A}_2(mem, bs)$
return $b' = b$

Game $\mathbf{G}^{b}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}$
$bs_0, bs_1, mem := \mathcal{A}_1(\eta, pk)$
$bs := \mathcal{E}(bs_b, pk)$
return $\mathcal{A}_2(mem, bs)$

Thus, game $\mathbf{G}^{1}_{\mathcal{A}_1,\mathcal{A}_2}$ is equivalent to game $\mathbf{G}^{Real}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta)$ when the bit b (generated by $\mathcal{A}'_1$) equals 1. Game $\mathbf{G}^{0}_{\mathcal{A}_1,\mathcal{A}_2}$ is equivalent to game $\mathbf{G}^{Real}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta)$ when the bit b (generated by $\mathcal{A}'_1$) equals 0 except that the output are opposite (the output of $\mathcal{A}_2$ in the first game and its negation in the second one). Hence we get the following equalities on probabilities::

$$Pr[\mathbf{G}^{Real}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) = 1 \mid b = 1] = Pr[\mathbf{G}^{1}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1]$$

$$Pr[\mathbf{G}^{Real}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) = 1 \mid b = 0] = 1 - Pr[\mathbf{G}^{0}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1]$$

By adding these two lines, we have that:

$$2Pr[\mathbf{G}^{Real}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) = 1] - 1 = \mathbf{Adv}^{IND-CPA}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$$

Let us now compare the game Rand when the randomly generated bit b from $\mathbf{G}^{Rand}$ is equal to 1 with the same game when this bit is equal to 0.

Game $\mathbf{G}^{Rand}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) \mid b = 1$:
$(pk, sk) := \mathcal{KG}$
$bs_0, bs_1, mem := \mathcal{A}_1(\eta, pk)$
$r \xleftarrow{R} [0,1]^{\ell(bs_1)}$
$bs' := \mathcal{E}(r, pk)$
$b' := \mathcal{A}_2(mem, bs)$
return $b' = 1$

Game $\mathbf{G}^{Rand}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) \mid b = 0$:
$(pk, sk) := \mathcal{KG}$
$bs_0, bs_1, mem := \mathcal{A}_1(\eta, pk)$
$r \xleftarrow{R} [0,1]^{\ell(bs_0)}$
$bs' := \mathcal{E}(r, pk)$
$b' := \mathcal{A}_2(mem, bs)$
return $b' = 0$

The two games are equivalent except that, once more their output are opposite. Thus we have that:

$$Pr[\mathbf{G}^{Rand}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) = 1 \mid b = 1] + Pr[\mathbf{G}^{Rand}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) = 1 \mid b = 0] = 1$$

$$2Pr[\mathbf{G}^{Rand}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) = 1] = 1$$

By combining this with the previous equality, we get the following relation on advantages:

$$2\mathbf{Adv}^{RR}_{\mathcal{A}'_1,\mathcal{A}'_2}(\eta) = \mathbf{Adv}^{IND-CPA}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$$

The encryption scheme $\mathcal{AE}$ is assumed to be secure against RRA, so the advantage of $(\mathcal{A}'_1, \mathcal{A}'_2)$ is negligible. Hence the advantage of any adversary $(\mathcal{A}_1, \mathcal{A}_2)$ against IND-CPA is also negligible and the encryption scheme is secure against IND-CPA. ■

There exist several other variants of real-or-random security. For example, in the case of real-or-zero security, there are two games: the "real" game is implemented as in the case of RRA, the "rand" game is replaced by a zero game where instead of encrypting a random bit-string, a sequence of 0 (which has the right size $\ell(bs)$) is encrypted using pk. Real-or-zero security is equivalent to security for RRA and to security for IND-CPA.

3.5.2 Non-malleability

Non-malleability is a security notion for encryption schemes that was introduced in [DDN91]. The definition used here corresponds to the NM-CPA variant as introduced by Bellare et al. in [BDPR98]. The idea beyond non-malleability is that it is impossible to transform a cipher-text into another different cipher-text such that the corresponding plain-texts are meaningfully related. More precisely, starting from a cipher-text bs' which is the encryption of bs, it is not possible to output a cipher-text bs'_1 different from bs' but whose included plain-text bs_1 can be related to bs. An adversary is composed of two stages $\mathcal{A}_1$ and $\mathcal{A}_2$. The first stage is given a public key pk and outputs a PRTM M (which describes a bit-string space). All the possible outputs of M must have the same length. Then a random bit-string bs is drawn from M and its encryption bs' using pk is given to $\mathcal{A}_2$. $\mathcal{A}_2$ has to produce a polynomial Turing machine R and a vector of bit-strings $\vec{bs'}$ such that: bs' does not occur in $\vec{bs'}$ and if $\vec{bs}$ is obtained by decrypting bit-strings from $\vec{bs'}$ (using sk) then $R(bs, \vec{bs})$ has to output 1 more frequently than $R(bs_f, \vec{bs})$ where bs_f is a fresh bit-string generated using M.

Let $\mathcal{AE}$ be an asymmetric encryption scheme $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$. The adversary can be faced to two different games, one where R is evaluated on bs and one where R is evaluated on bs_f.

Game $\mathbf{G}^{NM}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}(\eta)$
$M, mem := \mathcal{A}_1(\eta, pk)$
$bs := M(\eta)$
$bs' := \mathcal{E}(bs, pk)$
$R, \vec{bs'} := \mathcal{A}_2(mem, bs')$
$\vec{bs} := \mathcal{D}(\vec{bs'}, sk)$
return $bs \notin \vec{bs} \wedge \perp \notin \vec{bs} \wedge R(bs, \vec{bs})$

Game $\mathbf{G}^{NMf}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
$(pk, sk) := \mathcal{KG}(\eta)$
$M, mem := \mathcal{A}_1(\eta, pk)$
$bs := M(\eta)$
$bs' := \mathcal{E}(bs, pk)$
$R, \vec{bs'} := \mathcal{A}_2(mem, bs')$
$\vec{bs} := \mathcal{D}(\vec{bs'}, sk)$
$bs_f := M(\eta)$
return $bs \notin \vec{bs} \wedge \perp \notin \vec{bs} \wedge R(bs_f, \vec{bs})$

Then the advantage of an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ is defined by:

$$\mathbf{Adv}^{NM-CPA}_{(\mathcal{A}_1,\mathcal{A}_2)}(\eta) = Pr[\mathbf{G}^{NM}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1] - Pr[\mathbf{G}^{NMf}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1]$$

Definition 3.7 (NM-CPA) *An encryption scheme $\mathcal{AE}$ is said secure against Non Malleable Chosen Plain-text Attacks (NM-CPA) if the advantage of any pair of adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ is p-negligible in η.*

The main result concerning non-malleability is that security against NM-CPA implies security against IND-CPA. A generalized version of this result is proven in [BDPR98]. Therefore, the proof of the following proposition is not detailed here.

Proposition 3.8 *If an asymmetric encryption scheme $\mathcal{AE}$ is secure against NM-CPA then it is secure against IND-CPA.*

Part II

Relating the Computational Model and the Symbolic Model

Chapter 4

Characterizing Computational Safety

Contents

The usual definition for safety is that it should be impossible for any adversary to break some property. However, in the computational world, brute force attacks may lead to success (but with high time complexity). Thus safety of a cryptographic library is defined as the low probability for any (reasonable time) adversary to win some security game. Reasonable time means polynomial time in the security parameter η which characterizes the strength of the cryptographic library. The intuition of security games is simple: some secret challenges are generated, then the adversary tries to guess something on the challenges. For this purpose, the adversary (which is represented by a PRTM) has access to some oracles. The adversary can submit queries to the oracles and receives their outputs. We are interested in quantifying the probability that an adversary can gain some information on his challenges (and thus win the game) by querying the oracles.

We first have to introduce adversaries that can access oracles, this is done in section 4.1. Then section 4.2 gives some formal definitions for security games. These definitions are exemplified through section 4.3. Section 4.4 gives some simple properties for security criteria. Finally, section 4.5 applies the criterion formalism on extensions of classical security requirements.

4.1 Adversaries

Adversaries are polynomial-time random Turing machines (PRTM) with oracles. Oracles are also PRTM. In order to make precise the oracles an adversary can query, oracle names are given in a similar way as arguments are given to a procedure. Hence in our pseudo-syntax, the definition of an adversary $\mathcal{A}$ starts with a line of the form:

$$\textbf{Adversary}\mathcal{A}/\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_n$$

Here, the adversary has access to n oracles whose names are $\mathcal{O}_1$ to $\mathcal{O}_n$. Let us give a simple example of this:

Adversary $\mathcal{A}/\mathcal{O}_1, \mathcal{O}_2$:
  $s := \mathcal{O}_1(0)$
  $t := \mathcal{O}_2(s)$
  return t

This adversary $\mathcal{A}$ has access to two oracles. Queries can be made to the oracles using names $\mathcal{O}_1$ and $\mathcal{O}_2$. In this situation, when executing this adversary $\mathcal{A}$, we use the notation $\mathcal{A}/F_1, F_2$ where F_1 and F_2 are two PRTM to denote that oracle names $\mathcal{O}_1$ and $\mathcal{O}_2$ are respectively implemented with procedures F_1 and F_2.

We use the standard λ-notation to concisely describe PRTM obtained from others by fixing some arguments. For instance, let G be a PRTM that has two inputs. Then, we write $\lambda s.G(s, \theta)$ to describe the machine that is obtained from G by fixing the second argument to the value θ. Thus, $\mathcal{A}/\lambda s.G(s, \theta)$ denotes the machine $\mathcal{A}$ that may query an oracle obtained from G by instantiating its second argument by θ. The argument θ of G is defined in the context of $\mathcal{A}$ and may not be known by $\mathcal{A}$. So typically, $\mathcal{A}$ may be trying to compute some information on θ through successive queries to the oracle.

Moreover, adversaries are often used as sub-routines in other adversaries. Consider the following description of a randomized algorithm with oracles.

Adversary $\mathcal{A}'/\mathcal{O}_1$:
  θ_2:=...
  s:=$\mathcal{A}/\mathcal{O}_1$,
    $\lambda s.F_2(s, \theta_2)$

Here adversary $\mathcal{A}'$ uses $\mathcal{A}$ as a sub-routine. Adversary $\mathcal{A}'$ may query oracle $\mathcal{O}_1$. On its turn $\mathcal{A}$ may query the same oracle $\mathcal{O}_1$ and additionally the oracle $\lambda s.F_2(s, \theta_2)$. The latter is obtained from F_2 by fixing the second argument to θ_2 which is generated by $\mathcal{A}'$.

An important remark is that Turing machines that can access oracles can be simulated using standard Turing machines. Let us suppose, without loss of generality, that $\mathcal{A}$ is a Turing machine that has access to a single oracle $\mathcal{O}$ (there is no loss of generality as multiple oracles $\mathcal{O}_1, \cdots, \mathcal{O}_n$ can be grouped in a single oracle $\lambda(i, bs).\mathcal{O}_i(bs)$). Then, instead of writing $\mathcal{A}/F$ to denote the execution of adversary $\mathcal{A}$ where the oracle is implemented by F, it is possible to replace calls to the oracle by return code. This is done in the following code:

$mem := 0$
$end := 0$
$answer := 0$
while $!end$ **do**
  $result, end, mem := \mathcal{A}(mem, answer)$
  if $!end$ **then** $answer := F(result)$
done
return $result$

Of course, adversary $\mathcal{A}$ has to be slightly modified because of his arguments and values he has to return. This translation is called a *trampoline* in compilation. The main advantage of the oracle model is that its syntax is easier to use.

4.2 Security Criteria

Section 3.4 has introduced different security criteria. Each of these criteria corresponds to a game that an adversary tries to win. These games can be separated in three steps:

1. In the first step, some challenges are generated by a random algorithm. For example, in the case of IND-CPA a pair of keys (pk, sk) and a random bit b are randomly sampled.

2. In the second step, the adversary is executed and can access some oracles. Oracles can use the different challenge generated during the first step. In IND-CPA, there is one oracle for the public key and one for left-right encryption.

3. In the third step, the output of the adversary is analyzed to check if he wins his game. The challenges generated in step one are also helpful here to verify the correctness of the output. In IND-CPA, the adversary has to output the value of bit b.

As this three-steps scheme seems general, we introduce a formal notion of security criterion based on it. The objective is to allow one to formally describe security criterion without having to entirely describe the experiment performed on the adversary. This notion of criterion was first introduced in [JLM05a] but in a more complex and less general way.

4.2.1 Definition of Criteria

A security criterion is defined as a game involving an adversary (represented by a PRTM). The game proceeds as follows. First some parameters θ are generated randomly using a PRTM Θ. The adversary is executed and can query an oracle F which depends on θ. At the end, the adversary has to answer a string of bits whose correctness is checked by an algorithm V which also uses θ (e.g. θ includes a bit b and the adversary has to output the value of b). Thus, a criterion is given by a triple consisting of three randomized algorithms:

- Θ is a PRTM that randomly generates some challenge θ which is represented by a substitution from challenge names to bit-strings.

- F is a PRTM that takes as arguments a string of bits s and a challenge θ and outputs a new string of bits. F represents the oracles that an adversary can call to solve his challenge.

- V is a PRTM that takes as arguments a string of bits s and a challenge θ and outputs either true or false. It represents the verification made on the result computed by the adversary. The answer true (resp. false) means that the adversary solved (resp. did not solve) the challenge.

As a quick example consider an asymmetric encryption scheme and consider the IND-CCA security notion. Then, Θ generates a challenge bit b and a pair of keys (pk, sk); F represents the public-key, the left-right oracle, and the decryption oracle; and V checks whether the returned bit equals b.

Note that Θ can generate several challenges and F can represent several oracles. Technically, the different Θ_i are called one after another to create a substitution θ (which is the union of the basic substitutions θ_i created by each Θ_i). Oracle F can be used to access oracles F_1 through F_m by adding a parameter i which denotes the index of the oracle that the adversary tries to query. The code of F is given by $\lambda(i, bs).F_i(bs)$. Thus, it is possible to define criteria with multiple Θ and F. Such criteria are denoted by $(\Theta_1, \ldots, \Theta_n; F_1, \ldots F_m; V)$. In this case, F is called a *meta-oracle*. The extensions to multiple challenge generators and to multiple oracles are straightforward. However the case of multiple verifiers is more complex and is only detailed further in this section.

4.2.2 Experiments and Advantages

Let γ be a criterion (Θ, F, V). The advantage of an adversary $\mathcal{A}$ against γ measures the information that $\mathcal{A}$ gains by using oracle F. The execution of $\mathcal{A}$ with oracle F is performed through a game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$. This game is a Turing machine defined by:

Game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$:
θ:=$\Theta(\eta)$
d:=$\mathcal{A}(\eta)/\lambda s.F(s,\theta)$
return $V(d,\theta)$

During the game, a challenge θ is randomly generated using Θ, then $\mathcal{A}$ is executed and can query oracle F. At the end of its execution, $\mathcal{A}$ outputs a bit-string d which is tested using verifier V. Adversary $\mathcal{A}$ wins the game if verifier V returns *true*, thus if the game itself returns *true*. Otherwise, $\mathcal{A}$ loses the game.

In order to define the advantage of $\mathcal{A}$, we first have to introduce the best probability an adversary can get to win the game without being able to use F. Oracle F is replaced by a new oracle ϵ which does not use its inputs and always outputs the empty bit-string. Let γ' be the criterion $(\Theta; \epsilon; V)$ then $PrRand^{\gamma}(\eta)$ is defined by:

$$PrRand^{\gamma}(\eta) = \max_{\mathcal{A}} \big(Pr[\mathbf{G}^{\gamma'}_{\mathcal{A}}(\eta) = true]\big)$$

where $\mathcal{A}$ ranges over any possible PRTM.

The advantage of adversary $\mathcal{A}$ against γ is defined by:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 2\big(Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - PrRand^{\gamma}(\eta)\big)$$

Intuitively, the advantage of $\mathcal{A}$ is the probability that it wins minus the probability that an adversary playing at random (i.e. using an uninformative oracle) wins. The factor 2 is here in order for the advantage to be the same as the classical indistinguishability advantage as proved in proposition 4.2.

In some cases, $PrRand$ is easy to compute. For example if the adversary has to guess the value of a randomly chosen element, the probability is given by the following proposition.

Proposition 4.1 *Let $\gamma = (\Theta, F, V)$ be a criterion where Θ generates a random element x from a finite set E (with uniform probability) and the verifier V returns true if its argument s is equal to x then*

$$PrRand^{\gamma} = \frac{1}{|E|}$$

Proof: Let $\gamma = (\Theta, \epsilon, V)$ be a criterion such that Θ generates a random element x from $[1, n]$ with uniform probability $(1/n)$, and V tests that the adversary guessed x. Let $\mathcal{A}$ be an adversary. As $\mathcal{A}$ does not have access to any oracle, he can be represented by his probability p_i to answer i.

$$\begin{aligned} Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] &= \frac{1}{n}\sum_{i=1}^{n} Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true | x = i] \\ &= \frac{1}{n}\sum_{i=1}^{n} p_i \\ &= \frac{1}{n} \end{aligned}$$

Therefore the probability for any adversary to win is $1/n$ and so $PrRand^{\gamma} = 1/n$. ∎

Using this proposition, we relate our new definition of advantage with the classical definition of advantage for indistinguishability. Let us consider a criterion where a random bit b is generated and the adversary tries to deduce the value of b. For this classical definition, the advantage is the probability that the adversary outputs 1 when the value of b was really 1 minus the probability that the adversary outputs 1 when the value of b was 0.

Proposition 4.2 *Let $\gamma = (\Theta, F, V)$ be a criterion where Θ generates a random bit b (with uniform probability) and V returns true if its argument s is equal to b. Let $\mathcal{A}$ be an adversary whose output can only be 0 or 1. Then*

$$\begin{aligned}\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) \quad = \quad & Pr[\theta := \Theta \ ; \ \mathcal{A}(\eta)/\lambda s.F(s,\theta) = 1 \mid b\theta = 1] \\ & -Pr[\theta := \Theta \ ; \ \mathcal{A}(\eta)/\lambda s.F(s,\theta) = 1 \mid b\theta = 0]\end{aligned}$$

Proof: By applying proposition 4.1, it is easy to get that $PrRand^{\gamma} = 1/2$. Using that, let us compute the advantage of $\mathcal{A}$:

$$\begin{aligned}\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) &= 2\big(Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - 1/2\big) \\ &= 2\big(Pr[\theta := \Theta \ ; \ \mathcal{A}(\eta)/\lambda s.F(s,\theta) = b\theta]\big) - 1 \\ &= Pr[\theta := \Theta \ ; \ \mathcal{A}(\eta)/\lambda s.F(s,\theta) = 1 \mid b\theta = 1] \\ &\quad + Pr[\theta := \Theta \ ; \ \mathcal{A}(\eta)/\lambda s.F(s,\theta) = 0 \mid b\theta = 0] - 1 \\ &= Pr[\theta := \Theta \ ; \ \mathcal{A}(\eta)/\lambda s.F(s,\theta) = 1 \mid b\theta = 1] \\ &\quad - Pr[\theta := \Theta \ ; \ \mathcal{A}(\eta)/\lambda s.F(s,\theta) = 1 \mid b\theta = 0]\end{aligned}$$

The last equality is true only for adversaries that must output either 0 or 1 as the opposite of event "$\mathcal{A}$ outputs 0" is event "$\mathcal{A}$ outputs 1". This restriction on the outputs of $\mathcal{A}$ is not really strong as one can interpret outputs different from 0 and 1 to be 0. ■

Criteria with Multiple Verifiers Let $\gamma_i = (\Theta; F; V_i)$ be n criteria (i ranges between 1 and n). The challenge generators Θ and the oracles F provided by these criteria are the same. Hence it is natural to introduce a criterion that combines all these basic criteria. This criterion γ is denoted by $(\Theta; F; V_1, ..., V_n)$ and is said to be a *multiple verifiers* criterion. In this case, the advantage of an adversary $\mathcal{A}$ is defined by:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \max_{1 \le i \le n} \big(\mathbf{Adv}^{(\Theta;F;V_i)}_{\mathcal{A}}(\eta)\big)$$

A criterion γ is said *safe* iff the advantage of any adversary against γ is p-negligible. Then an immediate consequence of the definition is stated in the following proposition.

Proposition 4.3 *A criterion $(\Theta; F; V_1, ..., V_n)$ is safe if and only if all the sub-criteria $(\Theta; F; V_i)$ are safe (for i from 1 to n).*

Note that this property is also true when the different V_i represent multiple verifiers instead of just single verifiers.

Oracle Memory An important point about criteria is that frequently an oracle has to store some information that will be useful for further use of this oracle, other oracles or even the verifier. For example in the case of IND-CCA, the adversary cannot query the decryption oracle with outputs of the encryption oracle. To model this, the challenge generator Θ can generate mutable fields and these fields can be read and modified by oracles and the verifier. For example, in the case of IND-CCA, Θ generates an empty (mutable) list *mem* in substitution θ by using instruction:

$$mem\theta := []$$

The left-right encryption oracle appends its output to the list whenever it is called. This is done by the following statement:

$$mem\theta := out :: mem\theta$$

Finally, the decryption oracle has to test that its argument *bs* does not appear in list *mem*. If *bs* appears, then the decryption oracle returns an error, for example bit 1. The test has the form:

$$\textbf{if } bs \in mem \textbf{ then return } 1$$

An important point is that as for challenge generated by Θ and stored in θ, the adversary does not have any access to the mutable fields (except using his oracles). In particular, he cannot corrupt these fields between oracle calls.

4.3 Examples

This section illustrates criteria on simple examples. The first two examples are based on two classical problems that are assumed to be hard to solve. The first problem is: given two large prime integers p and q, it is hard to obtain p and q from their product $p.q$, i.e. factorization is hard. The second problem is discrete logarithm, given the modular exponentiation g^x, it is difficult to obtain x. A problem is said to be *hard to solve*, if the probability of success of any polynomial time adversary is negligible. The third example is based on "one time padding". The last example is a very simple version of a classical board game called master mind [1].

4.3.1 Factoring Large Integers

Let $Prime(n)$ be an efficient algorithm that randomly generates a prime number between 2 and n such that each prime number between 2 and n has the same probability to be generated. The advantage of an adversary $\mathcal{A}$ is the probability that $\mathcal{A}$ manages to factor a product of two large prime integers:

$$Adv^F_{\mathcal{A}}(\eta) = Pr\big[p := Prime(2^\eta)\ ;\ q := Prime(2^\eta)\ ;\ \mathcal{A}(p.q) = (p,q)\big]$$

A classical assumption in cryptography is that the advantage of any adversary $\mathcal{A}$ is negligible in η.

4.3.2 Discrete Logarithm

Let $\mathbb{G}$ be a cyclic group of prime order q and let g be a generator of $\mathbb{G}$. The operation of the group applied to x and y is denoted by $x.y$. The exponentiation of an element x is denoted by x^n. The order q is assumed large, i.e. its number of digits is linear in η. We suppose that everyone knows g, $\mathbb{G}$ and q. The discrete logarithm problem is said *hard* if for any adversary $\mathcal{A}$, the probability that $\mathcal{A}$ deduces x from g^x is negligible. Formally, the advantage $\mathbf{Adv}^{DL}_{\mathcal{A}}(\eta)$ is negligible for any adversary η where the discrete logarithm advantage is defined by:

$$\mathbf{Adv}^{DL}_{\mathcal{A}}(\eta) = Pr\big[x \xleftarrow{R} [1,q]\ ;\ \mathcal{A}(g,q,g^x) = x\big]$$

This criterion can be described using our formalism. Let $\gamma = (\Theta; F; V)$ be a criterion such that:

1. Θ randomly generates an element x between 1 and q.
2. Oracle F does not use its argument and returns g, q and $g^{x\theta}$, $F(bs,\theta) = (g,q,g^{x\theta})$.
3. Verifier V tests that $\mathcal{A}$ correctly guessed x, $V = \lambda bs.bs = x$.

Application of proposition 4.1 allows us to deduce that $PrRand^\gamma$ is negligible. Therefore it is easy to deduce that safety using the two definitions of advantage are equivalent.

Proposition 4.4 *The advantage* $\mathbf{Adv}^{DL}_{\mathcal{A}}(\eta)$ *of any adversary* $\mathcal{A}$ *against discrete logarithm is negligible if and only if the advantage of any adversary* $\mathcal{B}$ *against criterion* γ *is negligible.*

4.3.3 One Time Padding

The One Time Pad (OTP) algorithm is a symmetric encryption scheme that is unbreakable when correctly used. Even brute force attacks are useless against OTP. The principle is to have key which length is the same as the length of the message that has to be encrypted. Let us consider that this fixed length is η. Then the key generation of OTP consists in randomly sampling a bit-string of length η. This bit-string k is the key or *pad*. Encryption of bit-string bs is given by

[1] See the board game section of http://en.wikipedia.org/wiki/Mastermind for details.

$bs \oplus k$. Decryption of bit-string bs returns $bs \oplus k$, hence applying decryption to $bs \oplus k$ returns $bs \oplus k \oplus k = bs$. Let us characterize the safety of OTP through a simple indistinguishability criterion γ defined by: challenge generator Θ randomly samples a bit b and a bit-string of length η denoted by k. Oracle F can only be called once, it takes as argument a pair of bit-string $\langle bs_0, bs_1 \rangle$ of length η and returns the encryption of bs_b using key k. In order to ensure that F is only called once, Θ generates a mutable boolean ac (for already called). This boolean is initialized to $false$. When F is called, if ac is $false$ then ac is set to $true$ and F answers its query, if ac is $true$ then F outputs the empty bit-string. Finally, verifier V checks that the adversary correctly guessed the value of bit b. As we suppose that random number generation is perfect, OTP cannot be broken.

Proposition 4.5 *Let $\mathcal{A}$ be an adversary against γ that can only output 0 or 1, then the advantage of $\mathcal{A}$ against γ is null.*

$$\forall \eta, \mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 0$$

Proof: Let $\mathcal{A}$ be an adversary which output is either 0 or 1 Using proposition 4.2, the advantage of $\mathcal{A}$ is given by:

$$\begin{aligned} \mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) &= Pr[k \xleftarrow{R} [0,1]^{\eta} \; ; \; \mathcal{A}(\eta)/\big(\lambda \langle bs_0, bs_1 \rangle . bs_1 \oplus k\big) = 1] \\ &\quad - Pr[k \xleftarrow{R} [0,1]^{\eta} \; ; \; \mathcal{A}(\eta)/\big(\lambda \langle bs_0, bs_1 \rangle . bs_0 \oplus k\big) = 1] \end{aligned}$$

As k is randomly sampled, the output of the oracle is the uniform distribution over bit-strings of length η. Hence the output is the same in both cases, thus the probability to answer 1 is the same. We finally get that:

$$\forall \eta, \mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 0$$

■

Note that if F can be called twice (or more), it is easy to break OTP. For this purpose, let us consider adversary $\mathcal{A}$ that submits $\langle 0^{\eta}, 1^{\eta} \rangle$ to his oracle. This adversary receives $b^{\eta} \oplus k$. Then $\mathcal{A}$ submits $\langle 0^{\eta}, 0^{\eta} \rangle$ to its oracle, the resulting bit-string is k. Hence, by computing $b^{\eta} \oplus k \oplus k$, $\mathcal{A}$ obtains b^{η} and thus can deduce the value of b.

4.3.4 The Mastermind Board Game

Here we describe a simple model of the classical mastermind game. This model is given using our criterion formalism. The main interest of this example is to illustrate that an adversary can act adaptively: he can choose new queries according to the results he received for previous queries.

A game session involves two players: the *code-maker* chooses an initial combination and the *code-breaker* tries to guess it. At the beginning of the game, the code-maker chooses a pattern. This pattern consists in a vector of four colors, each of these colors is chosen amongst six possibilities. Therefore, the number of possible patterns is 1296. The code-breaker can make ten guesses to find the pattern. Each of his guesses is itself a pattern. When the code-breaker submits a guess, he wins the game if the guess is correct. Otherwise, the code-maker tells him the number of correct colors he has in his pattern and the number of correct colors that occur at the correct place. Using this information, the code-breaker can try to make a new guess. If the code-breaker does not manage to win in ten guesses, then the code-maker wins the game.

Let us now describe this game using our formalism. The code-breaker is the adversary and the code-maker is represented by a random pattern generator. The set of patterns is $[0,5]^4$. The game can be represented by a criterion γ which is composed of:

- A challenge generator Θ, this algorithm randomly samples a pattern $\vec{p}$ among the 1296 possibilities.
- An oracle F, this oracle takes as argument a pattern $\vec{q}$ issued by the code-breaker. It returns two integers, the number of correct colors cc and the number of correctly placed colors cp.

These numbers can be obtained by the following functions:

$$\begin{aligned} cp(\vec{q},\vec{p}) &= \left|\left\{i \in [0,3] | \vec{p}[i] = \vec{q}[i]\right\}\right| \\ cc(\vec{q},\vec{p}) &= \max_{\sigma\in\Sigma}\left(cp(\vec{q},\vec{p}\sigma)\right) \end{aligned}$$

Where σ ranges over the set Σ of permutations of vectors of length 4. Oracle F may only be called ten times. To ensure this, Θ also generates a mutable integer which stores the number of calls made to F. Oracle F has to update this integer value and when it reaches ten, F always outputs the empty bit-string.

- A verifier V that takes as argument a pattern $\vec{q}$ and returns true iff patterns $\vec{p}$ and $\vec{q}$ are identical.

4.4 Basic Properties

We introduce here some very simple and intuitive properties on advantages. First, if we consider a game with no oracles, the advantage for any adversary is negative.

Proposition 4.6 *Let γ be the criterion $(\Theta;\epsilon;V)$, for any adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) \leq 0$$

Proof: By definition of $PrRand^{\gamma}$, for any adversary $\mathcal{A}$,

$$Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] \leq PrRand^{\gamma}$$

■

The same thing occurs when considering a game where the oracles and the verifier use two independent parts of the challenge. In this case, the adversary cannot get any useful information by using his oracles.

Proposition 4.7 *Let γ be the criterion $(\Theta_1,\Theta_2;F;V)$ such that F does not depend on the part of the challenge generated by Θ_2 and V does not depend on the part of the challenge generated by Θ_1. Then for any adversary $\mathcal{A}$ we have that:*

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) \leq 0$$

Proof: Let $\mathcal{A}$ be an adversary against γ and let γ' be the criterion $(\Theta_2;\epsilon;V)$. We build an adversary $\mathcal{A}'$ against γ' using $\mathcal{A}$.

Adversary $\mathcal{A}'(\eta)$:
$\theta_1 := \Theta_1(\eta)$
$out := \mathcal{A}(\eta)/\lambda bs.F(bs,\theta_1)$
return out

Then after unfolding the definition of $\mathcal{A}'$ it is easy to notice that the game involving $\mathcal{A}'$ against γ' and the game involving $\mathcal{A}$ against γ are exactly equivalent. Let us describe these games precisely:

Game $\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta)$:
$\theta_2 := \Theta_2(\eta)$
$\theta_1 := \Theta_1(\eta)$
$out := \mathcal{A}(\eta)/\lambda bs.F(bs,\theta_1)$
return $V(out,\theta_2)$

Game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$:
$\theta_1 := \Theta_1(\eta)$
$\theta_2 := \Theta_2(\eta)$
$out := \mathcal{A}(\eta)/\lambda bs.F(bs,\theta_1)$
return $V(out,\theta_2)$

Therefore, the probability of success of $\mathcal{A}$ and $\mathcal{A}'$ are the same:

$$Pr[\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta) = true] = Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true]$$

Moreover, the two $PrRand$ are equal as these games have the same verifier: any adversary against $(\Theta_1, \Theta_2; \epsilon; V)$ can be used against $(\Theta_2; \epsilon; V)$ with the same probability of success and the opposite is also true.

$$\mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta) = \mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta)$$

The previous proposition tells us that the advantage of $\mathcal{A}'$ is negative so the advantage of $\mathcal{A}$ is also negative. ■

Let us consider two games γ and γ' with the same challenge generator and verifier. If there is a way to transform the oracle from γ' into the oracle from γ, then any adversary against γ can be transformed into an adversary for γ' that has the same advantage. The "transformation" between oracles has to be computable in polynomial time.

Proposition 4.8 *Let $\gamma = (\Theta; F; V)$ and $\gamma' = (\Theta; F'; V)$ be two criteria. Assume that there exists a PRTM G such that for any bit-string bs and any challenge θ, the following equality holds:*

$$F(bs, \theta) = G\big(F'(bs, \theta)\big)$$

Then for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{A}'$ such that

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

Proof: Adversary $\mathcal{A}'$ against γ' can be built using $\mathcal{A}$:

Adversary $\mathcal{A}'(\eta)/\mathcal{O}$:
bs:=$\mathcal{A}(\eta)/\lambda s.G\big(\mathcal{O}(s)\big)$
return bs

Then by definition of G, experiments $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$ and $\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta)$ are equivalent. Moreover, $PrRand^{\gamma'}$ is equal to $PrRand^{\gamma}$ as both games have the same challenge generator and verifier. Hence, we obtain that:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

■

A reformulation of this proposition is the following : if we consider a game with oracle F' and another game whose oracle can be built from the output of F', then adversaries against the second game can be transformed into adversaries against the first game. Let us give some immediate corollaries for the previous proposition. First, if the advantage against γ' is p-negligible for any adversary, then this is also true for γ.

Corollary 4.1 *Let $\gamma = (\Theta; F; V)$ and $\gamma' = (\Theta; F'; V)$ be two games such that there exists a PRTM G such that for any bit-string bs and any challenge θ, $F(bs, \theta) = G\big(F'(bs, \theta)\big)$. If criterion γ' is safe, then criterion γ is also safe.*

It is also possible to extend the previous corollary to an equivalence: if there is also a way to transform the oracle from γ into the oracle from γ' then safety for the two games are equivalent.

Corollary 4.2 *Let $\gamma = (\Theta; F; V)$ and $\gamma' = (\Theta; F'; V)$ be two games. We suppose that there exist PRTM G and PRTM H such that for any bit-string bs and any challenge θ, $F(bs, \theta) = G\big(F'(bs, \theta)\big)$ and $F'(bs, \theta) = H\big(F(bs, \theta)\big)$. Criterion γ is safe if and only if criterion γ' is safe.*

The former proposition proves the relation between criteria where one oracle can be obtained from another by adding a computation layer. The computation layer may also occur directly on the arguments of the oracle.

Proposition 4.9 *Let $\gamma = (\Theta; F; V)$ and $\gamma' = (\Theta; F'; V)$ be two games. If there exists a PRTM G such that for any bit-string bs and any generated challenge θ,*

$$F(bs, \theta) = F'\big(G(bs), \theta\big)$$

Then for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{A}'$ such that

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

Proof: Adversary $\mathcal{A}'$ against γ' is built using $\mathcal{A}$ as a sub-routine.

Adversary $\mathcal{A}'(\eta)/\mathcal{O}$:
s:=$\mathcal{A}(\eta)/, \lambda s.\mathcal{O}(G(s))$
return s

Then by definition of G, experiments $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$ and $\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta)$ are exactly the same. Moreover, $PrRand^{\gamma'}$ is equal to $PrRand^{\gamma}$ as both games have the same challenge generators and verifiers. Hence, we get that:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

■

This proposition can also be reformulated in order to produce two corollaries similar to the previous ones.

Corollary 4.3 *Let $\gamma = (\Theta; F; V)$ and $\gamma' = (\Theta; F'; V)$ be two criteria. We suppose that there exists a PRTM G such that for any bit-string bs and any generated challenge θ, $F(bs, \theta) = F'\big(G(bs), \theta\big)$. If criterion γ' is safe, then criterion γ is also safe.*

It is also possible to reformulate this proposition as an equivalence: if there is also a way to transform the oracle from γ into the oracle from γ' then safety for the two games are equivalent.

Corollary 4.4 *Let $\gamma = (\Theta; F; V)$ and $\gamma' = (\Theta; F'; V)$ be two criteria. Let us suppose that there exist a PRTM G and a PRTM H such that for any bit-string bs and any generated challenge θ,*

$$F(bs, \theta) = F'\big(G(bs), \theta\big)$$

$$F'(bs, \theta) = F\big(H(bs), \theta\big)$$

Criterion γ is safe if and only if criterion γ' is also safe.

Finally it is possible to generalize propositions 4.8 and 4.9 by allowing G to use F' as an oracle. In this case, oracle F can be simulated by using algorithm G which makes possibly many queries to F'. However mutable fields from θ must have the same values after calling F or after executing G which queries to F'.

Proposition 4.10 *Let $\gamma = (\Theta; F; V)$ and $\gamma' = (\Theta; F'; V)$ be two games. If there exists a PRTM G such that for any bit-string bs and any generated challenge θ,*

$$F(bs, \theta) = G(bs)/\lambda bs'.F'(bs', \theta)$$

and the mutable fields from θ have the same values after executing both algorithms. Then for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{A}'$ such that

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

Proof: Adversary $\mathcal{A}'$ against γ' is built using $\mathcal{A}$ as a sub-routine.

Adversary $\mathcal{A}'(\eta)/\mathcal{O}$:
s:=$\mathcal{A}(\eta)/\lambda s.(G(s)/\mathcal{O})$
return s

Then by definition of G, experiments $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$ and $\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta)$ are exactly the same. Moreover, $PrRand^{\gamma'}$ is equal to $PrRand^{\gamma}$ as both games have the same challenge generators and verifiers. Hence, we get that:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

■

Corollaries similar to the previous one can also be written for this generalized proposition.

Formal Treatment of Probabilities In this chapter, probabilities have not been considered when performing the proofs. The proofs considered in this document can usually be carried out when formally considering probabilities. In order to illustrate this, the following result and its proof provide an explicit use of probabilities as introduced in section 3.3.2. Moreover, the complexity arguments have always been hidden in previous proof, the next proof also deals with complexity. The next proposition is close to previous propositions. Its only interest relies in its proof which illustrates arguments that are usually hidden in the other proofs.

Proposition 4.11 *Let $\gamma = (\Theta; F; V)$ and $(\Theta; F'; V)$ be two criteria. Let us suppose that there exist two PRTM G and H such that for any bit-string bs and for any θ generated by Θ, the probabilistic distributions $G(F'(H(bs), \theta))$ and $F(bs, \theta)$ are identical. Then for any adversary $\mathcal{A}$ against γ, there exists an adversary $\mathcal{A}$ against γ' such that:*

$$\forall \eta,\ \mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

Proof: Let $\mathcal{A}$ be an adversary against γ. To simplify the proof, we consider that $\mathcal{A}$ always flips $r_{\mathcal{A}}$ random coins and only calls the F oracle once. We also consider that the F and F' oracles always flip r_F and $r_{F'}$ coins respectively and that the G and H algorithms always flip r_G and r_H coins. Finally, the challenge generator Θ uses r_Θ coins and the verifier is deterministic. From now on, the random argument of probabilistic Turing machines is made explicit.

The equality between probabilistic distributions of $G \circ F' \circ H$ and F is defined by:

$$\forall out,\ \frac{\left|\left\{(R_H, R_{F'}, R_G) \middle| out = G(F'(H(bs, R_H), \theta, R_{F'}), R_G)\right\}\right|}{2^{r_G . r_{F'} . r_H}} = \frac{\left|\left\{R_F \middle| out = F(bs, R_F)\right\}\right|}{2^{r_F}}$$

Adversary $\mathcal{A}'$ uses $r_{\mathcal{A}'}$ random coins where $r_{\mathcal{A}'} = r_{\mathcal{A}} + r_G + r_H$. This new adversary uses $\mathcal{A}$ and is defined by:

Adversary $\mathcal{A}'(\vec{r})/\mathcal{O}$:
$s{:=}\mathcal{A}(\vec{r}_{r_{\mathcal{A}'} - r_{\mathcal{A}} .. r_{\mathcal{A}'}})/\lambda s.G\big(\mathcal{O}(H(s, \vec{r}_{1..r_H})), \vec{r}_{r_H+1..r_H+r_G}\big)$
return s

Then the advantage of adversary $\mathcal{A}'$ is given by:

$$\begin{aligned}
\mathbf{Adv}^{\gamma'}_{\mathcal{A}'} &= 2\frac{\left|\left\{(R_\Theta, R_{\mathcal{A}'}, R_{F'}) \middle| V(\mathcal{A}'(R_{\mathcal{A}'})/\lambda s.F'(s, \Theta(R_\Theta), R_{F'}))\right\}\right|}{2^{r_\Theta + r_{\mathcal{A}'} + r_{F'}}} - 2PrRand^{\gamma'} \\
&= 2\frac{\left|\left\{(R_\Theta, R_{\mathcal{A}}, R_H, R_G, R_{F'}) \middle| V(\mathcal{A}'(R_G, R_H, R_{\mathcal{A}})/\lambda s.out) \wedge out = F'(s, \Theta(R_\Theta), R_{F'})\right\}\right|}{2^{r_\Theta + r_{\mathcal{A}'} + r_{F'}}} \\
&\quad -2PrRand^{\gamma} \\
&\quad \text{Thus by using the equality of distributions} \\
&= 2\frac{\left|\left\{(R_\Theta, R_{\mathcal{A}}, R_H, R_G, R_{F'}) \middle| V(\mathcal{A}(R_{\mathcal{A}})/\lambda s.G(out, R_G)) \wedge out = F'(H(s, R_H), \Theta(R_\Theta), R_{F'})\right\}\right|}{2^{r_\Theta + r_{\mathcal{A}'} + r_{F'}}} \\
&\quad -2PrRand^{\gamma} \\
&= 2\frac{\left|\left\{(R_\Theta, R_{\mathcal{A}}, R_F) \middle| V(\mathcal{A}(R_{\mathcal{A}})/\lambda s.out') \wedge out' = F(s, \Theta(R_\Theta), R_F)\right\}\right|}{2^{r_\Theta + r_{\mathcal{A}'} + r_{F'}}} - 2PrRand^{\gamma} \\
&= \mathbf{Adv}^{\gamma}_{\mathcal{A}}
\end{aligned}$$

■

The proof presented here is quite complicated even with all the restrictive hypothesis we made. Handling probabilities is a very heavy but systematic task. Therefore, in the rest of this document, some probabilities are not considered explicitly. For example,

$$\forall bs, \theta,\ G(F'(H(bs), \theta)) = F(bs, \theta)$$

denotes equality of the two distributions $G(F'(H(bs), \theta))$ and $F(bs, \theta)$.

4.5 Security Criteria for Encryption and Signature Schemes

We present here some variants of classical criteria. The main difference is the addition of patterns. Patterns allow the use of secret information with the oracle: for example, an adversary can ask for encryption of a secret key with another key, this cannot be achieved with classical oracles. We introduce pattern because in the protocols we consider, messages can contain such secret information. Therefore allowing the adversary to manipulate these information helps us to simply the soundness proof given later in this document.

4.5.1 Asymmetric Encryption: N-PAT-IND-CCA

The classical way to define safety of an asymmetric encryption scheme was given in section 3.4. Criterion IND-CCA is extended in this section in such a way that the attacker can ask for the encryption of a secret key. As the IND-CCA requirement is very strong, the intuition is that the adversary cannot get non-negligible advantage by using these new requests. We will see in the following that this is true if and only if we do not allow key cycles.

Let $(\mathcal{KG}^a, \mathcal{E}^a, \mathcal{D}^a)$ be an asymmetric encryption scheme that is used in the different oracles and in the challenge generator. Let N be an integer. In criterion N-PAT-IND-CCA, N pairs of keys are generated, these keys are numbered from 1 to N. Key pair number i is composed of the public part pk_i and the secret part sk_i.

As we want the adversary to be able to manipulate secret keys but we do not want the adversary to be given the secret keys, the adversary does not submit bit-strings to his oracles anymore. Instead, the adversary uses patterns where he can ask for inclusion of the secret keys.

Patterns

Pattern terms are a mix between symbolic messages and bit-strings: indeed these are messages where atoms can be either a bit-string or a pattern variable. These variables represent the different challenge secret keys and are denoted by $[i]$, variable $[i]$ asks the oracle to replace the pattern variable by the bit-string value of secret key number i, i.e. sk_i. Variables can be used as atomic messages (data pattern) or at a key position (key pattern). When a left-right oracle is given a pattern term, it replaces patterns by values of corresponding keys and encodes the so-obtained bit-string. More formally, patterns are given by the following grammar where bs is a bit-string and i is an integer.

$$\begin{array}{rcll} pat & ::= & \langle pat, pat\rangle \mid bs \mid [i] & \\ & & \mid \{pat\}^a_{bs} \mid \{pat\}^a_{[i]} & \text{asymmetric encryption} \end{array}$$

This grammar defines asymmetric patterns as the only cryptographic scheme that can be used by patterns in this section is asymmetric encryption. Further in this document, we define more general patterns that can use symmetric encryption, digital signature or even a whole cryptographic library. An important remark is that any encryption performed in a pattern has to use the same security parameter η. This restriction is necessary as we want to be able to control the length of the evaluation of a pattern.

The computation (also called concretization) made by the oracle is easily defined recursively. It uses a context θ associating bit-string values to the different keys. For an integer i, there exists a related encryption key pk_i and a related decryption key sk_i in θ. The concretization produces a bit-string and it uses the encryption algorithm $\mathcal{E}^a$ of the asymmetric encryption scheme and the computational version of concatenation which is denoted by operator "$\cdot$". The concretization algorithm is detailed thereafter:

Algorithm $concr(pat, \theta)$:
  match pat **with**
    $bs \rightarrow$ **return** bs

$$
\begin{array}{ll}
[i] \rightarrow & \textbf{return } sk_i\theta \\
\langle p_1, p_2\rangle \rightarrow & \textbf{return } concr(p_1,\theta)\cdot concr(p_2,\theta) \\
\{p\}^a_{[i]} \rightarrow & \textbf{return } \mathcal{E}^a(concr(p,\theta), pk_i\theta) \\
\{p\}^a_{bs} \rightarrow & \textbf{return } \mathcal{E}^a(concr(p,\theta), bs)
\end{array}
$$

endmatch

A key k is *asked* by a pattern pat if applying the $concr$ algorithm to pat uses $k\theta$. Indeed a public key pk_i is asked by pat if $[i]$ occurs at a key position in pat and a secret key sk_i is asked by pat if $[i]$ appears elsewhere in pat. In order for $concr$ not to produce an error on pat and θ, all the keys asked by pat have to be in $sup(\theta)$.

Patterns and Encryption Size When defining IND-CCA, a natural restriction is that whenever the adversary submits a pair of bit-strings $\langle bs_0, bs_1\rangle$ to his left-right encryption oracle, bs_0 and bs_1 must have the same length, otherwise the oracle does not answer correctly. It is not possible to apply the same restriction on patterns as pattern length is not properly defined. Therefore equivalent patterns are introduced: the idea is that if two patterns are equivalent, then their concretizations have the same size when using an encryption scheme that satisfies the fixed encryption size hypothesis. Then the restriction on the left-right encryption oracle is that the two submitted patterns must be equivalent in order for the oracle to answer correctly.

Definition 4.1 (Equivalent Patterns) *Two patterns pat_0 and pat_1 are equivalent if $pat_0 \approx pat_1$ where the $\approx$ relation is inductively defined by the following rules: two bit-string are equivalent if they have the same length.*

$$\frac{\ell(bs_0) = \ell(bs_1)}{bs_0 \approx bs_1} bs$$

Two pairs or encoding are equivalent if the underlying patterns are equivalent:

$$\frac{pat_0 \approx pat_1 \quad pat'_0 \approx pat'_1}{\langle pat_0, pat'_0\rangle \approx \langle pat_1, pat'_1\rangle} pair \qquad \frac{pat_0 \approx pat_1}{\{pat_0\}^a_u \approx \{pat_1\}^a_v} enc$$

Finally, pattern variables are always equivalent:

$$\frac{}{[i] \approx [j]} pat$$

The asymmetric encryption scheme used here $(\mathcal{KG}^a, \mathcal{E}^a, \mathcal{D}^a)$ is assumed to verify the fixed encryption size hypothesis. Therefore, it is possible to deduce that the concretization of two equivalent patterns leads to two bit-strings that have the same length.

Proposition 4.12 *Let pat_0 and pat_1 be two patterns such that $pat_0 \approx pat_1$, then for any θ such that $sup(\theta)$ contains all the keys asked by pat_0 and pat_1, the concretizations of pat_0 and pat_1 have the same length:*

$$\ell\big(concr(pat_0,\theta)\big) = \ell\big(concr(pat_1,\theta)\big)$$

Proof: The proof proceeds by induction upon the proof structure of $pat_0 \approx pat_1$. Let us consider the different case for the last deduction rule.

1. If the last rule is bs, then $pat_0 = bs_0$, $pat_1 = bs_1$ and $\ell(bs_0) = \ell(bs_1)$. As $concr(bs,\theta)$ returns bs for any bit-string bs, we immediately get that $concr(bs_0,\theta)$ and $concr(bs_1,\theta)$ have the same length.

2. If the last rule is pat, then $pat_0 = [i]$ and $pat_1 = [j]$. Thus we have that $concr(pat_0,\theta) = sk_i\theta$ and $concr(bs_1,\theta) = sk_j\theta$. As part of the fixed encryption size hypothesis, the length of secret keys is constant.

3. If the last rule is *pair*, then $pat_0 = \langle pat'_0, pat''_0 \rangle$ and $pat_1 = \langle pat'_1, pat''_1 \rangle$ where pat'_0 and pat'_1 are equivalent and pat''_0 and pat''_1 are also equivalent. The induction argument tells us that:

$$\ell\big(concr(pat'_0, \theta)\big) = \ell\big(concr(pat'_1, \theta)\big) \text{ and } \ell\big(concr(pat''_0, \theta)\big) = \ell\big(concr(pat''_1, \theta)\big)$$

   As concatenation preserves length equalities, we get the result.

4. Finally, if the last rule is *enc*, then $pat_0 = \{pat'_0\}_u$ and $pat_1 = \{pat'_1\}_v$ where pat'_0 and pat'_1 are equivalent. As the security parameter η is fixed, applying the fixed encryption size hypothesis gives us the awaited result.

■

Acyclicity Condition Key cycles are a well-known problem in cryptography. It appears that for some real encryption schemes like DES, encrypting a key with itself can lead to information leaks. Moreover, the IND-CCA criterion and similar criteria are not strong enough to prevent such leaks. This limitation has already been stated for IND-CPA in [AR00]. To avoid cycles, we put the following restriction on patterns submitted to the left-right oracle. When asking for encryption of pattern *pat* with key pk_i, *pat* must not contain any pattern variable $[j]$ for j lower than i or equal to i. The end of next section explains why this restriction is necessary.

Oracles and Criterion

Criterion N-PAT-IND-CCA is defined by $(\Theta; F; V)$. The challenge generator Θ produces N pairs of keys using $\mathcal{KG}^a$ and outputs a substitution θ linking pk_i to the public part of the i^{th} key and sk_i to its secret part. It also creates for each key a mutable field mem_i^a which is used to store the output of the left-right oracle. A bit b is also randomly generated, adversaries try to guess the value of this bit b.

Algorithm $\Theta(\eta)$:
  $\theta := []$
  $b\theta \xleftarrow{R} [0,1]$
  for i **from** 1 **to** N
    $(bspk, bssk) := \mathcal{KG}^a(\eta)$
    $pk_i\theta := bspk$
    $sk_i\theta := bssk$
    $mem_i^a\theta := []$
  endfor
  return θ

The oracle part F is composed of 3 oracles for each key pair (thus F contains $3.N$ oracles). The first one is the left-right encryption oracle. This oracle receives a pair of patterns $\langle pat_0, pat_1 \rangle$. It checks that these patterns verify the acyclicity condition and are equivalent. If so, it selects either pat_0 or pat_1 according to the value of bit b from θ, concretizes it using θ and encrypts the result using public key pk_i from θ. As the decryption oracles should not accept to work on the outputs of this oracle, the result of the encryption is stored in the list mem_i^a.

Oracle $ELR_i^a(\langle pat_0, pat_1 \rangle, \theta)$:
  if $\exists j \in var(\langle pat_0, pat_1 \rangle), j \leq i$ **or not** $pat_0 \approx pat_1$ **then**
    return 1
  else
    $bs := concr(pat_{b\theta}, \theta)$
    $out := \mathcal{E}^a(bs, pk_i\theta)$
    $mem_i^a\theta := out :: mem_i^a\theta$
    return *out*
  endif

The second oracle is the decryption oracle related to key i. This algorithm first verifies that its argument has not been produced by the left-right encryption oracle before, if this is not the case it decrypts its argument using secret key sk_i from θ.

Oracle $D_i(bs, \theta)$:
  if $bs \in mem_i^a\theta$ **then**
    return 1
  else
    return $\mathcal{D}^a(bs, sk_i\theta)$
  endif

The last oracle related to key i simply returns the bit-string value for key pk_i, it does not depend on its argument bs.

Oracle $PK_i(bs, \theta)$:
  return $pk_i\theta$

Finally, verifier V tests that the adversary correctly deduced the value of bit b. The algorithm that implements V is simple:

Verifier $V(bs, \theta)$:
  return $bs = b\theta$

An adversary wins against N-PAT-IND-CCA if he manages to deduce the value of the challenge bit b.

Definition 4.2 (Pattern Semantic Security) *An asymmetric encryption scheme is said to be N-PAT-IND-CCA if it is safe for N-PAT-IND-CCA, i.e. the advantage of any adversary against our criterion using this encryption scheme is negligible.*

Note that, although close, 1-PAT-IND-CCA and IND-CCA are different criteria. Indeed, in 1-PAT-IND-CCA, an adversary can submit patterns like $\langle bs, bs' \rangle$ to his left-right encryption oracle whereas it is not possible in IND-CCA.

It is easy to relate IND-CCA and N-PAT-IND-CCA in one way.

Proposition 4.13 *Let N be an integer greater than 0. If an asymmetric encryption scheme $(\mathcal{KG}^a, \mathcal{E}^a, \mathcal{D}^a)$ is safe for N-PAT-IND-CCA, then it is also safe for IND-CCA.*

Proof: The sketch of the proof is easy to understand: let $\mathcal{A}$ be an adversary against IND-CCA. This adversary can be used against N-PAT-IND-CCA with the same advantage. In this situation, $\mathcal{A}$ queries the oracles related to the first key. As the asymmetric encryption scheme is safe for N-PAT-IND-CCA, the advantage of $\mathcal{A}$ against N-PAT-IND-CCA is negligible. Consequently, the advantage of $\mathcal{A}$ against IND-CCA is negligible and so the encryption scheme is safe for IND-CCA. As this is the first proof of this type, we describe the different steps thereafter.

When formalizing the proof, the only difficulty is that $\mathcal{A}$ cannot be used against N-PAT-IND-CCA: queries made by $\mathcal{A}$ have to be redirected to the oracles corresponding to one of the key (the first one for example). Queries made by $\mathcal{A}$ to the left-right oracle are pairs of bit-strings that can also be used as a pair of patterns. Hence, if $\mathcal{A}$ is an adversary against IND-CCA then $\mathcal{B}$ is an adversary against N-PAT-IND-CCA that uses $\mathcal{A}$ as a sub-routine.

Adversary $\mathcal{B}(\eta)/\mathcal{O}_1^{pk}, \mathcal{O}_1^{LR}, \mathcal{O}_1^{D}, ..., \mathcal{O}_N^{pk}, \mathcal{O}_N^{LR}, \mathcal{O}_N^{D}$:
  $bs := \mathcal{A}(\eta)/\mathcal{O}_1^{pk}, \mathcal{O}_1^{LR}, \mathcal{O}_1^{D}$
  return bs

By using proposition 4.2, we get that the advantage of $\mathcal{B}$ is given by:

$$\begin{aligned} \mathbf{Adv}_{\mathcal{B}}^{\gamma}(\eta) \quad = \quad & Pr[\theta := \Theta \ ; \ \mathcal{B}(\eta)/\lambda s.F(s,\theta) = 1 \mid b\theta = 1] \\ & - Pr[\theta := \Theta \ ; \ \mathcal{B}(\eta)/\lambda s.F(s,\theta) = 1 \mid b\theta = 0] \end{aligned}$$

Where $\gamma = (\Theta, F, V)$ is criterion N-PAT-IND-CCA. Let o_i^b be the three oracles related to the pair of keys (pk_i, sk_i) where b is the challenge bit generated by Θ and used by the left-right oracles. Hence by developing Θ and F, the advantage of $\mathcal{B}$ becomes:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\gamma}(\eta) &= Pr[(pk_1, sk_1) := \mathcal{KG}(\eta), ..., (pk_N, sk_N) := \mathcal{KG}(\eta) \ ; \ \mathcal{B}(\eta)/o_1^1, ..., o_n^1 = 1] \\ &\quad -Pr[(pk_1, sk_1) := \mathcal{KG}(\eta), ..., (pk_N, sk_N) := \mathcal{KG}(\eta) \ ; \ \mathcal{B}(\eta)/o_1^0, ..., o_n^0 = 1]\end{aligned}$$

Now it is possible to replace $\mathcal{B}$ by its implementation that uses $\mathcal{A}$.

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\gamma}(\eta) &= Pr[(pk_1, sk_1) := \mathcal{KG}(\eta), ..., (pk_N, sk_N) := \mathcal{KG}(\eta) \ ; \ \mathcal{A}(\eta)/o_1^1 = 1] \\ &\quad -Pr[(pk_1, sk_1) := \mathcal{KG}(\eta), ..., (pk_N, sk_N) := \mathcal{KG}(\eta) \ ; \ \mathcal{A}(\eta)/o_1^0 = 1]\end{aligned}$$

Running the adversary $\mathcal{A}$ does not use any other key pair than (pk_1, sk_1). Thus, we finally get that:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\gamma}(\eta) &= Pr[(pk_1, sk_1) := \mathcal{KG}(\eta) \ ; \ \mathcal{A}(\eta)/o_1^1 = 1] - Pr[(pk_1, sk_1) := \mathcal{KG}(\eta) \ ; \ \mathcal{A}(\eta)/o_1^0 = 1] \\ &= \mathbf{Adv}_{\mathcal{A}}^{IND}(\eta)\end{aligned}$$

Hence for any adversary $\mathcal{A}$ against IND-CCA, there exists an adversary $\mathcal{B}$ against N-PAT-IND-CCA that has the same advantage. The encryption scheme is safe for N-PAT-IND-CCA thus $\mathbf{Adv}_{\mathcal{B}}^{\gamma}$ is negligible and so $\mathbf{Adv}_{\mathcal{A}}^{IND}$ is also negligible. Therefore the encryption scheme is safe for IND-CCA. ■

The converse of this proposition is also true. However its proof is far more complicated and uses the partition theorem introduced in chapter 5. There is a simple proof in the case $N = 1$ as the oracles from 1-PAT-IND-CCA can be simulated using oracles from IND-CCA.

Proposition 4.14 *An asymmetric encryption scheme $(\mathcal{KG}^a, \mathcal{E}^a, \mathcal{D}^a)$ is safe for 1-PAT-IND-CCA if and only if it is safe for IND-CCA.*

Proof: A first implication comes from proposition 4.13. Safety against 1-PAT-IND-CCA implies safety against IND-CCA. The converse is less trivial, queries to the left-right oracle can be pairs of patterns instead of just pairs of bit-strings. However, the acyclicity of patterns ensures that patterns must have the following form:

$$pat ::= \langle pat, pat \rangle \mid \{pat\}_{bs}^{a} \mid bs$$

There are no variables in these patterns. Thus, applying the *concr* algorithm with an empty environment produces a bit-string. It is possible to apply proposition 4.9. Let γ' denote criterion IND-CCA defined by the triple $(\Theta; F'; V)$ and γ denote criterion 1-PAT-IND-CCA defined by $(\Theta; F; V)$. Then the public keys and decryption oracles from F and F' are identical. The left-right encryption oracle F^{LR} from γ is linked to the left-right encryption oracle F'^{LR} from γ' by the following relation:

$$\forall pat_0, pat_1, F^{LR}\big(\langle pat_0, pat_1 \rangle, \theta\big) = F'^{LR}\big(concr(\langle pat_0, pat_1 \rangle, []), \theta\big)$$

By applying proposition 4.9, we get that for any adversary $\mathcal{A}$ against 1-PAT-IND-CCA (γ), there exists an adversary $\mathcal{A}'$ against IND-CCA (γ') such that:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = \mathbf{Adv}_{\mathcal{A}'}^{\gamma'}(\eta)$$

Criterion IND-CCA is supposed to be verified by our asymmetric encryption scheme. As a consequence, the advantage of $\mathcal{A}'$ is negligible and so the advantage of $\mathcal{A}$ is also negligible. The encryption scheme is also safe for 1-PAT-IND-CCA. ■

More on Key Cycles

In this paragraph, we assume that there exists an encryption scheme $\mathcal{AE}$ that is safe for IND-CCA. Our objective here is to show that key cycles create insecurity. Let N-PAT$_c$-IND-CCA be exactly the same criterion has N-PAT-IND-CCA except that the left-right encryption oracle is modified in order to allow cycles of length greater than c. Formally, the acyclicity restriction from N-PAT-IND-CCA is relaxed in N-PAT$_c$-IND-CCA. The $\prec$ order is defined as follows: if the adversary queries the encryption oracle related to key pk_j with a pattern that contains $[i]$, then we have that $j \prec i$. The restriction is that the adversary should not create cycles for $\prec$ which length is lower than or equal to c (a cycle of length c is $i_1 \prec i_2 \prec \cdots \prec i_c$), otherwise the oracle outputs the empty bit-string. Then this new criterion is not equivalent to IND-CCA, more precisely:

Cycles of Length 1 The idea is to build an encryption scheme such that patterns of the following form are easy to distinguish.

$$\{sk\}_{pk}$$

We assume that there exists a polynomial-time algorithm Inv such that $Inv(sk, pk)$ returns true iff sk is the secret key related to pk. We build another asymmetric encryption scheme $\mathcal{AE}' = (\mathcal{KG}, \mathcal{E}', \mathcal{D}')$ using $\mathcal{AE}$.

Algorithm $\mathcal{E}'(m, pk) =$
  if $Inv(m, pk)$
    return $0 \cdot m$
  else
    return $1 \cdot \mathcal{E}(m, pk)$

Algorithm $\mathcal{D}'(m, sk) =$
  if $m = 0 \cdot m$
    return m
  else if $m = 1 \cdot n$
    return $\mathcal{D}(n, sk)$

The asymmetric encryption scheme we just defined is a counter-example to the assumption that 1-PAT$_1$-IND-CCA where cycles are allowed is equivalent to IND-CCA.

Proposition 4.15 *If $\mathcal{AE}$ is secure against IND-CCA, then $\mathcal{AE}'$ is also secure against IND-CCA.*

Proof: Let $\mathcal{A}'$ be an adversary against IND-CCA using $\mathcal{AE}'$ (γ'). Then the adversary $\mathcal{A}$ against IND-CCA using $\mathcal{AE}$ (γ) is defined by:

Adversary $\mathcal{A}(\eta)/\mathcal{O}_{pk}, \mathcal{O}_{LR}, \mathcal{O}_D$:
  $pk := \mathcal{O}_{pk}$
  $E'_{LR} := \lambda(m_0, m_1)$.**if** $Inv(m_0, pk)$, deduce sk
    else if $Inv(m_1, pk)$, deduce sk
    else $1 \cdot \mathcal{O}_{LR}(m_0, m_1)$
  $D' := \lambda m$. **if** $m = 0 \cdot n$, n
    else $m = 1 \cdot n$, $\mathcal{O}_D(n)$
  $d := \mathcal{A}'(\eta)/pk, E'_{LR}, D'$
  return d

If $\mathcal{A}$ deduces sk, it is clear that it wins its challenge. Consequently if $\mathcal{A}'$ wins its challenge, then $\mathcal{A}$ also wins its challenge.

$$Pr[\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta) = true] \leq Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true]$$

Moreover, as both criteria have the same generator and verifier, their $PrRand$ is equal. Hence,

$$\mathbf{Adv}^{\gamma'}(\mathcal{A}') \leq \mathbf{Adv}^{\gamma}(\mathcal{A})$$

As $\mathcal{AE}$ is IND-CCA and $\mathcal{A}$ is a PRTM, the advantage of $\mathcal{A}$ is negligible. Hence, the advantage of $\mathcal{A}'$ is also negligible and $\mathcal{AE}'$ is also IND-CCA. ∎

Proposition 4.16 *$\mathcal{AE}'$ is not secure against 1-PAT$_1$-IND-CCA.*

Proof: Let us consider the adversary $\mathcal{A}$ that simply queries his left-right encryption oracle with $([1], 0^{n(\eta)})$ where $n(\eta)$ is the length of a secret key for security parameter η. Then $\mathcal{A}$ can easily deduce the value of the challenge bit b: if the result of the oracle call starts with 0, b equals 0 else b equals 1. Formally adversary $\mathcal{A}$ is described as follows:

Adversary $\mathcal{A}(\eta)/\mathcal{O}_{pk}, \mathcal{O}_{LR}, \mathcal{O}_D$:
$bs := \mathcal{O}_{LR}([1], 0^{n(\eta)})$
if $bs[0] = 0$ **then return** 0
else return 1

The advantage of $\mathcal{A}$ is non-negligible (the only case where $\mathcal{A}$ fails if is the secret key generated in the experiment is $0^{n(\eta)}$ and this has only negligible probability to happen). ■

By combining the two previous results, it is clear that an encryption scheme can be IND-CCA and not verify 1-PAT$_1$-IND-CCA. Therefore, we get the following corollary.

Corollary 4.5 *IND-CCA does not imply* 1-*PAT*$_1$-*IND-CCA.*

An immediate consequence is that key cycles represent real security problems, thus such cycles should not occur in correctly designed protocols.

Cycles of Length 2 The idea is to build an encryption scheme such that patterns of the following form are easy to distinguish.

$$\left\{\{sk_1\}_{pk_2}, sk_2\right\}_{pk_1}$$

We build another asymmetric encryption scheme $\mathcal{AE}' = (\mathcal{KG}, \mathcal{E}', \mathcal{D}')$ using $\mathcal{AE}$.

Algorithm $\mathcal{E}'(m, pk) =$
  if $m = \langle u, v\rangle$ **and** $Inv(\mathcal{D}(u, v), pk)$
    return $0 \cdot \langle u, v\rangle$
  else
    return $1 \cdot \mathcal{E}(m, pk)$

Algorithm $\mathcal{D}'(m, sk) =$
  if $m = 0 \cdot \langle u, v\rangle$
    return $\langle u, v\rangle$
  else if $m = 1 \cdot n$
    return $\mathcal{D}(n, sk)$

Proposition 4.17 *If $\mathcal{AE}$ is secure against IND-CCA, then $\mathcal{AE}'$ is also secure against IND-CCA.*

Proof: Let $\mathcal{A}'$ be an adversary against IND-CCA using $\mathcal{AE}'$ (γ'). Then the adversary $\mathcal{A}$ against IND-CCA using $\mathcal{AE}$ (γ) is defined by:

Adversary $\mathcal{A}(\eta)/\mathcal{O}_{pk}, \mathcal{O}_{LR}, \mathcal{O}_D$:
$pk := \mathcal{O}_{pk}$
$E'_{LR} := \lambda(m_0, m_1).$**if** $m_0 = \langle u, v\rangle$ **and** $Inv(\mathcal{D}(u, v), pk)$, deduce sk
  else if $m_1 = \langle u, v\rangle$ **and** $Inv(\mathcal{D}(u, v), pk)$, deduce sk
  else $\mathcal{O}_{LR}(m_0, m_1)$
$D' := \lambda m.$ **if** $m = 0 \cdot n$, n
  else $m = 1 \cdot n$, $\mathcal{O}_D(n)$
$d := \mathcal{A}'(\eta)/pk, E'_{LR}, D'$
return d

If $\mathcal{A}$ deduces sk, it is clear that he wins his challenge. And consequently, $\mathcal{A}$ wins its challenge more often than $\mathcal{A}'$ does.

$$Pr[\mathbf{Exp}^{\gamma'}_{\mathcal{A}'}(\eta) = true] \leq Pr[\mathbf{Exp}^{\gamma}_{\mathcal{A}}(\eta) = true]$$

Moreover, as both criteria have the same generator and verifier, their $PrRand$ is equal. Hence,

$$\mathbf{Adv}^{\gamma'}(\mathcal{A}') \leq \mathbf{Adv}^{\gamma}(\mathcal{A})$$

As $\mathcal{AE}$ is IND-CCA and $\mathcal{A}$ is a PRTM, the advantage of $\mathcal{A}$ is negligible. Hence, the advantage of $\mathcal{A}'$ is also negligible and $\mathcal{AE}'$ is also IND-CCA. ■

Proposition 4.18 *$\mathcal{AE}'$ is not secure against 2-PAT_2-IND-CCA.*

Proof: An adversary can ask for the encryption of $[sk_1]$ using the left-right oracle related to pk_2. Let bs be the result. Then he asks the encryption of $\langle bs, [sk_2]\rangle$ using the left-right oracle related to pk_1. If the final result starts with a 0, then $b = 0$ else $b = 1$. ■

By combining the two previous results, it is clear that an encryption scheme can be IND-CCA and not verify 2-PAT_2-IND-CCA.

Corollary 4.6 *IND-CCA does not imply 2-PAT_2-IND-CCA.*

We did not really prove that cycle of length 2 are a security threat. Indeed we use a cycle of length 1 but use two oracle queries to build it. Hence from the oracle point of view, cycles of length 1 are forbidden but it is still possible to build such cycles as cycles of length 2 are authorized. Let us now generalize this result to a length n.

Generalization to Arbitrary Length Let n be the cycle length. Then it is possible to generalize previous results by distinguishing such patterns:

$$\left\{...\left\{\{sk_1\}_{pk_2}, sk_2\right\}_{pk_3}..., sk_n\right\}_{pk_1}$$

The main result is:

Proposition 4.19 *Let m and n be two strictly positive integers, IND-CCA does not imply m-PAT_n-IND-CCA.*

Key Dependent Message (KDM) Security

In N-PAT-IND-CCA, the only operation that is allowed on secret keys is to copy their value. This operation is sufficient when considering security protocols however it is possible to define a more general criterion where an adversary can ask for any operations on secret keys. Thus KDM security has been introduced in [BRS01] as an extension of IND-CPA. However this can be adapted to the case of IND-CCA in a straightforward way. Hence we define N-KDM-IND-CCA in our criterion model as criterion $(\Theta; F; V)$ where Θ and V are the same as in N-PAT-IND-CCA. F still contains the decryption oracle and the public key oracle but its left-right encryption oracle is modified: it does not operate on patterns any more but on functions. Indeed, the left-right encryption oracles receive two functions f_0 and f_1 (represented by two PRTM) such that for any bit-string bs, $f_0(bs)$ and $f_1(bs)$ have the same length. The oracle applies f_b to secret keys and encrypt the result. Formally,

Oracle $ELR_i^a(\langle f_0, f_1\rangle, \theta)$:
$bs_0 := f_0(pk_1\theta, sk_1\theta \cdots pk_N\theta, sk_N\theta)$
$bs_1 := f_1(pk_1\theta, sk_1\theta \cdots pk_N\theta, sk_N\theta)$
if $\ell(bs_0) \neq \ell(bs_1)$ **then return** 1
$out := \mathcal{E}^a(bs_b, pk_i\theta)$
$mem_i^a\theta := out :: mem_i^a\theta$
return out

Definition 4.3 (Key Dependent Semantic Security) *An asymmetric encryption scheme is said to be N-KDM-IND-CCA if it is safe for N-PAT-IND-CCA, i.e. the advantage of any adversary against our criterion using this encryption scheme is negligible.*

Note that, although close, 1-PAT-IND-CCA and IND-CCA are different criteria. Indeed, in This criterion is very close to N-PAT-IND-CCA but more general as it allows any operations on keys, not just concatenation and encryption.

Proposition 4.20 *Let $\mathcal{AE}$ be an asymmetric encryption scheme. If $\mathcal{AE}$ is safe for N-KDM-IND-CCA then it is safe for N-PAT-IND-CCA.*

Proof: Let $\mathcal{AE}$ be an asymmetric encryption scheme that is safe for N-KDM-IND-CCA. Let $\mathcal{A}$ be an adversary against criterion N-PAT-IND-CCA (denoted by γ in the following). We build an adversary $\mathcal{B}$ against N-KDM-IND-CCA that has the same advantage. Criterion N-KDM-IND-CCA is denoted by δ. In order to give a simple definition of adversary $\mathcal{B}$, we only consider the oracles related to one key. The description proposed here can easily be adapted to the case of N keys. The only difference between γ and δ is that requests to the left-right encryption oracles differ. However, pattern requests (made by $\mathcal{A}$) can be converted to function requests so queries made by $\mathcal{A}$ can be converted into queries that $\mathcal{B}$ can perform. This conversion is made by function $p2f$ which is described below (and can be represented by a PRTM).

Adversary $\mathcal{B}(\eta)/\mathcal{O}_{LR}, \mathcal{O}_D, \mathcal{O}_{PK}$
  $d := \mathcal{A}/\lambda\langle pat_0, pat_1\rangle.\mathcal{O}_{LR}(\langle p2f(pat_0), p2f(pat_1)\rangle)$
    $\mathcal{O}_D$
    $\mathcal{O}_{PK}$
  return d

The pattern to function conversion function $p2f$ is given by:

Algorithm $p2f(pat)$:
  return $\lambda(pk_1^v, sk_1^v, ...).concr\big(pat, \{pk_1 \to pk_1^v, sk_1 \to sk_1^v, ...\}\big)$

Then the behavior of the pattern encryption oracle confronted to pat is identical to the behavior of the function encryption oracle confronted to $p2f(pat)$. The game where $\mathcal{B}$ is confronted to δ is the same as the game where $\mathcal{A}$ is confronted to γ. Therefore the advantages of $\mathcal{A}$ and $\mathcal{B}$ are equal. As $\mathcal{AE}$ is secure against N-KDM-IND-CCA, the advantage of $\mathcal{B}$ is negligible so the advantage of $\mathcal{A}$ is also negligible. As this is true for any adversary $\mathcal{A}$, it is possible to conclude that $\mathcal{AE}$ is secure against N-PAT-IND-CCA. ■

In the next chapter, we prove that if we assume an acyclicity hypothesis, then criterion IND-CCA and N-KDM-IND-CCA are equivalent. This hypothesis prevents the adversary from asking any operation on keys sk_j to the oracle related to key number i when $j \leq i$. We assume that this acyclicity hypothesis is part of the criterion in the following, thus we use a new left-right encryption oracle:

Oracle $ELR_i^a(\langle f_0, f_1\rangle, \theta)$:
  $bs_0 := f_0(pk_{i+1}\theta, sk_{i+1}\theta \cdots pk_N\theta, sk_N\theta)$
  $bs_1 := f_1(pk_{i+1}\theta, sk_{i+1}\theta \cdots pk_N\theta, sk_N\theta)$
  if $\ell(bs_0) \neq \ell(bs_1)$ **then return** 1
    $out := \mathcal{E}^a(bs_b, pk_i\theta)$
    $mem_i^a\theta := out :: mem_i^a\theta$
    return out
  endif

Then the oracle related to key pk_i can only manipulate keys (pk_j, sk_j) for j greater than i.

4.5.2 Digital Signature: N-UNF

The N-UNF criterion is an extension of the UNF criterion detailed before. This extension considers a multi-user setting instead of having just a single user.

The main requirement is that an adversary should not be able to forge a pair containing a bit-string bs and the signature of bs using the secret signature key. An N-UNF adversary $\mathcal{A}$ is given N verification keys and has to produce a message and its signature under one of the keys. The adversary also has access to N signature oracles $\mathcal{S}_{sik_i}(.)$.

Formally, the N-UNF criterion uses a signature scheme $\mathcal{SS} = (\mathcal{KG}^g, \mathcal{S}, \mathcal{V})$. This criterion is defined by $(\Theta; F; V)$ where Θ generates N signature key pairs using algorithm $\mathcal{KG}^g$, it also generates a list mem_i^g for each key pair in order to store the outputs of the signature oracles.

Algorithm $\Theta(\eta)$:
$\theta := []$
for i **from** 1 **to** N
$(bssk, bsvk) := \mathcal{KG}^g(\eta)$
$sik_i\theta := bssk$
$vk_i\theta := bsvk$
$mem_i^g\theta := []$
endfor

Meta-oracle F contains two oracles for each signature key pair: the first one allows to sign any string of bits. It has to store the result in mem_i^g as the adversary can only win if he finds a signature that has not been created by a signature oracle.

Oracle $SIGN_i(bs, \theta)$:
$out := \mathcal{S}(bs, sik_i\theta)$
$mem_i^g\theta := out :: mem_i^g\theta$
return out

The second oracle allows the adversary to access the verification key. It does not use its parameter bs.

Oracle $VK_i(bs, \theta)$:
return $vk_i\theta$

Meta-verifier V is composed of N verifiers, one for each key. Each verifier checks that the output of the adversary is a valid signature that has not been produced by the related signature oracle.

Verifier $V_i(bs, \theta)$:
if $bs \in mem_i^g$ **then**
return 0
else
return $\exists bs'.\mathcal{V}(bs', bs, vk\theta)$
endif

An adversary wins against N-UNF when he succeeds in producing a bit-string and its signature.

Definition 4.4 *A signature scheme $\mathcal{SS}$ is said N-UNF if the advantage of any adversary against criterion N-UNF is negligible and $PrRand^{UNF}$ is also negligible.*

When $N = 1$, N-UNF can be written UNF.

4.5.3 Symmetric Encryption: N-PAT-SYM-CPA

In some sense a symmetric encryption scheme includes both aspects, indistinguishability and authentication, which are present in asymmetric encryption and message signature respectively [BN00]. Therefore, our criterion for symmetric encryption is in a combination of IND-CPA and UNF. Indeed, a symmetric encryption should be secure in two ways. The first one is related to IND-CPA, any adversary should not be able to guess any information from messages encoded with an unknown key. The second one is related to UNF; any adversary should not be able to forge an encoding without knowing the key. Hence, oracles are similar to those presented in IND-CPA (except that no oracles output the public key), but there are two different ways to win the challenge.

Patterns

The challenge generation algorithm creates N symmetric keys using $\mathcal{KG}^s$. These keys are stored in θ under the names k_i for i ranging from 1 to N. This criterion cannot use asymmetric patterns as there are only symmetric keys in θ. Instead we define symmetric patterns using the following grammar:

$$\begin{array}{rcll} pat & ::= & \langle pat, pat\rangle \mid bs \mid [i] & \\ & & \mid \{pat\}^s_{bs} \mid \{pat\}^s_{[i]} & \text{symmetric encryption} \end{array}$$

For these patterns, the *concr* algorithm replaces $[i]$ by the value of k_i in θ. The *concr* algorithm is modified in order to handle symmetric patterns.

Algorithm $concr(pat, \theta)$:
  match pat **with**
    $bs \rightarrow$ **return** bs
    $[i] \rightarrow$ **return** $k_i\theta$
    $\langle p_1, p_2\rangle \rightarrow$ **return** $concr(p_1,\theta) \cdot concr(p_2,\theta)$
    $\{p\}^s_{[i]} \rightarrow$ **return** $\mathcal{E}^s(concr(p,\theta), k_i\theta)$
    $\{p\}^s_{bs} \rightarrow$ **return** $\mathcal{E}^s(concr(p,\theta), bs)$
  endmatch

The hypothesis of acyclicity regarding keys still holds: the encryption oracle related to key i works only on pair of symmetric patterns $\langle pat_0, pat_1\rangle$ such that for any j in $vars(\langle pat_0, pat_1\rangle)$, $i < j$.

Oracles and Criterion

The N-PAT-SYM-CPA criterion uses a symmetric encryption scheme $\mathcal{SE} = (\mathcal{KG}^s, \mathcal{E}^s, \mathcal{D}^s)$. It is defined by $(\Theta; F; V_{IND}, V_{UNF})$. Algorithm Θ generates N symmetric keys using $\mathcal{KG}^s$, a challenge bit b and a memory for each key used to store the outputs of the left-right oracle.

Algorithm $\Theta(\eta)$:
  $\theta := []$
  $b\theta \xleftarrow{R} [0,1]$
  for i **from** 1 **to** N
    $k_i\theta := \mathcal{KG}^s(\eta)$
    $mem^s_i\theta := []$
  endfor

The meta-oracle F only contains one oracle for each key: a left-right encryption oracle that takes as argument a pair of symmetric patterns $\langle pat_0, pat_1\rangle$ and outputs pat_b completed with the secret keys ($concr(pat_b, \theta)$) and encoded with k_i. As before, the acyclicity condition is checked and if it is not verified, 1 is returned. The oracle also checks that the two patterns are equivalent.

Oracle $ELR^s_i(\langle pat_0, pat_1\rangle, \theta)$:
  if $\exists j \in var(\langle pat_0, pat_1\rangle), j \leq i$ **or not** $pat_0 \approx pat_1$ **then**
    return 1
  else
    $bs := concr(pat_{b\theta}, \theta)$
    $out := \mathcal{E}^g(bs, k_i\theta)$
    $mem^s_i\theta := out :: mem^s_i\theta$
    return out
  endif

Finally, the verifier is composed of two parts: V_{IND} returns true when the adversary returns bit b.

Verifier $V_{IND}(bs, \theta)$:
  return $bs = b\theta$

The second part of the verifier V_{UNF} returns true when the adversary outputs a message encoded by one of the symmetric key and this message has not been produced by an encryption oracle. As in the case of digital signature, it is composed of N verifiers, one for each key.

Verifier $V_{UNF,i}(bs, \theta)$:
  if $bs \in mem_i^s$ **then**
    return 0
  else
    return $\mathcal{D}^g(bs, k_i\theta) \neq \perp$
  endif

Definition 4.5 *A symmetric encryption scheme $\mathcal{SE}$ is said N-PAT-SYM-CPA the advantage of any adversary against the criterion given above is negligible and if $PrRand^{\Theta;F;V_{UNF,i}}$ is negligible for any i.*

When $N = 1$, criterion 1-PAT-SYM-CPA is also denoted by SYM-CPA.

The criterion related to IND $(\Theta; F; V_{IND})$ (resp. to UNF $(\Theta; F; V_{UNF})$) is denoted by N-PAT-SYM-CPA/IND (resp. N-PAT-SYM-CPA/UNF).

Let N-PAT-SYM-CCA be the same criterion as N-PAT-SYM-CPA but where the adversary also have access to a decryption oracle (with the usual restrictions). Then this criterion is equivalent to N-PAT-SYM-CPA.

Proposition 4.21 *Let N be an integer and $\mathcal{SE}$ be a symmetric encryption scheme such that the PrRand linked to SYM-CPA is negligible. Then $\mathcal{SE}$ is N-PAT-SYM-CPA iff it is N-PAT-SYM-CCA.*

Proof: One of the two implication is immediate. Let $\mathcal{A}$ be an adversary against N-PAT-SYM-CPA. Oracles from N-PAT-SYM-CCA include the oracles from N-PAT-SYM-CPA. Hence adversary $\mathcal{A}$ can be used against N-PAT-SYM-CCA and we get that for any security parameter η,

$$\mathbf{Adv}_{\mathcal{A}}^{N-PAT-SYM-CCA}(\eta) = \mathbf{Adv}_{\mathcal{A}}^{N-PAT-SYM-CPA}(\eta)$$

Therefore, safety for N-PAT-SYM-CCA implies safety for N-PAT-SYM-CPA.

The reciprocal is less trivial. Let $\mathcal{A}$ be an adversary against N-PAT-SYM-CCA. We want to design an adversary $\mathcal{B}$ against N-PAT-SYM-CPA. The problem is that queries made by $\mathcal{A}$ to the decryption oracle cannot be simulated. This can be solved because queries made to the decryption oracle must be fresh encryptions and such encryptions may allow adversary $\mathcal{B}$ to win against the unforgeability part of the symmetric criterion.

Adversary $\mathcal{B}(\eta)/\mathcal{O}_1^{pk}, \mathcal{O}_1^{LR}, ..., \mathcal{O}_N^{pk}, \mathcal{O}_N^{LR}$:
  $bs := \mathcal{A}/\mathcal{O}_1^{pk}, \mathcal{O}_1^{LR}, \lambda q.$**return** q
  ...
  $\mathcal{O}_N^{pk}, \mathcal{O}_N^{LR}, \lambda q.$**return** q
  return bs

The adversary $\mathcal{B}$ represented here is simplified as $\mathcal{B}$ should check that queries to decryption oracles are fresh (i.e. not in mem) before returning them.

It is now possible to relate the advantage of $\mathcal{B}$ to the advantage of $\mathcal{A}$. Let $D(\mathcal{A})$ denote the event where $\mathcal{A}$ queries his decryption oracle with a fresh encryption, the related probability is denoted by p. In this case, adversary $\mathcal{B}$ wins against the UNF part of his criterion. Let γ and

γ' be respectively criterion N-PAT-SYM-CCA and N-PAT-SYM-CPA. Criterion γ_{IND} represents the indistinguishability part of γ whereas γ_{UNF} represents the unforgeability part.

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{\gamma_{UNF}}(\eta) &= 2\big(Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma_{UNF}}\big] - PrRand^{\gamma_{UNF}}\big)\\ &= 2\big(p.Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma_{UNF}}|D(\mathcal{A})\big] + (1-p).Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma_{UNF}}|\neg D(\mathcal{A})\big] - PrRand^{\gamma_{UNF}}\big)\end{aligned}$$

The same computation can be done for the indistinguishability part.

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{\gamma_{IND}}(\eta) &= 2Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma_{IND}}\big] - 1\\ &= 2p.Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma_{IND}}|D(\mathcal{A})\big] + 2(1-p).Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma_{IND}}|\neg D(\mathcal{A})\big] - 1\end{aligned}$$

When event $D(\mathcal{A})$ occurs, $\mathcal{B}$ always wins his challenge against UNF whereas when $D(\mathcal{A})$ does not occur, then $\mathcal{A}$ and $\mathcal{B}$ produce the exact same result. Therefore, the advantage of $\mathcal{B}$ against UNF is:

$$\mathbf{Adv}_{\mathcal{B}}^{\gamma'_{UNF}}(\eta) = 2\big(p + (1-p).Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma'_{UNF}}|\neg D(\mathcal{A})\big] - PrRand^{\gamma'_{UNF}}\big)$$

The two $PrRand$ for γ_{UNF} and γ'_{UNF} are the same and are negligible. Probability p is hence negligible and we have that:

$$\mathbf{Adv}_{\mathcal{B}}^{\gamma'_{UNF}}(\eta) \geq \mathbf{Adv}_{\mathcal{A}}^{\gamma_{UNF}}(\eta)$$

Hence the advantage of $\mathcal{A}$ against the UNF part is negligible. For the IND part, the advantage of $\mathcal{B}$ is given by:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\gamma'_{IND}}(\eta) &= 2Pr\big[\mathbf{G}_{\mathcal{B}}^{\gamma'_{IND}}\big] - 1\\ &= 2(1-p).Pr\big[\mathbf{G}_{\mathcal{A}}^{\gamma_{IND}}|\neg D(\mathcal{A})\big] - 1\end{aligned}$$

Hence we have the following inequality:

$$\mathbf{Adv}_{\mathcal{B}}^{\gamma'_{IND}}(\eta) + 2p \geq \mathbf{Adv}_{\mathcal{A}}^{\gamma_{IND}}(\eta)$$

The advantage of $\mathcal{B}$ and p are negligible so the advantage of $\mathcal{A}$ against IND is also negligible. We have that the advantage of $\mathcal{A}$ is negligible for any adversary $\mathcal{A}$, consequently the encryption scheme is safe for N-PAT-SYM-CCA. ■

Proposition 4.22 *Let N be an integer. If a symmetric encryption scheme $\mathcal{SE}$ is N-PAT-SYM-CPA then $\mathcal{SE}$ is SYM-CPA/IND and SYM-CPA/UNF.*

Proof: This proof is easy to perform by using the same argument as in the proof of proposition 4.13. ■

As previously, the converse of this proposition is also true but its proof is only provided later in this document.

Building a SYM-CPA Encryption Scheme

It is possible to build a SYM-CPA symmetric encryption scheme by using a digital signature scheme safe for UNF and an asymmetric encryption scheme safe for IND-CPA. This scheme is really inefficient as it uses public key cryptography to simulate symmetric key cryptography whereas in reality protocols use public key cryptography to safely exchange a symmetric key and after that use symmetric key cryptography as encryptions are much faster to compute in the symmetric case. However the existence of this algorithm allows us to prove that:

Proposition 4.23 *If there exist an asymmetric encryption scheme safe for IND-CPA and a digital signature scheme safe for UNF, then there exists a symmetric encryption scheme that is safe for SYM-CPA.*

Moreover, we later prove that N-PAT-SYM-CPA is equivalent to SYM-CPA for any integer N. This allows us to deduce the existence of a symmetric encryption scheme safe for N-PAT-SYM-CPA under the (reasonable) hypothesis that there exists an IND-CPA encryption scheme and an UNF signature scheme.

Let $(\mathcal{KG}^a, \mathcal{E}^a, \mathcal{D}^a)$ be an asymmetric encryption scheme that is safe for IND-CPA. Let $(\mathcal{KG}^g, \mathcal{S}^g, \mathcal{V}^g)$ be a digital signature scheme that is safe for UNF. Then the symmetric encryption scheme $(\mathcal{KG}^s, \mathcal{E}^s, \mathcal{D}^s)$ is build by combining these two primitives in the following way: the key generation algorithm uses $\mathcal{KG}^a$ and $\mathcal{KG}^g$ to generate a pair of asymmetric keys and a pair of signature keys. The symmetric key is constituted by the pairing of these two key pairs.

Algorithm $\mathcal{KG}^s(\eta)$:
$(pk, sk) := \mathcal{KG}^a(\eta)$
$(sik, vk) := \mathcal{KG}^g(\eta)$
return (pk, sk, sik, vk)

The encryption of a bit-string bs is obtained by concatenation of the encryption bs_1 of bs using the asymmetric encryption scheme and the signature of bs_1. The encryption provides the secrecy and the signature provides the authentication of the symmetric encryption.

Algorithm $\mathcal{E}^s(bs, (pk, sk, sik, vk))$:
$bs_1 := \mathcal{E}^a(bs, pk)$
$bs_2 := \mathcal{S}(bs_1, sik)$
return (bs_1, bs_2)

The decryption algorithm is designed in order to check that the signature is correct. In this case, it decrypts the message.

Algorithm $\mathcal{D}^s((bs_1, bs_2), (pk, sk, sik, vk))$:
if $\mathcal{V}(bs_1, bs_2, vk)$ **then return** $\mathcal{D}^a(bs_1, sk)$
else return $\perp$

The first thing one wants to verify on our algorithm is that for any bit-string bs and any symmetric key $k = (pk, sk, sik, vk)$ then $\mathcal{D}^s(\mathcal{E}^s(bs, k), k)$ returns bs. It is easy to see that this property is verified.

Now we want to check that our new symmetric encryption scheme is safe for SYM-CPA. Let $\mathcal{A}$ be an adversary against SYM-CPA. We want to prove that the advantage of $\mathcal{A}$ is negligible. This is done by building some adversaries against IND-CPA and UNF using $\mathcal{A}$. We first build an adversary $\mathcal{B}$ against IND-CPA (for the asymmetric encryption scheme) that uses $\mathcal{A}$ such that:

$$\mathbf{Adv}(\eta)_{\mathcal{A}}^{(\Theta;F;V_{IND})} = \mathbf{Adv}(\eta)_{\mathcal{B}}^{IND-CPA}$$

As the asymmetric encryption scheme used here is supposed safe for IND-CPA, the advantage of $\mathcal{B}$ is negligible and so the advantage of $\mathcal{A}$ against the indistinguishability part is also negligible. Let us make precise the construction of adversary $\mathcal{B}$. This adversary has access to two oracles, $\mathcal{O}_1$ is the left-right encryption oracle and $\mathcal{O}_2$ is the public key oracle.

Adversary $\mathcal{B}(\eta)/\mathcal{O}_1, \mathcal{O}_2$:
$(sik, vk) := \mathcal{KG}^g(\eta)$
$out := \mathcal{A}(\eta)/\lambda(bs_0, bs_1).bs := \mathcal{O}_1(bs_b);$
return $(bs, \mathcal{S}^g(bs, sik))$
return out

The experiments involving $\mathcal{A}$ against SYM-CPA/IND and $\mathcal{B}$ against IND-CPA are equivalent and $PrRand$ is the same in both criteria. Thus the advantages of $\mathcal{A}$ and $\mathcal{B}$ are equal.

To deal with the UNF part of the symmetric criterion, we introduce an adversary $\mathcal{C}$ against UNF (for the signature scheme) that also uses $\mathcal{A}$ such that:

$$\mathbf{Adv}(\eta)_{\mathcal{A}}^{(\Theta;F;V_{UNF})} = \mathbf{Adv}(\eta)_{\mathcal{C}}^{UNF} + f(\eta)$$

Where f is a negligible function. As the digital signature scheme used here is supposed safe for UNF, the advantage of $\mathcal{C}$ is negligible and so the advantage of $\mathcal{A}$ against the unforgeable part is also negligible. Let us describe the construction of adversary $\mathcal{C}$. This adversary can query two oracles, $\mathcal{O}_1$ is the signing oracle and $\mathcal{O}_2$ is the verification key oracle.

Adversary $\mathcal{C}(\eta)/\mathcal{O}_1, \mathcal{O}_2$:
$(pk, sk) := \mathcal{KG}^a(\eta)$
$b \xleftarrow{R} [0, 1]$
$out := \mathcal{A}(\eta)/\lambda(bs_0, bs_1).bs := \mathcal{E}^a(bs_b, pk);$
return $(bs, \mathcal{O}_1(bs))$
return *out*

The experiments involving $\mathcal{A}$ against SYM-CPA/UNF and $\mathcal{C}$ against UNF are equivalent. Furthermore, $PrRand^{SYM-CPA/UNF} \leq PrRand^{UNF}$ thus as $PrRand^{UNF}$ is negligible, $PrRand^{SYM-CPA/UNF}$ is also negligible. Thus the advantages of $\mathcal{A}$ and $\mathcal{C}$ are equal up to a negligible function.

As the advantages of $\mathcal{A}$ against the indistinguishability part and the unforgeable part are negligible, the advantage of $\mathcal{A}$ against SYM-CPA is negligible. Our new encryption scheme is safe for SYM-CPA.

4.5.4 Cryptographic Library: N-PASS

Criterion N-PASS is a mix of previous criteria: N-PAT-IND-CCA, N-UNF and N-SYM-CPA. It is used to verify safety of a whole cryptographic library that contains an asymmetric encryption scheme, a symmetric encryption scheme and a digital signature scheme. As this criterion combines the three previous ones, N signature, symmetric and asymmetric keys are generated as long as a single challenge bit b. The adversary can access oracles he was granted in the previous criteria (left-right encryption, public key and decryption for the asymmetric scheme) and can win either by deducing the value of b, by forging a fresh encryption with a challenge symmetric key or by forging a fresh signature using a challenge signature key.

Patterns

In this section, we define general patterns, general patterns are an extension of asymmetric patterns and symmetric patterns. Authorized operations are asymmetric encryption, symmetric encryption, digital signature and concatenation. There are also three different types of pattern variables: $[i]^a$ asks for the inclusion of the i^{th} secret key, $[i]^s$ for the inclusion of the i^{th} symmetric key and $[i]^g$ for the i^{th} verification key.

$$
\begin{array}{rcll}
pat & ::= & \langle pat, pat\rangle \mid bs & \\
 & & \mid [i]^a \mid [i]^s \mid [i]^g & \\
 & & \mid \{pat\}^a_{bs} \mid \{pat\}^a_{[i]} & \text{asymmetric encryption} \\
 & & \mid \{pat\}^s_{bs} \mid \{pat\}^s_{[i]} & \text{symmetric encryption} \\
 & & \mid \{pat\}^g_{bs} \mid \{pat\}^g_{[i]} & \text{digital signature}
\end{array}
$$

The *concr* algorithm is once more extended in order to handle general patterns. Note that this *concr* algorithm could have been used for asymmetric patterns and symmetric patterns if we rename pattern variables ($[i]^a \rightarrow [i]$ and $[i]^s \rightarrow [i]$ respectively).

Algorithm $concr(pat, \theta)$:
match *pat* **with**
$bs \rightarrow$ **return** bs
$[i]^a \rightarrow$ **return** $sk_i\theta$

$[i]^s \rightarrow$ **return** $k_i\theta$
$[i]^g \rightarrow$ **return** $sik_i\theta$
$\langle p_1, p_2\rangle \rightarrow$ **return** $concr(p_1,\theta)\cdot concr(p_2,\theta)$
$\{p\}^a_{[i]} \rightarrow$ **return** $\mathcal{E}^a(concr(p,\theta), pk_i\theta)$
$\{p\}^s_{[i]} \rightarrow$ **return** $\mathcal{E}^s(concr(p,\theta), k_i\theta)$
$\{p\}^g_{[i]} \rightarrow$ **return** $\mathcal{S}(concr(p,\theta), sik_i\theta)$
$\{p\}^a_{bs} \rightarrow$ **return** $\mathcal{E}^a(concr(p,\theta), bs)$
$\{p\}^s_{bs} \rightarrow$ **return** $\mathcal{E}^s(concr(p,\theta), bs)$
$\{p\}^g_{bs} \rightarrow$ **return** $\mathcal{S}(concr(p,\theta), bs)$
endmatch

The acyclicity condition still holds on both primitives. However, we authorize patterns using symmetric keys and signature keys when accessing left-right oracles from the asymmetric part. Hence signatures, signature keys, symmetric encryptions and symmetric keys can be used under asymmetric encryptions but the converse is forbidden. We also authorize signature and signature keys to appear under symmetric encryptions.

Oracles and Criterion

Criterion N-PASS applies on a cryptographic library $\mathcal{CL}$ which has to contain an asymmetric encryption scheme $(\mathcal{KG}^a, \mathcal{E}^a, \mathcal{D}^a)$, a symmetric encryption scheme $(\mathcal{KG}^s, \mathcal{E}^s, \mathcal{D}^s)$ and a digital signature scheme $(\mathcal{KG}^g, \mathcal{S}^g, \mathcal{V}^g)$. The criterion is defined as $(\Theta; F; V)$. Algorithm Θ generates a challenge bit b, N pairs of asymmetric keys using $\mathcal{KG}^a$ which are stored in θ linked to names (pk_i, sk_i), N symmetric keys using $\mathcal{KG}^s$ which names are k_i and N pairs of signature keys which names are (sik_i, vk_i). Memory is also created for every key using names mem^a_i, mem^s_i and mem^g_i.

Algorithm $\Theta(\eta)$:
$\theta := []$
$b\theta \xleftarrow{R} [0,1]$
for i **from** 1 **to** N
$(pk_i\theta, sk_i\theta) := \mathcal{KG}^a(\eta)$
$mem^a_i\theta := []$
$k_i\theta := \mathcal{KG}^s(\eta)$
$mem^s_i\theta := []$
$(sik_i\theta, vk_i\theta) := \mathcal{KG}^g(\eta)$
$mem^g_i\theta := []$
endfor

Meta-oracle F contains all the oracles described before. Let us recall informally what they do:

1. ELR^a_i is the left-right encryption oracle for asymmetric encryption.
2. D_i is the decryption oracle for asymmetric encryption.
3. PK_i returns the bit-string value of public key pk_i.
4. $SIGN_i$ is the signature oracle.
5. VK_i returns the bit-string value of public verification key vk_i.
6. ELR^s_i is the left-right encryption oracle for symmetric encryption.

Multiple verifiers are used to check the answer of the adversary. These verifiers are the same as the one given above:

1. V_{IND} checks that the adversary deduced the bit b.

2. $V^g_{UNF,i}$ checks that the output of the adversary is a valid signature using vk_i that has not been produced by $SIGN_i$.

3. $V^s_{UNF,i}$ checks that the output is a valid symmetric encryption using k_i that has not been produced by ELR^s_i

A cryptographic library is safe for N-PASS if the advantage of any adversary against the criterion where the cryptographic primitives are implemented using $\mathcal{CL}$, is negligible.

Proposition 4.24 *Let $\mathcal{CL}$ be a cryptographic library that contains an asymmetric encryption scheme $\mathcal{AE}$, a symmetric encryption scheme $\mathcal{SE}$ and a digital signature scheme $\mathcal{SS}$. If $\mathcal{CL}$ is safe for N-PASS, then $\mathcal{AE}$ is safe for IND-CCA, $\mathcal{SE}$ is safe for SYM-CPA and $\mathcal{SS}$ is safe for UNF.*

Proof: This proof is close to previously done ones. If we consider an adversary $\mathcal{A}$ against one of the sub-criteria, IND-CCA for example, then $\mathcal{A}$ can be used against N-PASS with the same advantage (provided that N is greater or equal to 1). Hence this safety implication is immediate. ■

The challenge bit b is common to symmetric and asymmetric encryptions, thus it is non trivial to prove that IND-CCA, UNF and SYM-CPA are equivalent to N-PASS-CCA. For this reason, we introduce criterion partition theorems which prove joint security using security of each of the primitives.

Chapter 5

Equivalence of Criteria

Contents

In this chapter, we present the criterion partition theorem. This theorem allows one to prove safety of a criterion γ under the condition that a sub-criterion of γ and an indistinguishability criterion derived from γ are safe. As the proof of this theorem is quite complicated, we first illustrate how things work on a simple example. This example consists in proving the equivalence between the 2-PAT-IND-CPA criterion and IND-CPA. Then two different versions of the criterion partition theorem are presented. Application of these theorems is afterwards illustrated on the security criteria that we introduced in the previous chapter.

5.1 Example of Partition

In this section, we detail the application of the partition theorem on a simple but non-trivial example. As we are interested in checking safety of an asymmetric encryption scheme, let $\mathcal{AE}$ be such an encryption scheme composed by the key generation algorithm $\mathcal{KG}$, the encryption algorithm $\mathcal{E}$ and the decryption algorithm $\mathcal{D}$.

5.1.1 The 2-KDM-IND-CPA Criterion

The 2-KDM-IND-CPA criterion is an extension of IND-CPA using key dependent messages as presented in section 4.5.1. In this criterion, two keys are generated along with the challenge bit b. These keys are denoted by (pk_1, sk_1) and (pk_2, sk_2). The challenge generator Θ is defined by:

Algorithm $\Theta(\eta)$:
$\theta := []$
$b\theta \stackrel{R}{\leftarrow} [0,1]$
$(pk_1\theta, sk_1\theta) := \mathcal{KG}(\eta)$
$(pk_2\theta, sk_2\theta) := \mathcal{KG}(\eta)$

The adversary has to deduce the value of b. For this purpose, he has access to two oracles for each key pair: a public key oracle and a left-right encryption oracle. The particularity of this criterion is that the left-right encryption oracle takes as argument a pair of functions (f_0, f_1) instead of a pair of bit-string. These functions are represented by two PRTM.

Oracle $ELR_i((f_0, f_1), \theta)$:
$bs := f_{b\theta}(pk_1\theta, sk_1\theta, pk_2\theta, sk_2\theta)$
return $\mathcal{E}^a(bs, pk_i\theta)$

For the sake of simplicity, this oracle does not explicitly verify that f_1 and f_2 return bit-strings of equal length. We have an acyclicity hypothesis: ELR_i can only use $pk_j\theta$ and $sk_j\theta$ if $i < j$. Thus, arguments of ELR_2 cannot use any $pk_i\theta$ or $sk_i\theta$ and arguments of ELR_1 can only use $pk_2\theta$ and $sk_2\theta$. With this restriction, oracle ELR_1 and ELR_2 can be detailed.

Oracle $ELR_1((f_0, f_1), \theta)$:
$bs := f_{b\theta}(pk_2\theta, sk_2\theta)$
return $\mathcal{E}^a(bs, pk_1\theta)$

Oracle $ELR_2((f_0, f_1), \theta)$:
$bs := f_{b\theta}()$
return $\mathcal{E}^a(bs, pk_2\theta)$

Oracles giving access to the public keys are implemented by:

Oracle $PK_i(bs, \theta)$:
return $pk_i\theta$

The verifier V only has to check that the adversary correctly guessed the value of bit b. Formally, we have as usual that $V(bs, \theta) =$ **return** $(bs = b\theta)$. Criterion 2-KDM-IND-CPA is denoted by $\gamma = (\Theta; PK_1, PK_2, ELR_1, ELR_2; V)$. It is clear that it is possible from an adversary against IND-CPA to build an adversary against criterion γ, this justifies the following proposition.

Proposition 5.1 *If an asymmetric encryption scheme $\mathcal{AE}$ is safe for 2-KDM-IND-CPA then $\mathcal{AE}$ is safe for IND-CPA.*

Proof: Let $\mathcal{A}$ be an adversary against IND-CPA. Adversary $\mathcal{B}$ is built from $\mathcal{A}$ by using the first oracle of $\mathcal{B}$ (denoted by $\mathcal{O}_1$ which returns $pk_1\theta$) and the third oracle of $\mathcal{B}$ (corresponding to ELR_1). The only slight problem is that $\mathcal{A}$ submits bit-string to its left-right oracle whereas $\mathcal{B}$ has to submit functions. Hence $\lambda().bs$ denotes the function that takes no arguments and returns bs.

Adversary $\mathcal{B}(\eta)/\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4$:
$bs := \mathcal{A}(\eta)/\mathcal{O}_1,$
$\lambda\langle bs_0, bs_1\rangle.\mathcal{O}_3((\lambda().bs_0, \lambda().bs_1))$
return bs

Let γ' denote criterion IND-CPA. The game involving $\mathcal{A}$ against γ' and $\mathcal{B}$ against γ are equivalent. Hence,

$$Pr[\mathbf{G}_{\mathcal{A}}^{\gamma'} = true] = Pr[\mathbf{G}_{\mathcal{B}}^{\gamma} = true]$$

Moreover both criteria consist in guessing the value of a randomly generated bit b, so the two $PrRand$ are equal to 1/2. Thus, we get that the advantage of $\mathcal{A}$ against γ' is equal to the advantage of $\mathcal{B}$ against γ. If $\mathcal{AE}$ is safe for 2-KDM-IND-CPA, then the advantage of $\mathcal{B}$ against γ is p-negligible and so is the advantage of $\mathcal{A}$ against γ'. Therefore $\mathcal{AE}$ is also safe for IND-CPA. ■

Our objective in this section is to prove the converse of this proposition.

5.1.2 Reducing to IND-CPA

In this subsection, we give a proof of the following result:

Proposition 5.2 *If an asymmetric encryption scheme $\mathcal{AE}$ is safe for IND-CPA then $\mathcal{AE}$ is safe for 2-KDM-IND-CPA.*

The proof of this result is non-trivial. We use a proof technique that is generalized under the name "partition theorem" in the rest of this chapter. Let $\mathcal{A}$ be an adversary against γ (i.e. 2-KDM-IND-CPA). The usual method to prove this kind of implication is to build an adversary $\mathcal{A}'$ against IND-CPA whose advantage would bound the advantage of $\mathcal{A}$. Hence, if the encryption scheme is safe for IND-CPA, the advantage of $\mathcal{A}'$ is p-negligible and so is the advantage of $\mathcal{A}$. As this is true for any adversary $\mathcal{A}$, it is possible to conclude that the encryption scheme is safe for γ. However, in our situation designing the adversary $\mathcal{A}'$ is a difficult task. A simple idea is to use the challenge key of $\mathcal{A}'$ for key pair (pk_2, sk_2) and to generate two other key pairs inside $\mathcal{A}'$ in order to answer queries that $\mathcal{A}$ makes to oracle ELR_2. A bit b' is also generated in order to answer such queries as the challenge bit b is (of course) not known to $\mathcal{A}'$. Adversary $\mathcal{A}'$ is formally described by:

Adversary $\mathcal{A}'(\eta)/\mathcal{O}_1, \mathcal{O}_2$:
$b' \xleftarrow{R} [0,1]$
$(pk'_2, sk'_2) := \mathcal{KG}(\eta)$
$(pk_1, sk_1) := \mathcal{KG}(\eta)$
$bs := \mathcal{A}(\eta)/\lambda bs.pk_1,$
$\mathcal{O}_1,$
$\lambda(f_0, f_1).\mathcal{E}(f_{b'}(pk'_2, sk'_2), pk_1)$
$\lambda(f_0, f_1).\mathcal{O}_2(f_0(), f_1()),$
return bs

Where $\mathcal{O}_1$ is the public-key oracle and $\mathcal{O}_2$ is the left-right encrytion oracle. Adversary $\mathcal{A}'$ simulates $\mathcal{A}$ confronted to *decorrelated* oracles, queries made by $\mathcal{A}$ to his first left-right oracle are not answered using the right bit b and the right key pair (pk_2, sk_2). The advantage of $\mathcal{A}'$ cannot be compared directly to the advantage of $\mathcal{A}$. For this reason, we introduce a new adversary $\mathcal{B}$ against criterion IND-CPA. The intuition is that whenever the challenge bit b'' of adversary $\mathcal{B}$ is equal to 1, the game involving $\mathcal{B}$ is equivalent to the game involving $\mathcal{A}$ against γ whereas when b'' equals 0, the game involving $\mathcal{B}$ is equivalent to the game involving $\mathcal{A}'$ against IND-CPA. Adversary $\mathcal{B}$ generates two challenge bits b and b' and two key pairs (pk_2, sk_2) and (pk'_2, sk'_2). In order to answer queries made by $\mathcal{A}$ to oracles related to the first key pair, $\mathcal{B}$ uses bit b and key pair (pk_2, sk_2). For queries made by $\mathcal{A}$ to oracles related to the second key pair, $\mathcal{B}$ forwards queries for the public key to its public key oracle, queries (f_0, f_1) to the left-right encryption oracle are converted to queries of the form $\langle f_{b'}(pk'_2, sk'_2), f_b(pk_2, sk_2)\rangle$. Thus according to the value of its challenge bit b'', the game involving $\mathcal{B}$ simulates either the game involving $\mathcal{A}'$ or the game involving $\mathcal{A}$. Finally, if $\mathcal{A}$ guesses the value of bit b, $\mathcal{B}$ answers 1 as $\mathcal{A}$ has better chances to win with non decorrelated oracles, whereas if $\mathcal{A}$ fails, $\mathcal{B}$ assumes that $\mathcal{A}$ was confronted to decorrelated oracles and returns 0.

Adversary $\mathcal{B}(\eta)/\mathcal{O}_1, \mathcal{O}_2$:
$b \xleftarrow{R} [0,1]$
$b' \xleftarrow{R} [0,1]$
$(pk_2, sk_2) := \mathcal{KG}(\eta)$

$(pk'_2, sk'_2) := \mathcal{KG}(\eta)$
$bs := \mathcal{A}(\eta)/\mathcal{O}_1,$
$\lambda bs.pk_2,$
$\lambda(f_0, f_1).\mathcal{O}_2(\langle f_{b'}(pk'_2, sk'_2), f_b(pk_2, sk_2)\rangle)$
$\lambda(f_0, f_1).\mathcal{E}(f_b(), pk_2),$
return $bs = b$

Let γ' denote criterion IND-CPA. The game involving $\mathcal{B}$ in the case where its challenge bit is equal to i is denoted by $\mathbf{G}^{\gamma'}_{\mathcal{B}}|b''\theta = i$ (by abuse of notation, the challenge bit of $\mathcal{B}$ in γ' is denoted by b'' instead of b, the challenge key pair is denoted by (pk_1, sk_1)). Then we describe the different games to put the focus on their equivalence.

Game $\mathbf{G}^{\gamma'}_{\mathcal{B}}(\eta)|b''\theta = 1$:
$(pk_1, sk_1) := \mathcal{KG}(\eta)$
$b \xleftarrow{R} [0, 1]$
$b' \xleftarrow{R} [0, 1]$
$(pk_2, sk_2) := \mathcal{KG}(\eta)$
$(pk'_2, sk'_2) := \mathcal{KG}(\eta)$
$bs := \mathcal{A}(\eta)/\lambda bs.pk_1,$
$\lambda bs.pk_2,$
$\lambda(f_0, f_1).\mathcal{E}(f_b(pk_2, sk_2), pk_1),$
$\lambda(f_0, f_1).\mathcal{E}(f_b(), pk_2)$
return $bs = b$

Game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$:
$b \xleftarrow{R} [0, 1]$
$(pk_1, sk_1) := \mathcal{KG}(\eta)$
$(pk_2, sk_2) := \mathcal{KG}(\eta)$
$bs{:=}\mathcal{A}(\eta)/\lambda bs.pk_1,$
$\lambda bs.pk_2,$
$\lambda(f_0, f_1).\mathcal{E}(f_b(pk_2, sk_2), pk_1),$
$\lambda(f_0, f_1).\mathcal{E}(f_b(), pk_2)$
return $bs = b$

Thus as the two games presented here are equivalent, we get that:

$$Pr[\mathbf{G}^{\gamma'}_{\mathcal{B}}(\eta) = true|b''\theta = 1] = Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true]$$

In a very similar way, when considering the case $b'' = 0$, it is easy to obtain that:

$$Pr[\mathbf{G}^{\gamma'}_{\mathcal{B}}(\eta) = true|b''\theta = 0] = Pr[\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta) = true]$$

Thus, by applying proposition 4.2 the advantage of $\mathcal{B}$ is given by:

$$\mathbf{Adv}^{\gamma'}_{\mathcal{B}}(\eta) = Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - Pr[\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta) = true]$$

Criteria γ and γ' are both indistinguishability criteria, hence $PrRand^{\gamma}$ and $PrRand^{\gamma'}$ are both equal to $1/2$. Thus, we get that:

$$\begin{aligned}\mathbf{Adv}^{\gamma'}_{\mathcal{B}}(\eta) \quad = \quad & Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - PrRand^{\gamma} \\ & -\big(Pr[\mathbf{G}^{\gamma'}_{\mathcal{A}'}(\eta) = true] - PrRand^{\gamma'}\big)\end{aligned}$$

The advantages of the three adversaries $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ can now be related by:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 2\mathbf{Adv}^{\gamma'}_{\mathcal{B}}(\eta) + \mathbf{Adv}^{\gamma'}_{\mathcal{A}'}(\eta)$$

This relation allows us to conclude. If $\mathcal{AE}$ is safe for IND-CPA, then the advantages of $\mathcal{B}$ and $\mathcal{A}'$ are p-negligible. So the advantage of $\mathcal{A}$ is also p-negligible. Thus, $\mathcal{AE}$ is safe for 2-KDM-PAT-IND-CPA.

5.2 Simplified Partition Theorem

Consider a criterion $\gamma = (\Theta_1, \Theta_2; F_1, F_2; V_1)$, composed of two challenge generators Θ_1 and Θ_2, two oracles F_i, and a verifier V_1 (which can represent multiple verifiers). Assume that oracle F_1 and verifier V_1 do not depend on θ_2. Because of these assumptions, it is possible to define with a slight abuse of notation a criterion $\gamma_1 = (\Theta_1; F_1; V_1)$. We are going to relate the advantages against γ and γ_1. To do so, let us consider the game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$ of an adversary $\mathcal{A}$ against γ:

Game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$:
$\theta_1 := \Theta_1(\eta)$
$\theta_2 := \Theta_2(\eta)$
$bs := \mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1),$
$\lambda s.F_2(s,\theta_1,\theta_2)$
return $V_1(bs,\theta_1)$

Using $\mathcal{A}$ as a sub-routine, we build an adversary $\mathcal{A}'$ that randomly generates θ'_1 and θ_2 in order to answer queries made by $\mathcal{A}$ to his second oracle:

Adversary $\mathcal{A}'(\eta)/\mathcal{O}_1$:
$\theta'_1 := \Theta_1(\eta)$
$\theta_2 := \Theta_2(\eta)$
$bs := \mathcal{A}(\eta)/\mathcal{O}_1,$
$\lambda s.F_2(s,\theta'_1,\theta_2)$
return bs

Adversary $\mathcal{A}'$ plays against γ_1. He tries to imitate the behavior of $\mathcal{A}$ except that $\mathcal{A}'$ computes himself the answer of queries to the second oracle of $\mathcal{A}$ (and most probably computes wrong answers). Therefore, we can consider the game $\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta)$ given by:

Game $\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta)$:
$\theta_1 := \Theta_1(\eta)$
$\theta'_1 := \Theta_1(\eta)$
$\theta_2 := \Theta_2(\eta)$
$s := \mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1),$
$\lambda s.F_2(s,\theta'_1,\theta_2)$
return $V_1(s,\theta_1)$

In this game, $\mathcal{A}$ is confronted to decorrelated oracles. When applying the theorem, one wants to bound the advantage of $\mathcal{A}$ by a function using the advantage of $\mathcal{A}'$. For this purpose, our aim is to provide a bound on:

$$Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - Pr[\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta) = true]$$

To do so, we construct an adversary $\mathcal{B}$ that tries to distinguish game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$ and game $\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta)$, i.e. $\mathcal{B}$ tries to distinguish the case where $\mathcal{A}$ uses correlated oracles (i.e. the same θ_1 and θ_2 are used by F_1 and F_2, $F_1(.,\theta_1)$ and $F_2(.,\theta_1,\theta_2)$) from the case where $\mathcal{A}$ uses decorrelated oracles (i.e. θ_1 is used by F_1 and a different θ'_1 with θ_2 is used by F_2, $F_1(.,\theta_1)$ and $F_2(.,\theta'_1,\theta_2)$ see figure 5.1). That is, we would like to define a new indistinguishability criterion γ_2 that uses a challenge bit b and a distinguisher $\mathcal{B}$ such that the following equations hold:

$$Pr[\mathbf{G}^{\gamma_2}_{\mathcal{B}} = true \mid b\theta = 1] = Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] \tag{5.1}$$

$$Pr[\mathbf{G}^{\gamma_2}_{\mathcal{B}} = true \mid b\theta = 0] = Pr[\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta) = true] \tag{5.2}$$

Indeed, these equations allow us to derive the following bound:

$$Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - Pr[\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta) = true] = \mathbf{Adv}^{\gamma_2}_{\mathcal{B}}$$

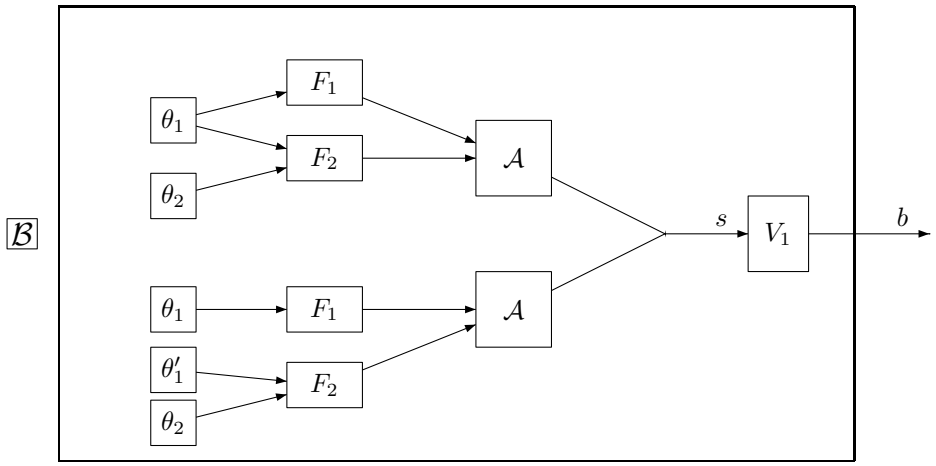


Figure 5.1: Correlated and Decorrelated Oracles

Construction of the Distinguisher In the following, we give a method that tells us how to construct the indistinguishability criterion γ_2 and the adversary $\mathcal{B}$. To do so, we need the following assumption: there exist three probabilistic random functions (implementable using PRTM) f, g and f' such that oracle F_2's implementation consists of two parts: $\lambda s.f(g(s,\theta_1),\theta_2)$ and $\lambda s.f'(s,\theta_2)$. The first part depends on both θ_1 and θ_2 whereas the second depends only on θ_2.

The idea when introducing two parts for oracle F_2 is to separate the oracles contained in F_2 that depend really on both θ_1 and θ_2 (these oracles are placed in $f(g(...))$) from the oracles that do not depend on θ_1 (placed in f'). Given F_2, there are multiple possible choices for f, g and f' (for example, a choice that is always possible is to take an empty f', $g(s,\theta_1)$ returns both s and θ_1 and f is F_2). However, different choices lead to different partitions and some choices can lead to non-interesting partitions.

Adversary $\mathcal{B}$ plays against an indistinguishability criterion. He has access to two oracles: $\hat{\mathcal{O}}_1$ is implemented by the left-right oracle $f \circ LR^b$ defined by $f \circ LR^b(\langle m_0, m_1\rangle, \theta_2) = f(m_b, \theta_2)$ and $\hat{\mathcal{O}}_2$ is simply implemented by f'. Notice now that we have the following equations:

$$\begin{array}{rcl} f \circ LR^b(\langle g(s,\theta'_1), g(s,\theta_1)\rangle) & = & f(g(s,\theta_1),\theta_2), \text{ if } b=1 \\ f \circ LR^b(\langle g(s,\theta'_1), g(s,\theta_1)\rangle) & = & f(g(s,\theta'_1),\theta_2), \text{ if } b=0 \end{array}$$

The first line corresponds to the f part of $F_2(s,\theta_1,\theta_2)$ whereas the second corresponds to $F_2(s,\theta'_1,\theta_2)$. More formally, our γ_2 criterion is given by $\gamma_2 = (b, \Theta_2; f \circ LR^b, f'; v_b)$, where v_b just checks that the bit returned by the adversary equals b, i.e. $v_b(bs,\theta) = \textbf{return } (bs = b\theta)$.

We are now ready to give a distinguisher $\mathcal{B}$ such that equation (5.1) and equation (5.2) hold:

Adversary $\mathcal{B}(\eta)/\hat{\mathcal{O}}_1, \hat{\mathcal{O}}_2$:
θ_1:=$\Theta_1(\eta)$
θ'_1:=$\Theta_1(\eta)$
s:=$\mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1)$, // oracle F_1
$\lambda s.\hat{\mathcal{O}}_1(\langle g(s,\theta'_1), g(s,\theta_1)\rangle)$, // part f of oracle F_2
$\hat{\mathcal{O}}_2$ // part f' of oracle F_2
$\hat{b} := V_1(s,\theta_1)$
return $\hat{b}$

Thus, $\mathcal{B}$ uses $\mathcal{A}$ as a sub-routine. Recall that $\mathcal{A}$ may query three oracles: F_1, part f of oracle F_2 and part f' of oracle F_2 while $\mathcal{B}$ may query the left-right oracle $f \circ LR^b$ and f'. Therefore, $\mathcal{B}$ uses

Θ_1 to generate θ_1 and θ'_1. It is important to notice that θ_1 and θ'_1 are generated independently. Then, $\mathcal{B}$ uses $\mathcal{A}$ as a sub-routine using $\lambda s.F_1(s,\theta)$ for $\mathcal{A}$'s first oracle, and the two functions $\lambda s.\hat{\mathcal{O}}_1(\langle g(s,\theta'_1), g(s,\theta_1)\rangle)$ and f' for F_2.

We can now describe the game corresponding to $\mathcal{B}$ and γ_2:

Game $\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta)$:
$b \xleftarrow{R} [0,1]$
$\theta_2 := \Theta_2(\eta)$
$\hat{b} := \mathcal{B}(\eta)/\lambda s.f(LR^b(s),\theta_2),$
$\lambda s.f'(s,\theta_2)$
return $v_b(\hat{b})$

Comparing the Games Let us now check that equations (5.1) and (5.2) are verified. To do so, assume first that $b = 1$. Then, Game $\mathbf{G}^{\gamma_2}_{\mathcal{B}}|b = 1$ is obtained by replacing the definition of $\mathcal{B}$ within the game:

Game $\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta)|b = 1$:
$\theta_2 := \Theta_2(\eta)$
$\theta_1 := \Theta_1(\eta)$
$\theta'_1 := \Theta_1(\eta)$
$s := \mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1)$
$\lambda s.f(g(s,\theta_1),\theta_2),$
$\lambda s.f'(s,\theta_2)$
$\hat{b} := V_1(s,\theta_1)$
return $v_1(\hat{b})$

After the hypothesis we made about the decomposition of oracle F_2, and when detailing $\mathcal{B}$, this game can be rewritten as follows, and rigorously compared to the game played by adversary $\mathcal{A}$ against criterion γ:

Game $\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta)|b\theta = 1$:
$\theta_1 := \Theta_1(\eta)$
$\theta'_1 := \Theta_1(\eta)$
$\theta_2 := \Theta_2(\eta)$
$s := \mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1),$
$\lambda s.F_2(s,\theta_1,\theta_2)$
$\hat{b} := V_1(s,\theta_1)$
return $v_1(\hat{b})$

Game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$:
$\theta_1 := \Theta_1(\eta)$
$\theta_2 := \Theta_2(\eta)$
$s := \mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1),$
$\lambda s.F_2(s,\theta_1,\theta_2)$
return $V_1(s,\theta_1)$

The difference between these games lies in the generation of challenge θ'_1. However, this can be taken into account in the calculus of B's probability to win, i.e. to output 1. Let $p = Pr[\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta) = true|b\theta = 1]$.

$$\begin{aligned} p &= Pr[\theta_1 := \Theta_1, \theta'_1 := \Theta_1, \theta_2 := \Theta_2, s := A/F_1, F_2 : V_1(s,\theta_1)] \\ &= \sum_{\tilde{\theta}_1 := \Theta_1} Pr[\theta_1 := \Theta_1, \theta'_1 := \Theta_1, \theta_2 := \Theta_2, s := A/F_1, F_2 : V_1(s,\theta_1)|\theta'_1 = \tilde{\theta}_1] \\ &\qquad .Pr[\tilde{\theta}_1 := \Theta_1] \\ &= \sum_{\tilde{\theta}_1 := \Theta_1} Pr[\theta_1 := \Theta_1, \theta_2 := \Theta_2, s := A/F_1, F_2 : V_1(s,\theta_1)].Pr[\tilde{\theta}_1 := \Theta_1] \end{aligned}$$

since the execution of $\mathcal{A}$ does not depend on θ'_1.

$$\begin{aligned} p &= Pr[\theta_1 := \Theta_1, \theta_2 := \Theta_2, s := A/F_1, F_2 : V_1(s,\theta_1)].1 \\ &= Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] \end{aligned}$$

We now compare $\mathcal{B}$'s game when $b = 0$ and $\mathcal{A}'$'s, after having detailed $\mathcal{B}$'s behavior for easier comparison.

Game $\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta)|b\theta = 0$:
$\theta_2 := \Theta_2(\eta)$
$\theta_1 := \Theta_1(\eta)$
$\theta'_1 := \Theta_1(\eta)$
$s := \mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1),$
$\lambda s.F_2(s,\theta'_1,\theta_2)$
$\hat{b} := V_1(s,\theta_1)$
return $v_0(\hat{b})$

Game $\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta)$:
$\theta_1 := \Theta_1(\eta)$
$\theta'_1 := \Theta_1(\eta)$
$\theta_2 := \Theta_2(\eta)$
$s := \mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1),$
$\lambda s.F_2(s,\theta'_1,\theta_2)$
return $V_1(s,\theta_1)$

It is easy to see that, this time, the games played are the same. Adversary $\mathcal{B}$ wins anytime $\mathcal{A}'$ loses, and thus:

$$Pr[\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta) = false] = Pr[\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta) = true|b\theta = 0]$$

We can therefore compute our distinguisher's advantage :

$$\begin{aligned}
\mathbf{Adv}^{\gamma_2}_{\mathcal{B}}(\eta) &= 2\big(Pr[\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta) = true] - PrRand^{\gamma_2}\big) \\
&= 2Pr[\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta) = true|b\theta = 1]Pr[b = 1] + \\
&\quad 2Pr[\mathbf{G}^{\gamma_2}_{\mathcal{B}}(\eta) = true|b\theta = 0]Pr[b = 0] - 2PrRand^{\gamma_2} \\
&= Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] + Pr[\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta) = false] - 1
\end{aligned}$$

since $PrRand^{\gamma_2} = \frac{1}{2}$, considering it is a as coin flip

$$= Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - Pr[\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta) = true]$$

as criteria γ and γ_1 have the same verifier V_1, $PrRand^{\gamma} = PrRand^{\gamma_1}$ so

$$\begin{aligned}
&= Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - PrRand^{\gamma} \\
&\quad + PrRand^{\gamma_1} - Pr[\mathbf{G}^{\gamma_1}_{\mathcal{A}'}(\eta) = true] \\
&= \frac{1}{2}\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) - \frac{1}{2}\mathbf{Adv}^{\gamma_1}_{\mathcal{A}'}(\eta)
\end{aligned}$$

Let us now sum up what we have proved, this is done in the simplified partition theorem. This theorem is simplified as the verifier can only depend on the output of Θ_1.

Theorem 5.1 (Criterion Partition) *Let γ be the criterion $(\Theta_1, \Theta_2; F_1, F_2; V_1)$ where :*

1. *V_1 and F_1 only depend on the challenge generated by Θ_1, denoted by θ_1.*
2. *There exist some PRTM f, f' and g such that F_2 is constituted of two parts: $\lambda s.f(g(s,\theta_1),\theta_2)$ and $\lambda s.f'(s,\theta_2)$*

Then, for any adversary $\mathcal{A}$ against criterion γ, there exist two adversaries $\mathcal{B}$ and $\mathcal{A}'$, such that :

$$\forall \eta, \boldsymbol{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 2\boldsymbol{Adv}^{\gamma_2}_{\mathcal{B}}(\eta) + \boldsymbol{Adv}^{\gamma_1}_{\mathcal{A}'}(\eta)$$

where $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ is an indistinguishability criterion and $\gamma_1 = (\Theta_1; F_1; V_1)$.

Multiple Verifiers A quick generalization of this theorem consists in allowing sets of verifiers for V_1 instead of just one verifier. Let us verify that the theorem is still true. Let γ be the criterion $(\Theta_1, \Theta_2; F_1, F_2; V_1^1, \ldots, V_1^\alpha)$ where :

1. For any i in $[1, \alpha]$, V_1^i and F_1 only depend on the challenge generated by Θ_1, denoted by θ_1.
2. There exist some PRTM f, f' and g such that F_2 is constituted of two parts: $\lambda s.f(g(s,\theta_1),\theta_2)$ and $\lambda s.f'(s,\theta_2)$

Let γ^i denote criterion $(\Theta_1, \Theta_2; F_1, F_2; V_1^i)$ and $\mathcal{A}$ be an adversary against criterion γ. The advantage of $\mathcal{A}$ is defined by:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = \max_{1 \leq i \leq \alpha} \mathbf{Adv}_{\mathcal{A}}^{\gamma^i}$$

Then theorem 5.1 can be applied to each criteria γ^i. Hence for each i, there exist two adversaries $\mathcal{B}_i$ and $\mathcal{A}'_i$, such that:

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\gamma_i}(\eta) = 2\mathbf{Adv}_{\mathcal{B}_i}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'_i}^{\gamma_1^i}(\eta)$$

where $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ does not depend on i and $\gamma_1 = (\Theta_1; F_1; V_1^i)$.

The important point here is that by construction, adversary $\mathcal{A}'_i$ does not depend on i. Let us call $\mathcal{A}'$ this adversary. Hence by applying the max operation, the advantage of $\mathcal{A}$ against γ is bounded by:

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = 2 \max_{1 \leq i \leq \alpha} \left(\mathbf{Adv}_{\mathcal{B}_i}^{\gamma_2}(\eta)\right) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta)$$

Let $\mathcal{B}$ be the adversary $\mathcal{B}_i$ whose advantage is the largest of the advantages of all the $\mathcal{B}_i$. Then we get:

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = 2\mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta)$$

Therefore, theorem 5.1 is still true when considering that V_1 is a set of verifiers instead of just a verifier.

Another generalization is to consider that in the set of verifiers, some of them depend only on θ_1 and some of them depend only on θ_2. In this case, the result has to be modified and this leads to the (not-simplified) partition theorem.

5.3 Partition Theorem

We can generalize the partition theorem to the case where the verifier V has two parts V_1 and V_2 such that V_i only depends on challenge θ_i generated by Θ_i. The advantage related to the part of the criterion using V_1 can be bounded by the simplified partition theorem whereas the advantage related to V_2 has a very simple (yet interesting) bound. This theorem is useful when considering criteria like N-SYM-CPA where there is a common verifier (that checks the challenge bit b in N-SYM-CPA) and some challenge related verifiers (that check for forgeability in N-SYM-CPA).

Theorem 5.2 (Extended Criterion Partition) *Let γ be the criterion $(\Theta_1, \Theta_2; F_1, F_2; V_1, V_2)$ where :*

1. *V_1 and F_1 only depend on the challenge generated by Θ_1, denoted by θ_1. V_2 only depends on θ_2 (challenge generated by Θ_2).*
2. *There exist some PRTM f, f' and g such that F_2 is constituted of two parts: $\lambda s.f(g(s, \theta_1), \theta_2)$ and $\lambda s.f'(s, \theta_2)$*

Then, for any adversary $\mathcal{A}$ against criterion γ, there exist three adversaries $\mathcal{B}$, $\mathcal{A}'$ and $\mathcal{A}''$, such that :

$$\forall \eta, \boldsymbol{Adv}_{\mathcal{A}}^{\gamma}(\eta) = 2\boldsymbol{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \boldsymbol{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta) + \boldsymbol{Adv}_{\mathcal{A}''}^{\gamma_3}(\eta)$$

where $\gamma_3 = (\Theta_2; f, f'; V_2)$ is the criterion related to V_2, $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ is an indistinguishability criterion and $\gamma_1 = (\Theta_1; F_1; V_1)$ is the main criterion.

Proof: Let $\mathcal{A}$ be an adversary against γ. Even if V_1 and V_2 are themselves sets of verifiers, the advantage of $\mathcal{A}$ against γ is given by:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = \max\left(\mathbf{Adv}_{\mathcal{A}}^{\gamma, V_1}(\eta), \mathbf{Adv}_{\mathcal{A}}^{\gamma, V_2}(\eta)\right)$$

When considering the advantage of $\mathcal{A}$ against verifier V_1, theorem 5.1 applies. Hence there exists an adversary $\mathcal{A}'$ against γ_1 and an adversary $\mathcal{B}$ against γ_2 such that:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = \max\left(2\mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta), \mathbf{Adv}_{\mathcal{A}}^{\gamma, V_2}(\eta)\right)$$

Let us now define an adversary $\mathcal{A}''$ against criterion γ_3 that also uses $\mathcal{A}$ as a sub-routine. Adversary $\mathcal{A}'$ simulates the oracles related to Θ_1 by generating its own challenge.

Adversary $\mathcal{A}''(\eta)/\ \mathcal{O}_1, \mathcal{O}_2$:
θ_1:=$\Theta_1(\eta)$
bs:=$\mathcal{A}(\eta)/\lambda s.F_1(s,\theta_1)$, // oracle F_1
$\lambda s.\mathcal{O}_1(g(s,\theta_1))$, // part f of oracle F_2
$\mathcal{O}_2$ // part f' of oracle F_2
return bs

Then the game that involves $\mathcal{A}$ against γ, V_2 and the game that involves $\mathcal{A}''$ against γ_3 are the same. Moreover, the PrRand of these two criteria are identical. Hence,

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma,V_2}(\eta) = \mathbf{Adv}_{\mathcal{A}''}^{\gamma_3}(\eta)$$

Therefore the advantage of $\mathcal{A}$ against γ is given by:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = \max\left(2\mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta), \mathbf{Adv}_{\mathcal{A}''}^{\gamma_3}(\eta)\right)$$

The last step is easy to achieve. The advantage of $\mathcal{A}$ is either given by the first argument of the max or by the second argument. Let us first consider that:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = 2\mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta)$$

Then instead of using the $\mathcal{A}''$ defined above, we consider the adversary $\mathcal{A}''$ whose advantage is null (such adversary exist according to the definition of PrRand). We get that:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma}(\eta) = 2\mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta) + \mathbf{Adv}_{\mathcal{A}''}^{\gamma_3}(\eta)$$

In the other case, the advantage of $\mathcal{A}$ is equal to the advantage of $\mathcal{A}''$. Hence we use two adversaries $\mathcal{A}'$ and $\mathcal{B}$ whose advantages are null in order to get the same equality. ■

This theorem still holds when V_1 and V_2 are not verifiers but sets of verifiers.

5.4 Proving the Equivalence of Different Criteria

5.4.1 Proofs that do not use the Partition Theorem

In this section, we give the proofs for some results from previous sections. These proofs relate different criteria but do not use the partition theorem introduced above. Instead, a direct argument is used. The two first propositions we want to prove were stated in chapter 3.

Proposition 3.5 *If an asymmetric encryption scheme is secure against IND-CCA, then it is also secure against IND-CPA.*

Let $\mathcal{A}$ be an adversary against IND-CPA. Then the game involving $\mathcal{A}$ against IND-CCA and the game involving $\mathcal{A}$ against IND-CPA are equivalent as $\mathcal{A}$ does not call the decryption oracle. So the probabilities of success are the same:

$$Pr[\mathbf{G}_{\mathcal{A}}^{IND-CCA}(\eta) = true] = Pr[\mathbf{G}_{\mathcal{A}}^{IND-CPA}(\eta) = true]$$

As both $PrRand$ are equal to 1/2, $\mathcal{A}$ has the same advantage in both cases:

$$\mathbf{Adv}_{\mathcal{A}}^{IND-CCA}(\eta) = \mathbf{Adv}_{\mathcal{A}}^{IND-CPA}(\eta)$$

If the asymmetric encryption scheme used here is secure against IND-CCA, then the advantage of $\mathcal{A}$ against IND-CCA is negligible and so is the advantage of $\mathcal{A}$ against IND-CPA. Thus, it is possible to conclude that the encryption scheme is secure against IND-CPA.

Proposition 3.6 *If there exists an asymmetric encryption scheme secure against IND-CPA, then there exists an asymmetric encryption scheme that is secure against IND-CPA but not secure against IND-CCA.*

Let $\mathcal{AE}$ be an asymmetric encryption scheme $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$ secure against IND-CPA. Then it is possible to create from $\mathcal{AE}$ an encryption scheme that is still secure against IND-CPA but which is insecure against IND-CCA. We consider the encryption scheme $\mathcal{AE}' = (\mathcal{KG}', \mathcal{E}', \mathcal{D}')$, where the different components are defined by:

1. The key generation algorithm $\mathcal{KG}'$ is the same as $\mathcal{KG}$.
2. The encryption algorithm $\mathcal{E}'$ uses $\mathcal{E}$ and appends a random bit to the output of $\mathcal{E}$.

   Algorithm $\mathcal{E}'(bs, pk)$:
   $bs := \mathcal{E}(bs, pk)$
   $b' \xleftarrow{R} [0, 1]$
   return $bs.b'$

3. The decryption algorithm $\mathcal{D}'$ is defined in order to correctly decrypt cipher-texts produced by $\mathcal{E}'$. This algorithm does not take the last bit value into account.

   Algorithm $\mathcal{D}'(bs.b, sk)$:
   return $\mathcal{D}(bs, sk)$

Then encryption scheme $\mathcal{AE}'$ is not secure against IND-CCA as it is possible to build an adversary $\mathcal{A}$ which has a non-negligible advantage. This adversary submits the pair $\langle 0, 1\rangle$ to his encryption oracle. He receives the encryption $\mathcal{E}(b, pk)$ appended to a random bit b'. Then $\mathcal{A}$ submits $\mathcal{E}(b, pk).b''$ where b'' is the opposite of b' to its decryption oracle. This bit-string has not been produced by the encryption oracle hence the decryption oracle returns b and $\mathcal{A}$ is able to win its challenge. Thus the advantage of $\mathcal{A}$ is 1 and $\mathcal{AE}'$ is not secure against IND-CCA.

Now let us verify that encryption scheme $\mathcal{AE}'$ is secure against IND-CPA. Let $\mathcal{A}$ be an adversary against IND-CPA that uses $\mathcal{AE}'$. Then we build an adversary $\mathcal{A}'$ against IND-CPA (but IND-CPA for $\mathcal{AE}$). This adversary uses $\mathcal{A}$ as a sub-routine and generates the random bit needed at the end of encryptions.

Adversary $\mathcal{A}'(\eta)/\mathcal{O}_1, \mathcal{O}_2$:
$bs := \mathcal{A}/\mathcal{O}_1$
$\lambda s.b' \xleftarrow{R} [0, 1];$
return $\mathcal{O}_2(s).b'$
return bs

Then the games involving $\mathcal{A}$ and $\mathcal{A}'$ are the same, thus the advantages of $\mathcal{A}$ and $\mathcal{A}'$ are also the same. As $\mathcal{AE}$ is secure against IND-CPA, then the advantage of $\mathcal{A}$ is negligible and so is the advantage of $\mathcal{A}'$. Encryption scheme $\mathcal{AE}'$ is secure against IND-CPA and insecure against IND-CCA.

5.4.2 Using the Partition Theorem

The partition theorem introduced above can be used in a wide variety of situations. As a first example of its use, we consider the proposition that motivated the theorem. This proposition states the equivalence between IND-CPA and 2-KDM-IND-CPA. This result was already proved as an example of partition in section 5.1, however the partition theorem allows us to prove the proposition without having to describe the different adversaries. These adversaries descriptions are embedded in the proof of the theorem.

Proposition 5.3 *Let $\mathcal{AE}$ be an asymmetric encryption scheme. If $\mathcal{AE}$ is secure against IND-CPA then $\mathcal{AE}$ is secure against 2-KDM-IND-CPA.*

Proof: In this proof, we describe precisely how the partition theorem is applied. Further proofs using the theorem will be made more quickly.

Let $\mathcal{AE}$ be an asymmetric encryption scheme composed of the key generation algorithm $\mathcal{KG}$, the encryption algorithm $\mathcal{E}$ and the decryption algorithm $\mathcal{D}$.

Let γ denote criterion 2-KDM-IND-CPA and $\mathcal{A}$ be an adversary against γ. The verifier of γ only depends on a single challenge: bit b, hence we use the simplified version of the partition theorem (theorem 5.1). Two key pairs (pk_1, sk_1) and (pk_2, sk_2) are generated. The left-right encryption oracle related to pk_1 takes as argument functions that can use the values of pk_2 and sk_2. Hence the idea is to generate key pair (pk_1, sk_1) in Θ_2 and the other key pair in Θ_1, thus the oracle related to pk_1 can use pk_2 and sk_2 whereas the oracle related to pk_2 cannot use pk_1 or sk_1. As challenge bit b is used by both left-right oracles, this bit is generated in Θ_1. Let us consider the partition of γ that is given by $(\Theta_1, \Theta_2; F_1, F_2; V_1)$ where:

1. Θ_1 randomly generates a key pair (pk_2, sk_2) using $\mathcal{KG}$, it also randomly samples the challenge bit b. The output of Θ_1 is denoted by θ_1 in the following.

2. Θ_2 randomly generates a key pair (pk_1, sk_1) using $\mathcal{KG}$. The output of Θ_2 is denoted by θ_2 in the following.

3. F_1 contains the oracles related to key pair (pk_2, sk_2). Acyclicity implies that F_1 only depends on θ_1.

4. Oracle F_2 contains the oracles related to key pair (pk_1, sk_1). Hence this oracle can be split in two parts:
$$\lambda s.f(g(s, \theta_1), \theta_2) \qquad \text{and} \qquad \lambda s.f'(s, \theta_2)$$
The f' part contains the public key oracle. Therefore f' only depends on θ_2.

   The other part is separated in two layers and is used for the left-right encryption oracle. The g layer is submitted a pair of functions $\langle f_0, f_1\rangle$. It returns the result given by applying function f_b to the different keys from θ_1. Thus this layer only depends on θ_1 and is formally defined by:
$$g(\langle f_0, f_1\rangle, \theta_1) = f_{b\theta_1}(pk_2\theta_1, sk_2\theta_1)$$
   Layer f performs the encryption part:

   Algorithm $f(bs, \theta_2)$:
   $out := \mathcal{E}(bs, pk_1\theta_2)$
   return out

   Note that f takes a bit-string as argument and that it only depends on θ_2.

5. Verifier V_1 checks that the adversary correctly output the value of challenge bit b from θ_1. Hence V_1 only depends on θ_1.

Note that it is possible to place the public key oracle from F_2 in the first part. However after applying the theorem, this would lead to criteria that cannot be related to IND-CPA. The advantage of adversaries against these new criteria is not negligible hence it is impossible to conclude that the advantage of adversary $\mathcal{A}$ is negligible.

The partition described above is valid and theorem 5.1 can be applied. Thus there exist two adversaries $\mathcal{B}$ and $\mathcal{A}'$ such that:

$$\forall \eta, \mathbf{Adv}^{\delta_N}_{\mathcal{A}}(\eta) = 2\mathbf{Adv}^{\gamma_2}_{\mathcal{B}}(\eta) + \mathbf{Adv}^{\gamma_1}_{\mathcal{A}'}(\eta)$$

where $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ is an indistinguishability criterion and $\gamma_1 = (\Theta_1; F_1; V_1)$.

In criterion γ_2, a key pair (pk, sk) is randomly generated along with a challenge bit b. The adversary has access to an oracle f' that outputs a public key and to another oracle $f \circ LR^b$ that performs left-right encryption. Finally, an adversary has to deduce the value of b to win its game. Hence this criterion is exactly IND-CPA. As we assumed that $\mathcal{AE}$ is secure against IND-CPA, the advantage of $\mathcal{B}$ is negligible.

Criterion γ_1 is close to IND-CPA, however it is not exactly the same thing. A key pair and a challenge bit are generated in both cases. The adversary still has access to a public key oracle and has to find the value of the challenge bit. But the left-right encryption oracles are different: in the case of IND-CPA, this oracle takes as input a pair of bit-strings whereas in the case of γ_1, it takes as argument a pair of functions (that are applied to zero arguments because of acyclicity). Let $\mathcal{C}$ be an adversary against γ_1, it is easy to design an adversary $\mathcal{C}'$ against IND-CPA whose advantage is the same:

Adversary $\mathcal{C}'(\eta)/\mathcal{O}_{LR}, \mathcal{O}_D, \mathcal{O}_{PK}$:
$d := \mathcal{C}(\eta)/\lambda\langle f_0, f_1\rangle.\mathcal{O}_{LR}\big(\langle f_0(), f_1()\rangle\big)$
$\mathcal{O}_D$
$\mathcal{O}_{PK}$
return d

The games involving $\mathcal{C}'$ against IND-CPA and $\mathcal{C}$ against γ_1 are the same thus the advantages of these two adversaries are equal. Hence security against IND-CPA implies security against γ_1. We suppose that $\mathcal{AE}$ is secure for IND-CPA, then it is secure for γ_1 and the advantage of adversary $\mathcal{A}'$ is negligible.

Finally, we proved that the advantage of $\mathcal{A}$ is negligible for any adversary $\mathcal{A}$. Therefore, it is possible to conclude that $\mathcal{AE}$ is secure for 2-KDM-IND-CPA. ■

This example is used to show how a previous proof can be done using the partition theorem. However it does not fully take advantage of the theorem as the partition technique is only applied once. A typical use of the partition theorem is to break a criterion involving N keys into N criteria using a single key. This can be used for example to relate N-PAT-IND-CCA to IND-CCA as shown in the following section.

5.5 Relating our Security Criteria to Classical Ones

The aim of this section is to relate the different criteria that we introduced in section 4.5 to classical criteria in provable cryptography. Classical criteria were introduced in chapter 3. For this purpose, we use the partition theorems introduced previously in this chapter.

5.5.1 Asymmetric Encryption

Concerning asymmetric encryption schemes, we use the simplified partition theorem in order to prove two different results. The first one states that an asymmetric encryption scheme is secure against IND-CCA iff it is secure against N-PAT-IND-CCA (for $N > 0$). The second one relates IND-CCA to cycle free KDM: an encryption scheme is secure against N-KDM-IND-CCA iff it is secure against IND-CCA.

N-PAT-IND-CCA is Equivalent to IND-CCA

Using our partition theorem, we are interested in proving the following proposition.

Proposition 5.4 *Let N be an integer. If an asymmetric encryption scheme $\mathcal{AE}$ is IND-CCA, then $\mathcal{AE}$ is N-PAT-IND-CCA.*

We want to establish first that an IND-CCA asymmetric encryption scheme turns out to be an N-PAT-IND-CCA secure one. We use the criterion partition theorem on N-PAT-IND-CCA

(denoted by δ_N). We now consider $\delta_N = (\Theta_1, \Theta_2; F_1, F_2; V_1)$, where the criterion partition has been performed the following way:

- Θ_1 randomly generates the challenge bit b and $N-1$ pairs of matching public and secret keys (pk_i, sk_i) for i between 2 and N along with their related memories (i.e. mutable fields).
- Θ_2 randomly generates the first key pair (pk_1, sk_1) and the related mutable field mem_1.
- F_1 contains the oracles related to Θ_1, i.e. ELR_i^a, D_i and PK_i for i between 2 and N; hence as neither pk_1 or sk_1 can be asked to this oracle (because of acyclicity), F_1 does not depend on Θ_2.
- F_2 contains the oracles related to key pair 1, i.e. ELR_1^a, D_1 and PK_1, it uses θ_1 (generated by Θ_1) for the bit b and the different keys needed to fill in patterns.
- V_1 compares the output to b, and therefore only depends on θ_1.

This splitting complies with the first hypothesis of theorem 5.1. Let us then check whether the second hypothesis holds. The decryption and public key oracles D_1 and PK_1 included in F_2 only depend on θ_2 (generated by Θ_2), we place them in f'. We let the encryption oracle be $\lambda s.f(g(s,\theta_1),\theta_2)$ where $g(\langle pat_0, pat_1\rangle, \theta_1) = concr(pat_{b\theta_1}, \theta_1)$ plays the role of a left-right oracle: it uses the challenge bit b included in θ_1 to select pattern pat_b, then this pattern is evaluated using the *concr* function. Moreover, the acyclicity hypothesis implies that pattern variables related to key pair (pk_1, sk_1) cannot occur in the patterns, hence the result of the application of the *concr* function is a bit-string. The second layer f performs the encryption itself and stores the result in mem_1:

Algorithm $f(bs, \theta_2)$:
$out := \mathcal{E}(bs, pk_1\theta_2)$
$mem_1\theta_2 := out :: mem_1\theta_2$
return out

This is the original encryption oracle: the bit-string is encrypted and the result is stored in mem_1.

The theorem can now be applied. It thus follows that for any adversary $\mathcal{A}$ against criterion δ_N, there exist two adversaries $\mathcal{B}$ and $\mathcal{A}'$, such that :

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\delta_N}(\eta) = 2.\mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta)$$

where $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ and $\gamma_1 = (\Theta_1; F_1; V_1)$. In criterion γ_2, the challenge generator randomly produces a key pair (pk_1, sk_1) and a challenge bit b. The adversary has to guess the value of bit b and can use three oracles: the left-right encryption oracle related to pk_1, a decryption oracle using sk_1 (that does not work on outputs of the encryption oracle) and a public key oracle that outputs pk_1. Criterion γ_2 is the classical IND-CCA criterion. In criterion γ_1, the challenge generator produces $N-1$ key pairs, their relating memories and a bit b. The adversary also has to guess the value of bit b and can use for each key a left-right encryption oracle, a decryption oracle and a public key oracle. The left-right encryption oracle works on pairs of acyclic patterns, hence γ_1 is criterion δ_{N-1}.

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\delta_N}(\eta) = 2.\mathbf{Adv}_{\mathcal{B}}^{IND-CCA}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\delta_{N-1}}(\eta)$$

An easy recursion on N gives us that for any adversary $\mathcal{A}$, there exist an adversary $\mathcal{A}'$ and N adversaries $\mathcal{B}_1$ to $\mathcal{B}_N$ such that:

$$\mathbf{Adv}_{\mathcal{A}}^{\delta_N}(\eta) = 2.\sum_{i=1}^{N} \mathbf{Adv}_{\mathcal{B}_i}^{IND-CCA}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\delta_0}(\eta)$$

Criterion δ_0 is composed of a challenge generator which randomly produces a bit b, no oracle and to win, an adversary has to guess the value of b. Using proposition 4.6, we get that this advantage

is negative (in fact, this advantage is 0 if we consider adversaries whose final output can only be 0 or 1). The advantage of $\mathcal{A}$ can now be bounded by:

$$\mathbf{Adv}_{\mathcal{A}}^{\delta_N}(\eta) = 2.\sum_{i=1}^{N} \mathbf{Adv}_{\mathcal{B}_i}^{IND-CCA}(\eta)$$

The asymmetric encryption scheme used here is IND-CCA, so the advantage of any adversary against IND-CCA is p-negligible. As the sum of a fixed number of p-negligible functions is p-negligible, we have that the advantage of $\mathcal{A}$ is p-negligible. We can now conclude that the asymmetric encryption scheme used here is also safe for N-PAT-IND-CCA.

Cycle Free N-KDM-IND-CCA is equivalent to IND-CCA

KDM security has been introduced in section 4.5.1. In this section, we prove that under the hypotheses of acyclicity, security against N-KDM-IND-CCA is equivalent to security against IND-CCA if N is a strictly positive integer. This is formalized by the next proposition.

Proposition 5.5 *Let $\mathcal{AE}$ be an asymmetric encryption scheme. Then for any strictly positive integer N, $\mathcal{AE}$ is secure against IND-CCA iff it is secure against N-KDM-IND-CCA.*

Proof: Proposition 4.20 states that if $\mathcal{AE}$ is secure against N-KDM-IND-CCA, then it is secure against N-PAT-IND-CCA. Moreover proposition 4.13 proves that if $\mathcal{AE}$ is secure against N-PAT-IND-CCA then $\mathcal{AE}$ is secure from IND-CCA. Hence it is easy to conclude that if $\mathcal{AE}$ is secure against N-KDM-IND-CCA then it secure against IND-CCA.

The converse is more interesting and we use the simplified partition theorem 5.1 to prove it. We proceed using an induction over N. Let N be an integer and let $\mathcal{AE}$ be an asymmetric encryption scheme that is secure against IND-CCA. Let δ_N denote criterion N-KDM-IND-CCA and $\mathcal{A}$ be an adversary against δ_N. A possible partition of δ_N is $(\Theta_1, \Theta_2; F_1, F_2; V_1)$ where:

1. Challenge generator Θ_1 randomly generates a bit b along with $N-1$ key pairs (pk_2, sk_2) to (pk_N, sk_N) (using the key generation algorithm from $\mathcal{AE}$) and their related mutable fields mem_2 to mem_N. Substitution θ_1 designates the output of Θ_1.

2. Challenge generator Θ_2 randomly generates a key pair (pk_1, sk_1) (using the key generation algorithm from $\mathcal{AE}$) and the related mutable field mem_1. Substitution θ_2 designates the output of Θ_2.

3. Oracle F_1 contains all the oracles related to the $N-1$ key pairs from θ_1. The acyclicity requirement prevents pk_1 and sk_1 from being used by functions submitted to these oracles, F_1 only depends on Θ_1.

4. Oracle F_2 contains the oracle related to key pair (pk_1, sk_1). Hence this oracle can be split in two parts:

   $$\lambda s.f(g(s,\theta_1),\theta_2) \quad \text{and} \quad \lambda s.f'(s,\theta_2)$$

   The f' part contains the decryption and public key oracles. Therefore f' only depends on θ_2.

   The other part is separated in two layers and is used for the left-right encryption oracle. The g layer is submitted a pair of functions $\langle f_0, f_1\rangle$. It returns the result given by applying function f_b to the different keys from θ_1. Thus this layer only depends on θ_1 and is formally defined by:

   $$g(\langle f_0, f_1\rangle, \theta_1) = f_b(pk_2\theta, sk_2\theta \cdots pk_N\theta, sk_N\theta)$$

   Layer f performs the encryption part:

   Algorithm $f(bs, \theta_2)$:
   $out := \mathcal{E}(bs, pk_1\theta_2)$
   $mem_1\theta_2 := out :: mem_1\theta_2$
   return out

Note that f takes bit-strings as arguments and that it only depends on θ_2.

5. Verifier V_1 checks that the adversary correctly guessed the value of b.

We have built a valid partition of criterion δ_N, thus the simplified version of the partition theorem can be applied. Hence there exist an adversary $\mathcal{B}$ and an adversary $\mathcal{A}'$ such that:

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\delta_N}(\eta) = 2.\mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta)$$

where $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ is an indistinguishability criterion and $\gamma_1 = (\Theta_1; F_1; V_1)$.

In criterion γ_2, a key pair (pk_1, sk_1) is randomly generated along with a challenge bit b. The adversary has access to an oracle f' containing the public key and decryption oracles and to another oracle $f \circ LR^b$ that performs left-right encryption. Finally, an adversary has to deduce the value of b to win its game. Hence this criterion is exactly IND-CCA. As we assumed that $\mathcal{AE}$ is secure against IND-CCA, the advantage of $\mathcal{B}$ is p-negligible.

Criterion γ_1 is constituted as follows: $N-1$ challenge key pairs and a random bit b are randomly sampled. Then the adversary has access to the classical KDM oracles for these key pairs. Verifier V_1 checks that the adversary correctly guessed the value of the challenge bit b. Hence γ_1 is criterion δ_{N-1}.

Let us suppose that the encryption scheme is secure against $(N-1)$-KDM-IND-CCA, then it is secure for criterion γ_1 and so the advantage of $\mathcal{A}'$ is also p-negligible. In this case the advantage of $\mathcal{A}$ is p-negligible and it is possible to conclude that $\mathcal{AE}$ is secure against N-KDM-IND-CCA.

A quick induction gives the awaited result: in criterion δ_0, a random bit is generated and the adversary has to guess its value without any oracle. Therefore, the advantage of any adversary against δ_0 is null and so is p-negligible. Encryption scheme $\mathcal{AE}$ is secure against 0-KDM-IND-CCA. Moreover, if $\mathcal{AE}$ is secure against $(N-1)$-KDM-IND-CCA then it is secure against N-KDM-IND-CCA. Therefore, for any N, $\mathcal{AE}$ is secure against N-KDM-IND-CCA. ■

An immediate corollary of this proposition is an extension of proposition 4.20: safety for N-KDM-IND-CCA and safety for N-PAT-IND-CCA are equivalent.

Corollary 5.1 *Let $\mathcal{AE}$ be an asymmetric encryption scheme and N and M be two strictly positive integers. Then $\mathcal{AE}$ is secure against N-KDM-IND-CCA (with the acyclicity requirement) iff it is secure against M-PAT-IND-CCA.*

An important restriction of our results is that the number of challenge keys N is fixed and does not depend on the security parameter η. Thus an interesting extension detailed in chapter 8 consists in allowing a polynomial number of challenge keys.

5.5.2 Digital Signature

The third result that we want to prove in this section is related to digital signature. This result says that a signature scheme that is secure in a single user setting (for the UNF criterion) is also secure in the multi-user setting (i.e. criterion N-UNF).

Proposition 5.6 *Let N be an integer. If a signature scheme $\mathcal{SS}$ is UNF, then $\mathcal{SS}$ is N-UNF.*

There are two different ways to prove this result. As the different oracles do not share any knowledge (e.g. there is no common challenge bit or no shared memory across oracles), it is possible to perform the proof in a straightforward way without using our partition theorem. This proof is done in the following subsection.

However, as the objective of this section is to illustrate the use of the extended criterion partition theorem (theorem 5.2), it is also possible to use this result in order to prove the former proposition. The extended version of the theorem is required here as the verifiers depend on all the generated challenges. The main advantage of using the theorem in this case is that no adversary has to be detailed. However although it applies, the partition theorem is not perfectly suited as it creates an indistinguishability criterion.

Both cases use a digital signature scheme $\mathcal{SS}$ composed of a key generation algorithm $\mathcal{KG}$, a signing algorithm $\mathcal{S}$ and a verification algorithm $\mathcal{V}$.

Classical Proof

Oracles related to the different signature keys are totally decorrelated, hence it is possible to simulate them perfectly. Let $\mathcal{A}$ be an adversary against N-UNF. Let i be an integer between 1 and N and $\mathcal{B}_i$ be N adversaries against UNF which use $\mathcal{A}$ as a subroutine. Oracles that $\mathcal{B}_i$ can access are used for the i^{th} key of $\mathcal{A}$. Other keys are perfectly simulated using algorithms from $\mathcal{SS}$. Adversaries $\mathcal{B}_i$ can access one signature oracle denoted by $\mathcal{O}_1$ in the following and one verification key oracle denoted by $\mathcal{O}_2$. Formal definition of $\mathcal{B}_i$ is given by the algorithm described thereafter.

Adversary $\mathcal{B}_i(\eta)/\mathcal{O}_1, \mathcal{O}_2$:
  for i **from** 1 **to** N
    $(sik_i, vk_i) := \mathcal{KG}(\eta)$
  $d := \mathcal{A}(\eta)/\lambda s.\mathcal{S}(s, sik_1)$
    $\lambda().vk_1$
    $\ldots$
    $\lambda s.\mathcal{O}_1(s)$
    $\lambda().\mathcal{O}_2()$
    $\ldots$
  return d

Let us denote criterion N-UNF by γ_N. We also denote by γ_N, V_i the N-UNF criterion where V_i is the only considered verifier. The game where $\mathcal{B}$ plays against γ_N, V_i is the same as the game where $\mathcal{B}_i$ plays against UNF. Then the advantage of $\mathcal{B}$ is defined by:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\gamma_N}(\eta) &= \max_{1\leq i\leq n} \mathbf{Adv}_{\mathcal{B}}^{\gamma_N, V_i}(\eta) \\ &= \max_{1\leq i\leq n} \mathbf{Adv}_{\mathcal{B}_i}^{\gamma_1}(\eta)\end{aligned}$$

We suppose that the signature scheme is safe for UNF, hence the advantages of all the different $\mathcal{B}_i$ are negligible. As there are only a fixed number of $\mathcal{B}_i$, there is a common negligible bound to the advantages of the $\mathcal{B}_i$. Thus the advantage of $\mathcal{B}$ is negligible and $\mathcal{SS}$ is safe for N-UNF.

Proof Using the Partition Theorem

We use theorem 5.2 to prove the result. Let δ_N denote criterion N-UNF. We consider the following partition of δ_N:

- Θ_1 randomly generates key pair (sik_1, vk_1) using $\mathcal{KG}$ and the related memory mem_i.
- Θ_2 randomly generates the other key pairs (sik_2, vk_2) to (sik_N, vk_N) using $\mathcal{KG}$ as well as their related memories mem_2 to mem_N.
- F_1 contains the signature oracle related to sik_1.
- F_2 contains the other signature oracles.
- V_1 contains the verifier related to vk_1
- V_2 contains the other verifiers.

As F_1 depends only on θ_1 and F_2 depends only on θ_2, we consider that F_2 is only composed of an f' part (the f part is not required since F_2 does not depend on θ_1).

Applying theorem 5.2, we get that for any adversary $\mathcal{A}$ against δ_N, there exist three adversaries $\mathcal{B}$, $\mathcal{A}'$ and $\mathcal{A}''$, such that:

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\delta_N}(\eta) \leq 2 \cdot \mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta) + \mathbf{Adv}_{\mathcal{A}''}^{\gamma_3}(\eta)$$

Criterion $\gamma_3 = (\Theta_2; f'; V_2)$ is composed by a challenge generator that creates $N-1$ key pairs using $\mathcal{KG}$ and their related memories. It contains the signature oracles for the previous keys. To win, the adversary has to produce a fresh valid signature for vk_i where $i \geq 2$. Hence this criterion actually represents $(N-1)$-UNF. In criterion $\gamma_2 = (\Theta_2, b; f'; v_b)$, an adversary has to guess the value of bit b but none of the oracles use bit b, thus using proposition 4.7 the advantage of any adversary is negative. Finally, $\gamma_1 = (\Theta_1; F_1; V_1)$ is a criterion where a key pair is generated (and its related memory), there is a signing oracle and a verification key oracle for that key pair and the adversary has to produce a valid fresh signature. Hence this is the classical UNF criterion.

$$\mathbf{Adv}^{\delta_N}_{\mathcal{A}}(\eta) \leq \mathbf{Adv}^{UNF}_{\mathcal{A}'}(\eta) + \mathbf{Adv}^{\delta_{N-1}}_{\mathcal{A}''}(\eta)$$

By iterating this, we get that there exist N adversaries $\mathcal{A}_1$ to $\mathcal{A}_N$ such that:

$$\mathbf{Adv}^{\delta_N}_{\mathcal{A}}(\eta) \leq \sum_{i=1}^{N} \mathbf{Adv}^{UNF}_{\mathcal{A}_i}(\eta)$$

If the signature scheme used here $\mathcal{SS}$ is safe for UNF, then the advantages of the different $\mathcal{A}_i$ are p-negligible. Hence, the advantage of $\mathcal{A}$ is also p-negligible and signature scheme $\mathcal{SS}$ is safe for N-UNF.

5.5.3 Symmetric Encryption

Using the partition theorem, it is possible to reduce criterion N-PAT-SYM-CPA to criterion IND-CPA and criterion UNF. Hence, we aim at proving the following proposition:

Proposition 5.7 *Let N be an integer. If a symmetric encryption scheme $\mathcal{SE}$ is IND-CPA and UNF, then $\mathcal{SE}$ is N-PAT-SYM-CPA.*

This time, we want to reduce the N-PAT-SYM-CPA criterion, denoted by δ_N, to SYM-CPA security. We need the extended version of the partition theorem: as defined in subsection 4.5.3, an adversary can indeed win by forging a fresh encryption for any pair of challenge keys, the hypothesis following which the verifier V only depends on challenges generated by Θ_1 is thus too restrictive.

The criterion partition can be achieved as follows:

- Θ_1 randomly generates the bit b, the $N-1$ challenge keys k_2 to k_N and the related mutable fields mem_2 to mem_N. Key generation is achieved using $\mathcal{KG}$.
- Θ_2 randomly generates the first key k_1, thanks to $\mathcal{KG}$, it also creates a mutable field mem_1.
- F_1 contains the oracles related to θ_1. Because of acyclicity, F_1 does not depend on θ_2.
- F_2 contains the oracles related to key k_1, it uses the bit b and the different keys, needed to fill in patterns, that are contained in θ_1.
- V_1 contains N verifiers. The first one compares the output of the adversary to b, the $N-1$ other ones test whether a fresh encryption has been forged using key k_2 to k_N. Hence all the verifiers from V_1 only depend on θ_1.
- V_2 tests that the output of the adversary is a fresh encryption using k_1, it only depends on θ_2.

Before applying the extended theorem 5.2, let us describe the partition of oracles we choose so that the theorem hypothesis hold. The public key oracle from F_2 only depends on θ_2, we let it constitute f'. The encryption oracle can be written as $\lambda s.f(g(s,\theta_1),\theta_2)$, with $g(\langle pat_0, pat_1\rangle, \theta_1) = concr(pat_{b\theta_1}, \theta_1)$ standing, as previously, for the composition of our concretization algorithm *concr* that fills in patterns and a left-right oracle depending on the challenge bit b included in θ_1, and with $f(bs, \theta_2) = \mathcal{E}(bs, pk_1)$, standing for the original encryption oracle.

Applying theorem 5.2 thus allows us to deduce that for any adversary $\mathcal{A}$ against criterion δ_N, there exist three adversaries $\mathcal{B}$, $\mathcal{A}'$ and $\mathcal{A}''$, such that :

$$\forall \eta, \mathbf{Adv}_{\mathcal{A}}^{\delta_N}(\eta) \leq 2 \cdot \mathbf{Adv}_{\mathcal{B}}^{\gamma_2}(\eta) + \mathbf{Adv}_{\mathcal{A}'}^{\gamma_1}(\eta) + \mathbf{Adv}_{\mathcal{A}''}^{\gamma_3}(\eta)$$

Criterion $\gamma_3 = (\Theta_2; f, f'; V_2)$ actually represents 1-PAT-SYM-CPA/UNF. Indeed, patterns are not used as arguments of f because of the acyclicity assumption. Moreover, V_2 stands for a verifier checking whether the adversary has forged a fresh encryption using key k_1; this is why criterion γ_3 is claimed to be SYM-CPA with respect to unforgeability. Criterion $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ is in fact 1-PAT-SYM-CPA/IND, that is, 1-PAT-SYM-CPA with respect to indistinguishability, since the verifier only checks on the value of the challenge bit b. Finally, $\gamma_1 = (\Theta_1; F_1; V_1)$ is meant to be criterion $(N-1)$-PAT-SYM-CPA.

To conclude, a recursion can be done as previously. Adversaries playing against 0-PAT-SYM-CPA actually have a null advantage. Hence any symmetric encryption scheme is secure against 0-PAT-SYM-CPA. Moreover, if a symmetric encryption scheme $\mathcal{SE}$ is secure against $(N-1)$-PAT-SYM-CPA and against SYM-CPA, then it is secure against N-PAT-SYM-CPA. Thus we get that if $\mathcal{SE}$ is secure against SYM-CPA, it is also secure against N-PAT-SYM-CPA.

5.5.4 Combining all the Cryptographic Primitives

As a last application of the partition theorem, we consider the N-PASS criterion that defines safety of a cryptographic library composed of an asymmetric encryption scheme, a symmetric encryption scheme and a digital signature scheme.

We aim at proving that safety of the whole cryptographic library is implied by safety of the asymmetric encryption scheme for IND-CCA, safety of the symmetric encryption scheme for SYM-CPA and safety of the digital signature scheme for UNF. Thus the combination of secure primitives allows one to constitute a secure library.

Proposition 5.8 *Let N be an integer. If an asymmetric encryption scheme $\mathcal{AE}$ is IND-CCA, a symmetric encryption scheme $\mathcal{SE}$ is SYM-CCA and a signature scheme $\mathcal{SS}$ is UNF, then the composition $(\mathcal{AE}, \mathcal{SE}, \mathcal{SS})$ is N-PASS.*

The acyclicity condition on N-PASS is more complex than the previous conditions for N-PAT-IND-CCA or N-PAT-SYM-CPA. There is an acyclicity condition for asymmetric keys, another acyclicity condition for symmetric keys and a global restriction: asymmetric encryption keys and asymmetric encryptions can only occur in patterns that are given to the left-right oracle for an asymmetric encryption key. Thus symmetric encryptions and symmetric keys can occur encrypted by an asymmetric encryption but the converse is forbidden.

This condition leads to a simpler proof than when considering a global acyclicity condition. The proof is modular and uses the propositions that were previously proved in this section. Let $(\mathcal{AE}, \mathcal{SE}, \mathcal{SS})$ be a cryptographic library. We first prove that safety of this cryptographic library is implied by safety of $\mathcal{AE}$ for N-PAT-IND-CCA and by safety of $(\mathcal{SE}, \mathcal{SS})$ for a new criterion N-PSS (criterion that combines symmetric encryption and digital signature). Then safety for N-PAT-IND-CCA is implied by safety for IND-CCA (see proposition 5.4). We also prove that safety of $(\mathcal{SE}, \mathcal{SS})$ for N-PSS is implied by safety of $\mathcal{SE}$ for N-PAT-SYM-CPA and safety of $\mathcal{SS}$ for N-UNF. Propositions 5.6 and 5.7 can then be used: safety of $\mathcal{SE}$ for N-PAT-SYM-CPA is implied by safety of $\mathcal{SE}$ for SYM-CPA and safety of $\mathcal{SS}$ for N-UNF is implied by safety of $\mathcal{SS}$ for UNF.

Breaking N-PASS into N-PAT-IND-CCA and N-PSS

Let us first define criterion N-PSS. This criterion involves a symmetric encryption scheme $\mathcal{SE}$ and a digital signature scheme $\mathcal{SS}$. First N symmetric keys and N signature keys are randomly generated. A challenge bit b is also generated. Adversaries have access to a left-right encryption oracle for each symmetric key and to two oracles for each signature key: one that gives access to

the verification key and a signature oracle. In order to win his game, the adversary can guess the value of b or can forge either a fresh symmetric encryption or a fresh signature. Formal definitions of the previous oracles appear in section 4.5.

The result of this section is stated in the following proposition: safety of $(\mathcal{AE}, \mathcal{SE}, \mathcal{SS})$ is implied by safety of $\mathcal{AE}$ for N-PAT-IND-CCA and by safety of $(\mathcal{SE}, \mathcal{SS})$ for the new N-PSS criterion.

Proposition 5.9 *Let N be an integer, $\mathcal{AE}$ be an asymmetric encryption scheme, $\mathcal{SE}$ be a symmetric encryption scheme and $\mathcal{SS}$ be a digital signature scheme. If $\mathcal{AE}$ is secure for N-PAT-IND-CCA and $(\mathcal{SE}, \mathcal{SS})$ is secure for N-PSS, then cryptographic library $(\mathcal{AE}, \mathcal{SE}, \mathcal{SS})$ is secure for N-PASS.*

Proof: Let us suppose that $\mathcal{AE}$ is secure for N-PAT-IND-CCA and $(\mathcal{SE}, \mathcal{SS})$ is secure for N-PSS. Let us use γ to denote criterion N-PASS. A partition of γ can be achieved in the following way, $\gamma = (\Theta_1, \Theta_2; F_1, F_2; V_1)$ where:

1. Θ_1 generates the challenge bit b, the N challenge signature key pairs and the N challenge symmetric keys (and the related memories). Let θ_1 denote the challenge generated by Θ_1.

2. Θ_2 generates the N challenge asymmetric key pairs (and the related memories). Let θ_2 denote the challenge generated by Θ_2.

3. F_1 contains all the oracles related to signature and symmetric encryption. As symmetric encryption or signature of asymmetric keys is forbidden, oracle F_1 only depends on θ_1.

4. F_2 contains all the oracles related to asymmetric encryption, i.e. left-right encryption oracles, public key oracles and decryption oracles. F_2 depends on θ_1 and on θ_2. F_2 is constituted of two parts:
$$\lambda s.f(g(s, \theta_1), \theta_2) \qquad \text{and} \qquad \lambda s.f'(s, \theta_2)$$
The first part contains the left-right encryption oracles. Layer g consists in applying the *concr* function and selecting either the left or right pattern according to the value of g. Formally, $g(\langle pat_0, pat_1\rangle)$ returns $concr(pat_b, \theta_1)$ where b's value is found in θ_1. Applying g does not always produce a bit-string but can produce a pattern as variables requesting asymmetric keys are not filled in g. Layer f completes the pattern and applies the encryption using the corresponding public key (and stores the output bit-string in the related memory). The second part f' contains the different public key and decryption oracles, so f' only depends on θ_2.

5. V_1 contains all the verifiers, i.e. verifier for challenge bit b, for fresh signatures and for fresh encryptions. Therefore V_1 only depends on θ_1.

This partition is valid hence theorem 5.1 applies: for any adversary $\mathcal{A}$ against criterion γ, there exist two adversaries $\mathcal{B}$ and $\mathcal{A}'$ such that :

$$\forall \eta, \mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 2.\mathbf{Adv}^{\gamma_2}_{\mathcal{B}}(\eta) + \mathbf{Adv}^{\gamma_1}_{\mathcal{A}'}(\eta)$$

where $\gamma_2 = (\Theta_2, b; f \circ LR^b, f'; v_b)$ and $\gamma_1 = (\Theta_1; F_1; V_1)$.

Criterion γ_2 works as follows: N asymmetric key pairs are randomly sampled in Θ_2 (this is done using the key generation algorithm from $\mathcal{AE}$). Memories used to store the encryptions produced by the left-right oracle and a random bit b are also generated. The adversary has access to oracle $f \circ LR^b$. Function f takes as argument a pattern and returns its encryption using a specified challenge key (the output bit-string is also stored). Thus $f \circ LR^b$ takes as argument a pair of patterns $\langle pat_0, pat_1\rangle$ and returns the encryption of pat_b, this is exactly the left-right encryption oracle of PAT-IND-CCA. Oracle f' contains the decryption oracles and the public keys oracles for all the challenge keys. Finally, the verifier v_b checks that the adversary correctly guessed the value of b. Hence γ_2 is criterion N-PAT-IND-CCA. The hypothesis of safety of $\mathcal{AE}$ for N-PAT-IND-CCA implies that the advantage of $\mathcal{B}$ is negligible.

In criterion γ_1, a challenge bit b, N symmetric keys and N signature key pairs are generated. Oracle F_1 allows access to the different verification keys, to the signature oracle and to the left-right encryption oracle for symmetric keys. Finally, verifier V_1 allows the adversary to win in multiple ways: by guessing the challenge bit b, by forging a fresh signature or by forging a fresh encryption. Hence γ_1 is criterion N-PSS. The hypothesis of safety of $(\mathcal{SE}, \mathcal{SS})$ for N-PSS implies that the advantage of $\mathcal{A}'$ is negligible.

Thus we finally get that for any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ against γ is negligible. So cryptographic library $(\mathcal{AE}, \mathcal{SE}, \mathcal{SS})$ is secure for N-PASS. ■

Full Decomposition of N-PASS

The second step to prove proposition 5.8 consists in showing that safety of $(\mathcal{SE}, \mathcal{SS})$ for N-PSS is implied by safety of $\mathcal{SE}$ for N-PAT-SYM-CPA and safety of $\mathcal{SS}$ for N-UNF. This is formally stated in the next proposition.

Proposition 5.10 *Let N be an integer, $\mathcal{SE}$ be a symmetric encryption scheme and $\mathcal{SS}$ be a digital signature scheme. If $\mathcal{SE}$ is secure for N-PAT-SYM-CPA and $\mathcal{SS}$ is secure for N-UNF, then cryptographic library $(\mathcal{SE}, \mathcal{SS})$ is secure for N-PSS.*

Proof: Let $\mathcal{SE}$ be a symmetric encryption scheme that is secure for N-PAT-SYM-CPA. Let $\mathcal{SS}$ be a digital signature scheme that is secure for N-UNF. Let us consider an adversary $\mathcal{A}$ against N-PSS (designated by γ in the following). Symmetric keys can be used to protect signatures however signature of symmetric keys is forbidden. Therefore when partitioning criterion γ, Θ_1 is related to signature whereas Θ_2 is related to symmetric encryption (as oracle F_1 can only use the output of Θ_1 and oracle F_2 may use the outputs of both Θ_1 and Θ_2). Formally criterion γ can be partitioned in $(\Theta_1, \Theta_2; F_1, F_2; V_1, V_2)$ where:

1. Θ_1 generates the N challenge signature keys. The output of Θ_1 is denoted by θ_1.
2. Θ_2 generates the challenge bit b and the N signature keys. The output of Θ_2 is called θ_2.
3. F_1 contains all the signature oracles and the verification key oracles. Hence F_1 only depends on θ_1.
4. F_2 contains the left-right encryption oracles for symmetric keys. As patterns may ask for the encryption of a signature key, F_2 depends on both θ_1 and θ_2. As there are no oracle in F_2 that only depends on θ_2, F_2 is only constituted of a single part with two layers: $\lambda s.f(g(s, \theta_1), \theta_2)$. The g layer consists in the *concr* function. Formally its argument is a pair of patterns $\langle pat_0, pat_1 \rangle$ and it returns a pair of patterns $\langle concr(pat_0, \theta_1), concr(pat_1, \theta_1) \rangle$. Layer f takes as argument a pair of patterns $\langle pat_0, pat_1 \rangle$ and returns the encryption of the concretization of pat_b. Encryption is done using the corresponding symmetric key and the result of the encryption is also stored (so as the verifier can check that the adversary forged a fresh encryption).
5. V_1 is the verifier related to θ_1, i.e. it checks that the adversary forged a fresh signature.
6. V_2 is the verifier related to θ_2, the adversary can win either by guessing challenge bit b or by forging a fresh symmetric encryption using one of the challenge keys.

This partition is valid thus it is possible to apply the extended version of the partition theorem (theorem 5.2). Then, there exist three adversaries $\mathcal{B}$, $\mathcal{A}'$ and $\mathcal{A}''$, such that:

$$\forall \eta, \mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 2.\mathbf{Adv}^{\gamma_2}_{\mathcal{B}}(\eta) + \mathbf{Adv}^{\gamma_1}_{\mathcal{A}'}(\eta) + \mathbf{Adv}^{\gamma_3}_{\mathcal{A}''}(\eta)$$

where $\gamma_3 = (\Theta_2; f; V_2)$ is the criterion related to V_2, $\gamma_2 = (\Theta_2, b; f \circ LR^b; v_b)$ and $\gamma_1 = (\Theta_1; F_1; V_1)$.

In criterion γ_3, a challenge bit b and N symmetric keys are randomly generated (using the key generation algorithm from $\mathcal{SE}$). Oracle f is the classical left-right encryption oracles for patterns and the adversary can win this game in two ways: by guessing the value of b and by forging a

fresh symmetric encryption. Therefore γ_3 is criterion N-PAT-SYM-CPA. The hypothesis of safety of $\mathcal{SE}$ for N-PAT-SYM-CPA implies that the advantage of $\mathcal{A}''$ is negligible.

Criterion γ_2 works as follows: a challenge bit b' is generated by Θ_2, another challenge bit b is generated. N symmetric keys are also randomly generated using the key generation algorithm from $\mathcal{SE}$. The adversary has to guess the value of challenge bit b. For this purpose he has access to oracle $f \circ LR^b$. Therefore the adversary submits pairs of pairs of patterns of the form:

$$\big\langle \langle pat_0^0, pat_0^1 \rangle, \langle pat_1^0, pat_1^1 \rangle \big\rangle$$

The LR^b layer first selects the pair $\langle pat_b^0, pat_b^1 \rangle$. Then the f layer returns the encryption (after concretization) of $pat_b^{b'}$. Let $\mathcal{C}$ be an adversary against this criterion, it is possible to build an adversary $\mathcal{C}'$ against N-PAT-SYM-CPA/IND that has the same advantage. This adversary $\mathcal{C}'$ is trivially obtained from $\mathcal{C}$ as described below:

Adversary $\mathcal{C}'(\eta)/\mathcal{O}_1, ..., \mathcal{O}_N$:
$b' \xleftarrow{R} [0,1]$
$d := \mathcal{C}/\lambda\big\langle \langle pat_0^0, pat_0^1 \rangle, \langle pat_1^0, pat_1^1 \rangle \big\rangle.\mathcal{O}_1(\langle pat_0^{b'}, pat_1^{b'} \rangle)$
$\quad ...$
$\quad \lambda\big\langle \langle pat_0^0, pat_0^1 \rangle, \langle pat_1^0, pat_1^1 \rangle \big\rangle.\mathcal{O}_N(\langle pat_0^{b'}, pat_1^{b'} \rangle)$
return d

Then the games involving $\mathcal{C}$ against γ_2 and $\mathcal{C}'$ against N-PAT-SYM-CPA/IND are exactly the same. Thus for any adversary $\mathcal{C}$ against γ_2, there exists an adversary $\mathcal{C}'$ against N-PAT-SYM-CPA/IND that has the same advantage. So safety for γ_2 is implied by safety for N-PAT-SYM-CPA/IND. Therefore the hypothesis of safety of $\mathcal{SE}$ for N-PAT-SYM-CPA can be used once more ensuring that the advantage of $\mathcal{B}$ is negligible.

In criterion γ_1, N signature key pairs are generated (using the key generation algorithm from $\mathcal{SS}$). F_1 is composed by the signature oracles related to the different challenge keys. Finally, V_1 allows the adversary to win by forging a fresh signature. Thus γ_1 is exactly criterion N-UNF and as the N-UNF hypothesis has been made for $\mathcal{SS}$, the advantage of $\mathcal{A}'$ is negligible.

Thus we finally get that for any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ against γ is negligible. So cryptographic library $(\mathcal{SE}, \mathcal{SS})$ is secure for N-PSS. ■

Chapter 6

Linking the Computational and Symbolic Worlds

Contents

In this chapter, we state the main result of this thesis: computational soundness of the symbolic model for security protocols. In a first section, we introduce the computational semantics for security protocols and give various simplified models in order to make the main proof easier to understand. Then we state our main result: the probability for a polytime computational adversary to produce a trace that cannot be abstracted on a valid trace in the symbolic model is negligible. This result essentially holds on traces. Thus in the last section, we show how this result can be applied on some classical properties like authentication and how we can adapt the proof of the main result for strong secrecy for nonces and keys.

6.1 Computational Semantics for Security Protocols

Computational semantics are far more realistic than symbolic semantics: as in real life, messages sent over the network are bit-strings and the adversary is modeled as a polynomial-time random Turing machine that can interact with the protocol.

Computational semantics of a protocol depends on a cryptographic library $\mathcal{CL}$ that contains an asymmetric encryption scheme $(\mathcal{KG}^a, \mathcal{E}^a, \mathcal{D}^a)$, a digital signature scheme $(\mathcal{KG}^g, \mathcal{S}, \mathcal{V})$ and a symmetric encryption scheme $(\mathcal{KG}^s, \mathcal{E}^s, \mathcal{D}^s)$. These cryptographic primitives are used to implement the protocol: if an agent has to send message $\{m\}^a_{pk}$, then the bit-string value bs_1 for m is computed, the bit-string value bs_2 for pk is retrieved and the bit-string that is sent over the network is obtained by evaluating $\mathcal{E}^a(bs_1, bs_2)$.

6.1.1 Execution Without Adversary (Passive Adversary Semantics)

In this section, the adversary has no control over the network, he cannot intercept, block, modify or send any message. His only ability is to observe the exchanged messages. Hence this model is referred to as the *Passive Adversary* model (PAM). The semantics for execution of the protocol is the usual semantics when no adversary is present: this is detailed later in this section. After executing the protocol, the resulting *computational trace* is given to the adversary.

Definition 6.1 *A computational trace is a sequence of emissions and receptions of bit-strings. Therefore computational traces are given by the following grammar where bs ranges over bit-strings.*

$$\begin{array}{rcl} ct & ::= & \epsilon \\ & | & \textsc{Send}(bs).ct \\ & | & \textsc{Recv}(bs).ct \end{array}$$

Passive adversary semantics are given for linear protocols. These semantics defines the possible computational traces for a protocol in the PAM. Possible traces for a general protocol $\Pi = (S, IK)$ can be defined as the union of the possible traces of (R, IK) where R ranges over the inter-leavings of S. Let $\Pi = (R, IK)$ be a linear protocol. In a first step, the different atoms that occur in R have to be generated. Then to execute a protocol, an algorithm that produces bit-strings from symbolic messages is described. Finally, the execution algorithm is given and explains precisely the computational semantics.

To describe these different algorithms, let *msgt* be a type that represents the possible nature of a message. An element of *msgt* can be *noncet* for nonces, *apkeyt* for asymmetric public keys, *askeyt* for asymmetric secret keys, *gskeyt* for signature keys, *gvkeyt* for verification keys, *skeyt* for symmetric keys, *aenct* for asymmetric encryptions, *gsigt* for signatures, *senct* for symmetric encryptions and *pairt* for pairs. We assume the existence of some simple algorithms (which can easily be implemented with a polynomial time complexity in η):

1. *atoms*: this algorithm takes as argument a role R and returns a list of nonces and keys. A nonce N is in $atoms(R)$ if N occurs in R, a public key pk is in $atoms(R)$ if pk or pk^{-1} occurs in R, a symmetric key k is in $atoms(R)$ if k appears in R and finally a signature verification key vk is in $atoms(R)$ if vk or vk^{-1} appears in R. Hence $atoms(R)$ does not contain any secret key, it only contains public keys, symmetric keys, verification keys and nonces.

2. *type*: this algorithm takes as argument a symbolic (closed) message and returns a *msgt* describing the type of m.

3. *inv*: this algorithm takes as argument a symbolic atom which has to be a key and returns the inverse key (which is also a symbolic atom).

Generating the Atoms

Generating the different atoms of a protocol is done by the init algorithm. This algorithm takes as argument a security parameter η and a role R. It returns a computational substitution that links atoms of R to freshly generated values. The algorithms from the cryptographic library are used to generate the different values. For nonces, random bit-strings of size η are generated.

init(η, R) :
  $\theta := []$
  for a **in** $atoms(R)$
    if $type(a) = noncet$ **then**
      $a\theta \xleftarrow{R} [0,1]^\eta$
    else if $type(a) = apkeyt$ **then**

$(a\theta, a^{-1}\theta) := \mathcal{KG}^a(\eta)$
else if $type(a) = gvkeyt$ **then**
$(a\theta, a^{-1}\theta) := \mathcal{KG}^g(\eta)$
else if $type(a) = skeyt$ **then**
$a\theta := \mathcal{KG}^s(\eta)$
endif
endfor
return θ

The init algorithm is close to a challenge generator as described in our criterion formalism: it generates some values, the adversary has access to some computations performed on these values and has to produce an output according to these values.

Forging and Parsing Messages

Execution of the protocol heavily uses two auxiliary algorithms. The *concr* algorithm takes as argument a symbolic message m and a computational substitution θ. It randomly generates a possible bit-string value for m. For this purpose, it uses the different encryption and signature algorithms from the cryptographic library $\mathcal{CL}$. This algorithm is described recursively on the structure of its argument m. This algorithm can fail and produce an error in two cases: if a variable or an atom is not defined by θ and if a cryptographic primitive that it uses produces an error.

concr(t, θ) :
match t **with**
$[x]$ — Variable
return $x\theta$
$[a]$ — Atoms (Keys, Nonces, Identities)
return $a\theta$
$[enc_a(t_1, t_2)]$ — Asymmetric Encryption
return $\mathcal{E}^a(concr(t_1, \theta), concr(t_2, \theta))$
$[enc_s(t_1, t_2)]$ — Symmetric Encryption
return $\mathcal{E}^s(concr(t_1, \theta), concr(t_2, \theta))$
$[enc_g(t_1, t_2)]$ — Digital Signature
return $\mathcal{S}(concr(t_1, \theta), concr(t_2, \theta))$
$[pair(t_1, t_2)]$ — Pairing
return $concr(t_1, \theta) \cdot concr(t_2, \theta)$

The second auxiliary algorithm is the *parse* algorithm which achieves parsing. This algorithm takes as arguments a bit-string bs, a message term t and a computational substitution θ. It tries to parse bit-string bs using as prototype term t. For this purpose it uses the decryption algorithms provided by the cryptographic library. Parsing may fail and produce a parse error. This is the case when t is a signature as signature verification is achieved using the VERI() statement. An error can also occur if a variable or an atom is already defined in θ with a different value. Finally, the decryption algorithms can produce some errors.

parse(bs, t, θ) :
match t **with**
$[x]$ — Variable or Atom
if $x \in sup(\theta)$ **then**
if $x\theta = bs$ **then return** θ
else raise parse-error
else $x\theta := bs$

$[enc_a(t_1, t_2)]$	Asymmetric Encryption
$\quad parse\big(\mathcal{D}^a(bs, concr(t_2, \theta)), t_1, \theta\big)$	
$[enc_s(t_1, t_2)]$	Symmetric Encryption
$\quad parse\big(\mathcal{D}^s(bs, concr(t_2, \theta)), t_1, \theta\big)$	
$[enc_g(t_1, t_2)]$	Digital Signature
$\quad$ **raise** parse-error	
$[pair(t_1, t_2)]$	Pairing
$\quad parse(pr_1(bs), t_1, \theta)$	
$\quad parse(pr_2(bs), t_2, \theta)$	

Executing Protocols

Possible computational traces are given by the possible executions of a (random) algorithm. This algorithm denoted by $PAMexec$ takes as argument a security parameter η, a computational substitution θ and a protocol (R, IK) (note that IK is not used by this algorithm). Variable *trace* is used to store the trace and its value is returned at the end of the execution. Variable *bs* stores the last bit-string that has been sent over the network.

PAMexec$(\eta, \theta, (inst_1 \dots inst_\ell, IK))$:	
$trace := []$	Initialisation
for i **from** 1 **to** ℓ	
$\quad$ **match** $inst_i$ **with**	
$\quad\quad [[\textsc{Veri}(t_1, t_2, t_3)]]$	Signature verification
$\quad\quad\quad$ **if** $\mathcal{V}(concr(t_1, \theta), concr(t_2, \theta), concr(inv(t_3), \theta))) = 0$	
$\quad\quad\quad$ **then raise** test-failed	
$\quad\quad [\textsc{Recv}(t)]$	Message reception
$\quad\quad\quad \theta := parse(bs, t, \theta)$	
$\quad\quad\quad trace := trace :: \textsc{Recv}(bs)$	
$\quad\quad [\textsc{Send}(t)]$	Message emission
$\quad\quad\quad bs := concr(t, \theta)$	
$\quad\quad\quad trace := trace :: \textsc{Send}(bs)$	
$\quad$ **endmatch**	
endfor	
return $(trace, \theta)$	

This algorithm outputs the *trace* that has been produced by the execution of the protocol. It also outputs the updated version of substitution θ (the updates take place during parsing).

Using the $PAMexec$ algorithm, it is now easy to define possible traces in the PAM model.

Definition 6.2 (PAMtraces) *The set $PAMtraces$ of possibles traces in the PAM model contains all possible outputs for $PAMexec$ for all possible values of θ generated by init.*

$$PAMtraces(R, IK) = \big\{t \big| \theta := init(\eta, R)\ ,\ (t, \_) := PAMexec(\eta, \theta, (R, IK))\big\}$$

6.1.2 The Computational Model (CM) of Adversaries

We want to let the adversary have arbitrary control over the network, as in the symbolic model, and hence we eliminate the network. Moreover, the adversary drives the computation by sending messages to the other players and receiving messages from them. In the *computational model*, the messages that are exchanged are bit-strings (and depend on the security parameter η). We define three separate models: the computational model is the more general one, the adversary exchanges bit-strings with the protocol. The second model is less realistic, the adversary also uses bit-strings

to communicate with the protocol but it also has to provide justifications for its bit-string: the adversary has to specify how the bit-string can be parsed. In the third model, we put a restriction on the general model: variables can only be assigned to with atoms (keys, nonces or identities). Hence it is forbidden for a protocol to receive a message in a variable x that can be a composed message like a pair, an encryption or a signature. In particular, message forwarding is impossible in this model. Proofs in the last two models are easier to design but the drawback is that it lacks the generality of the first model. Execution of the protocol is performed in the same way for all the models.

Executing the Protocol

The $CMexec$ algorithm given here is close to the $PAMexec$ algorithm that was detailed earlier. The distinction is that an adversary $\mathcal{A}$ interacts actively with the protocol. This algorithm takes four arguments: the first one is a security parameter η, the second one is a computational adversary $\mathcal{A}$, the third one is a computational substitution θ and the last one is a protocol Π. It produces a trace that is possible for protocol Π. The variable *trace* is used as in $PAMexec$. The variable *mem* stores the memory of the adversary. Whenever the adversary is called, it takes *mem* as argument and returns a new value for *mem*, this means that the adversary can modify its memory. When a receive statement $\textsc{Recv}(t)$ is reached, the adversary is called and output a bit-string *bs*. This bit-string is parsed using t as prototype and the protocol continues. When a send statement $\textsc{Send}(t)$ is reached, a bit-string value for t is generated using the *concr* algorithm. This value is appended to the trace and to the memory of the adversary. The initial memory of the adversary contains the values for the parameters that appear in IK. The *cut* algorithm is used here, it takes as arguments a computational substitution θ and a list l and returns the restriction of θ to elements of l.

CMExec$(\eta, \mathcal{A}, \theta, (inst_1 \cdots inst_\ell, IK))$:
$trace := []$; $mem := cut(\theta, IK)$; — Initialisation
for i **from** 1 **to** ℓ
  match $inst_i$ **with**
    $[[\textsc{Veri}(t_1, t_2, t_3)]]$ — Signature verification
      if $\mathcal{V}(concr(t_1, \theta), concr(t_2, \theta), concr(inv(t_3), \theta))) = 0$
      then raise test-failed
    $[\textsc{Recv}(t)]$ — Message reception
      $(bs, mem) := \mathcal{A}(mem)$
      $\theta := parse(bs, t, \theta)$
      $trace := trace :: \textsc{Recv}(bs)$
    $[\textsc{Send}(t)]$ — Message emission
      $bs := concr(t, \theta)$
      $trace := trace :: \textsc{Send}(bs)$
      $mem := mem :: bs$
  endmatch
endfor
return $(trace, \theta, mem)$

This algorithm outputs the *trace* that has been produced by the execution of the protocol. It also outputs the updated version of substitution θ (the updates take place during parsing) and the final memory of the adversary. This memory is useful when defining properties of a protocol: the adversary has to win a game thus it is called one more time with its final memory and outputs its final guess for its challenge.

The Simplified Computational Models

The main advantage of the computational model is its realism. However, the proof relating the computational and the symbolic model (see section 6.2) is hard to achieve in this setting. The idea of the proof is to build an adversary $\mathcal{B}$ against cryptographic primitives using an adversary $\mathcal{A}$ against the protocol. Hence adversary $\mathcal{B}$ has to correctly simulate the execution of the protocol, i.e. the $CMExec$ algorithm, and $\mathcal{B}$ wins his challenge whenever a non-deducible message (for Dolev-Yao) is produced. The main difficulty when describing $\mathcal{B}$ is to correctly parse messages: if $\mathcal{B}$ receives a bit-string bs, parses it according to the protocol and later in the protocol the same bit-string is parsed as a non-deducible message, then it is possible that $\mathcal{B}$ cannot win its challenge anymore. To exemplify this, let us consider the following protocol:

$$\textsc{Recv}(x).\textsc{Send}(\{x\}_k).\textsc{Send}(N).\textsc{Recv}(\{N\}_k)$$

A possible execution is that the adversary sends the bit-string value bs_N associated to N, however, this bit-string is parsed as x, then the adversary receives two bit-strings bs_1 and bs_2 and outputs bs_1 as the bit-string necessary to complete the protocol. At this point, $\mathcal{B}$ knows that bs_1 can be parsed as $\{N\}_k$ hence the first bit-string can be parsed as N. Therefore the related symbolic trace is:

$$\textsc{Recv}(N).\textsc{Send}(\{N\}_k).\textsc{Send}(N).\textsc{Recv}(\{N\}_k)$$

This trace is not possible but the problem for $\mathcal{B}$ is that it only understands that the trace is impossible when receiving the last bit-string. At this point, it is too late for $\mathcal{B}$ to break cryptographic schemes because he already had to output the value of N. More details on this sort of problems appear in the next section. However there are two simple solutions so that this problem cannot occur anymore. For both solutions, we first ask computational messages to be strongly typed (i.e. it is possible from bs to know its type using some computational function).

- The first possibility is to only accept variables for nonces, keys and identities in the protocol and to forbid message forwarding. In this situation, when receiving a new bit-string bs, it is possible to test its equality with all the atoms in the protocol that have the same type. This is called the Atomic Computational Model (ACM) in the following.
- Another possibility is to ask the adversary $\mathcal{A}$ playing against the protocol to justify any bit-string he has produced. Whenever he sends a bit-string, $\mathcal{A}$ has to send the formal version of this bit-string and all the necessary atoms (and random coins) such that it is possible to verify that $\mathcal{A}$ does not lie. This model is called the Justified Computational Model (JCM) in the following.

ACM The main advantage of ACM is that the semantics of protocols are not changed compared to the general computational model, the $CMExec$ algorithm does not have to be modified. The drawback is that message forwarding is not allowed anymore, hence our results cannot be applied to some protocols. For example let us detail the case of the symmetric version of Needham-Schroeder protocol (this protocol is described in [NS78], an attack against authentication is given in [DS81]).

$$\begin{array}{lllll} 1 & A \to S & : & A, B, Na \\ 2 & S \to A & : & \{Na, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas} \\ 3 & A \to B & : & \{Kab, A\}_{Kbs} \\ 4 & B \to A & : & \{Nb\}_{Kab} \\ 5 & A \to B & : & \{dec(Nb)\}_{Kab} \end{array}$$

In the description above, pairing is not explicitly detailed in order to simplify notations. The dec operator used here is decrementing. This operator is not present in our protocol syntax, however

it can be simulated using encryption by a key Km that every agent knows. The final line of the protocol becomes:

$$5 \quad A \to B \quad : \quad \{\{Nb\}_{Km}\}_{Kab}$$

This last operation is not important for our example thus we omit the last line of the protocol in the following and consider only lines 1 to 3. The important point is between line 2 and line 3. Agent A receives a message that contains a sub-message $\{Kab, A\}_{Kbs}$ that it cannot test (as key Kbs is shared between B and S), hence the correct way to describe this protocol in our syntax is given by the following in the case of a single session involving agents A, B and S.

$$\begin{array}{ll} 1 & \textsc{Send}(A, B, Na).\textsc{Recv}(A, B, x). \\ 2 & \textsc{Send}(\{x, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas}).\textsc{Recv}(\{Na, B, Kab, x\}_{Kas}). \\ 3 & \textsc{Send}(x).\textsc{Recv}(\{Kab, A\}_{Kbs}) \end{array}$$

In this description, x is a variable which is assigned to with a composed message (an encryption in this case). Therefore it is impossible to express this protocol without message forwarding.

Although the ACM is as realistic as the general model, there are some protocols that cannot be expressed in this model. The main interest of this model is that it is possible to test every received bit-string, even when authorizing key sending. Thus the computational soundness proof is less technical in this model.

JCM The JCM model is another way to simplify the general computational model. The idea is that whenever the adversary sends a bit-string, he has to justify its content: he has to give the symbolic message related to this bit-string and also has to give enough information so that it is possible to check that he does not lie. In the following, we focus on the ACM, hence we do not give formal definitions for the JCM. Compared to the ACM, the JCM can be applied to all the protocols on which the general computational model applies, there are no restrictions like "no message forwarding". However this model cannot be implemented in practice as we do not want the agents involved in a protocol to have to justify for their messages: in this case, honest agents would have to reveal their secret keys.

6.1.3 Protocol Properties

Properties in the symbolic model were defined using possible (symbolic) traces for the protocol. There is an attack against these properties as soon as one of this possible traces leads to an attack. However in the computational setting, it is possible that a trace leads to an attack but the protocol is safe for the property. This is the case for brute-force attacks or when an adversary tries to guess the value of a secret nonce for example. This is why a protocol is declared as not safe for the property only if there exists an adversary that can create an attack with a non-negligible probability (in η).

We present here how simple properties can be described in the computational model. This is done for secrecy and for trace properties. These definitions hold for all the different adversary models described previously, the only difference is the execution algorithm that is called. Relating these properties to properties in the symbolic model and security of the cryptographic library is a difficult task, the idea of a simple proof for this result was first given in the work of Bogdan Warinschi [War03] for the Needham-Schroeder-Lowe protocol. It was later generalized to other protocols in [MW04c]. An adaptation of this proof is detailed in section 6.2 and is used in section 6.3.

Secrecy Properties: SecNonce

This SecNonce property has been introduced in [CW05]. It is a way to define strong secrecy for a nonce during a protocol execution. Let $\Pi = (R, IK)$ be a linear protocol and N be a nonce that

occurs in Π. The challenge is the following: a random bit b is generated and two bit-strings of length η are generated. These bit-strings are denoted by bs_0 and bs_1. The exec algorithm is called and the value used for N is bs_0 if b equals 0 and bs_1 if b equals 1. Finally, the adversary is given bs_0 and bs_1 and has to output the value of b.

Formally, the advantage of the adversary in the passive setting is defined by:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{PAM-SN(\Pi,N)}(\eta) \quad = Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta) := PAMexec(\eta, \theta, \Pi) \\ & bs \xleftarrow{R} [0,1]^\eta \\ & \mathcal{A}(t, cut(\theta, IK), bs, N\theta) = 1] \\ -Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta) := PAMexec(\eta, \theta, \Pi) \\ & bs \xleftarrow{R} [0,1]^\eta \\ & \mathcal{A}(t, cut(\theta, IK), N\theta, bs) = 1]\end{aligned}$$

In the first case, the adversary has output 1 and it was given a pair containing a random bit-string and the value used for N so the adversary solved his challenge whereas in the second case, it has output 1 and was given a pair containing the value used for N first and in second a random bit-string, therefore the adversary failed in solving his challenge. The adversary has access to the computational trace produced by $PAMexec$ and to the bit-string values used for elements of his initial knowledge IK. The definition of advantage for the CM is exactly similar. The only difference is that the exec procedure outputs the memory of the adversary. When asked the value of bit b, the adversary has access to this memory.

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{CM-SN(\Pi,N)}(\eta) \quad = Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}, \theta, \Pi) \\ & bs \xleftarrow{R} [0,1]^\eta \\ & \mathcal{A}(mem, bs, N\theta) = 1] \\ -Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}, \theta, \Pi) \\ & bs \xleftarrow{R} [0,1]^\eta \\ & \mathcal{A}(mem, N\theta, bs) = 1]\end{aligned}$$

Definition 6.3 (SecNonce) *Protocol* Π *verifies SecNonce for nonce* N *if the advantage of any adversary* $\mathcal{A}$ *is negligible in* η*. This defines three flavors of SecNonce: SecNonce in the passive adversary model, SecNonce in the Computational Model and SecNonce in the Simplified Computational Models.*

There are some trivial implications between the different SecNonce properties. The most important one is that if a protocol Π verifies SecNonce for N in the CM, the it also verifies SecNonce for N in the PAM.

The SecNonce property can be described using our criterion formalism. However we stick with this definition as our semantics are not given in terms of oracles.

Trace Properties

As in the case of the symbolic model, trace properties are easy to formulate here. The computational and symbolic versions of these properties are usually quite close.

Authentication Authentication properties are not directly related to the knowledge of the adversary: the adversary does not have to output anything at the end of its execution. However in authentication, we want to check that some actions of the protocol must have occurred before.

Let us now illustrate how authentication can be modeled in the computational setting. With this aim in view, we consider the case of non-injective agreement of role B with role A on message m. We have to consider all the linear protocols Π that represent possible interleavings of A and B. Then in these protocols, let us consider that x and y (which can be either variables or atoms) represent the idea that A and B respectively have of message m. Then authentication holds if for any adversary $\mathcal{A}$, the probability to produce a trace where x and y have different bit-string representations is negligible. This probability p is properly defined in the case of the CM by:

$$\begin{aligned} p(\eta) \quad = Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}, \theta, \Pi) \\ & x\theta \neq y\theta] \end{aligned}$$

The case of simplified computational models is not detailed here as the exec algorithms are the same as in the general computation model. Concerning the passive setting, adversaries cannot attack authentication, thus we are not interested by the PAM. Other flavors of authentication can be described in a similar way.

Weak Secrecy Weak secrecy is a version of secrecy weaker than SecNonce. Whereas in SecNone, the adversary has to distinguish the use of bs_0 from the use of bs_1 for nonce N, in secrecy the adversary has to output the bit-string value used for N. The challenge that defines weak secrecy of nonce N in protocol $\Pi = (R, IK)$ is the following. Initial values are generated using $init$, then the protocol is executed according to the adversary model. Finally, the adversary has to produce the bit-string that was used for nonce N. The advantage in the case of the PAM is defined by:

$$\begin{aligned} \mathbf{Adv}_{\mathcal{A}}^{PAM-WS(\Pi,N)}(\eta) \quad = Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta) := PAMexec(\eta, \theta, \Pi) \\ & bs := \mathcal{A}(t, cut(\theta, IK)) \\ & bs = N\theta] \end{aligned}$$

In the case of the CM, the adversary has access to the last memory output when playing against the protocol.

$$\begin{aligned} \mathbf{Adv}_{\mathcal{A}}^{CM-WS(\Pi,N)}(\eta) \quad = Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}, \theta, \Pi) \\ & bs := \mathcal{A}(mem) \\ & bs = N\theta] \end{aligned}$$

Definition 6.4 (Weak Secrecy) *Protocol Π verifies Weak Secrecy for nonce N if the advantage of any adversary $\mathcal{A}$ is negligible in η. As for SecNonce, this defines three flavors of Weak Secrecy: Weak Secrecy in the passive adversary model, Weak Secrecy in the Computational Model and Weak Secrecy in the Simplified Computational Models.*

There are also some easy implications between these weak secrecy notions.

An interesting point is that this secrecy property can be considered as a trace property. For this purpose, the idea is to add a statement $\textsc{Recv}(N)$ at the end of R. However this is not sufficient as an adversary might be able to deduce the value of N without being able to complete the protocol execution (e.g. the adversary is not able to forge the signature of some message). Hence the $\textsc{Recv}(N)$ statement has to be placed after each possible statements. If R is composed of statements $inst_1 \cdots inst_\ell$, then weak secrecy of N is equivalent to the ability of an adversary to complete any

of the protocols $(\text{RECV}(N), IK)$, $(inst_1.\text{RECV}(N), IK)$ to $(inst_1 \cdots inst_\ell.\text{RECV}(N), IK)$. Then weak secrecy can be verified only by observing the trace produced by the execution. For this reason, weak secrecy can be considered as a trace property. Note that defining weak secrecy as a trace property has the advantage that this definition can also be used for any composed messages including for example keys or digital signatures.

6.2 Linking Symbolic and Computational Traces

In this section, we aim at proving that under some restrictions on cryptographic primitives and on security protocols, the symbolic model is a safe abstraction of the computational model. The main result is that the probability that a computational adversary outputs a trace that cannot be abstracted to a correct symbolic trace is negligible. Thus the symbolic model correctly abstracts the computational model with overwhelming probability.

The proof presented here originates from the work of Bogdan Warinschi. It was first stated only in the case of the Needham-Schroeder-Lowe protocol in [War03]. The next year, Micciancio and Warinschi extended this result to the case of general protocols [MW04c]. However severe restrictions prevent these results from applying to real-life protocols: in particular, message forwarding is not allowed, the only cryptographic primitive is asymmetric encryption and secret keys cannot be sent. Another restriction is that only trace properties are considered, allowing to describe authentication or weak versions of secrecy. In [CW05], Cortier and Warinschi generalized the previous results by considering protocols with asymmetric encryption and digital signature. Emission of secret keys is still forbidden but message forwarding is allowed. Moreover, although the main result holds on traces, the proof is adapted to describe strong secrecy of nonces (using the SecNonce property). The proof of the main theorems of these three papers have a common structure which is also used here.

This section is structured as follows. In the first part, the restrictions over protocols and cryptographic primitives are made explicit. Then the second part formulates the main result and its proof in a simple case: the ACM model where only asymmetric encryption is considered. The following part generalizes this result to ACM protocols that also involves symmetric encryption and digital signature. Part four describes how the proof can be extended to bypass the ACM limitation and finally the last part deals with possible extensions.

6.2.1 Hypotheses

Different restrictions are needed in order to prove the main result of this section. These restrictions can be split in two distinct categories: restrictions over protocols only depend on the protocol that is considered, such restrictions are expressed in the symbolic setting; restrictions over cryptographic primitives, such restrictions ensure that the different schemes used to implement the protocol are safe, hence they are formulated in the computational setting.

Hypotheses over Protocols, Acyclicity

First, let us describe the restrictions on protocols. Let Π be a protocol that is composed of a linear scenario R and an initial knowledge IK. This protocol is said to be *valid* if it satisfies the following conditions:

1. The protocol is executable.
2. Any public key from R appears in IK.
3. Any agent identity from R appears in IK.
4. Correct traces do not contain any encryption cycle. Secret keys cannot appear encrypted using symmetric cryptography.

The first requirement is easily understandable: only executable protocols are of interest. Other protocols cannot be used properly. The second and third requirement indicates that public keys and agent identities are known by the adversary. The security of a protocol should not rely on the secret of such atoms hence this restriction is always met in practice. The last restriction is more technical, it already appeared in [AR00].

Key Cycles Let us detail a little more the last restriction. As stated earlier, this restriction comes from a real vulnerability of IND-CCA algorithms. Therefore this restriction is necessary when using such encryption schemes (another solution would be to consider KDM security).

We propose two versions of this acyclicity restriction. The first version is harder to satisfy but easier to formulate and understand. For the two versions, we suppose that R contains α public keys and β symmetric keys. We also assume that there exist a total order among public keys from R $pk_1 \prec pk_2 \prec ... \prec pk_\alpha$ and a total order among symmetric keys $k_1 \prec k_2 \prec ... \prec k_\beta$. Let m be a message contained in a correct trace t of the protocol, $SecK$ be the set of secret and symmetric keys that are not deducible from IK and the messages in t. This set contains all the keys whose secrecy is preserved during the protocol execution.

The strict version of the requirement consists in saying that for every occurrence of a secret key sk, the most external encryption that protects this occurrence of sk is performed using a public key pk' such that $pk' \prec sk^{-1}$ (the encryption using pk' is said to protect the occurence of sk if pk'^{-1} is in $SecK$). Formally, if atom pk_i^{-1} from $SecK$ appears in message m at a position p. Then let p' be the smallest prefix of p such that $m_{|p'}$ is an encryption using a public pk_j which inverse is in $SecK$. The restriction asks j to be (strictly) lower than i. The same restriction holds for symmetric keys. Let k_i be a symmetric key from $SecK$. If k_i appears in m, then let k_j be the most external key from m that protects k_i such that k_j is in $SecK$. Then either k_j is an asymmetric encryption key and there are no restrictions or k_j is a symmetric key and j has to be (strictly) lower than i.

Example 6.1 *Let us exemplify our strict acyclicity requirement on some simple examples. For this purpose, we consider the following messages in the case where all the symmetric keys used by the protocol appear in SecK.*

$$m_1 = \{k_1\}_{k_1} \quad m_2 = \langle \{k_1\}_{k_2}, \{k_2\}_{k_1} \rangle \quad m_3 = \left\{\{k_2\}_{k_2}\right\}_{k_1} \quad m_4 = \left\{\left\{\{k_2\}_{k_2}\right\}_{k_1}\right\}_{k_2}$$

Message m_1 contains the most simple example of key cycle. In message m_2, key k_1 is protected by key k_2 and $k_1 \prec k_2$, hence this is also an encryption cycle. Message m_3 contains a cycle using key k_2. However this cycle is "protected" by key k_1. Therefore, message m_3 satisfies the strict requirement of acyclicity. Finally, in message m_4, the most external layer protecting k_2 is k_2 itself. Therefore, this message contains a cycle.

Let us now explain how the acyclicity restriction can be relaxed. The idea is that the condition we asked on the most external protecting key can be asked to hold on one of the protecting keys. This can be formalized as follows: If atom pk_i^{-1} from $SecK$ appears in message m at position p, then there exists a prefix p' of p such that p' is a encryption with pk_j such that:

1. pk_j^{-1} is in $SecK$ hence pk_j "protects" pk_i.
2. j is strictly lower than i

With the first notion, message $m = \{\{k_2\}_{k_1}\}_{k_2}$ is a cycle whereas it is not the case when using the second notion. Note that message m can be concretized using the classical pattern encryption oracles: first we ask for encryption of pattern [2] using key k_1. The resulting bit-string is bs, then we submit bs to the encryption oracle which uses k_2. Hence we obtain a valid concretization of message m.

The interested reader should consult the key cycle chapter of [Jan06] where further details are available.

Hypotheses on Cryptographic Primitives

The computational semantics of a security protocol depends on the cryptographic library $\mathcal{CL}$ that is used to implement the protocol. This cryptographic library has to implement an asymmetric encryption scheme $\mathcal{AE}$, a symmetric encryption scheme $\mathcal{SE}$ and a signature scheme $\mathcal{SS}$. Such a cryptographic library is said to be *safe* if all the primitives contained in the library are safe. Thus $\mathcal{CL}$ is safe if the following requirements are met:

1. The asymmetric encryption scheme $\mathcal{AE}$ is secure against IND-CCA.
2. The symmetric encryption scheme $\mathcal{SE}$ is secure against SYM-CPA.
3. The digital signature scheme $\mathcal{SS}$ is secure against UNF.

Using IND-CPA instead of IND-CCA would not be adequate. Let us consider an IND-CPA encryption scheme $\mathcal{AE}$ composed of key generation algorithm $\mathcal{KG}$, encryption algorithm $\mathcal{E}$ and decryption algorithm $\mathcal{D}$. Then it is possible to build another asymmetric encryption scheme $\mathcal{AE}'$ such that $\mathcal{AE}'$ is also secure against IND-CPA and for any public key pk and any bit-strings bs and bs' of length η,

$$\mathcal{E}'(bs.bs', pk) = \mathcal{E}(bs, pk).\mathcal{E}(bs', pk)$$

Then let us consider the protocol composed of the following role R and the initial knowledge IK that contains pk.

$$R = \textsc{Send}(\{\langle N, N\rangle\}_{pk}).\textsc{Recv}(\{N\}_{pk})$$

Then it is not possible to execute the two actions of R in the symbolic world. However, in the computational setting, the second message can be deduced from the first one. Thus there exists an adversary that produces an incorrect trace with probability 1. For this reason, IND-CPA is not sufficient and we ask the encryption scheme to be resistant against IND-CCA.

6.2.2 Considering only Asymmetric Encryption in the ACM

In this section, we consider protocols that only involve asymmetric encryption. Hence the proof is easier to understand than the proof in the general case. Moreover as we want the proof to be as simple as possible, the computational model used here is the ACM and we use the strict restriction over cycles.

A symbolic trace t_f may have multiple possible concretizations for a computational substitution θ. This multiplicity can occur because of the non-determinism of encryption or because θ does not define every atom from t_f. Therefore, given a symbolic trace t_f, we denote by $Concr(t_f, \theta)$ the set of computational traces obtained by applying $concr(\cdot, \theta')$ on each symbolic message of t_f where θ' is a mapping from bit-strings to symbolic messages that extends θ. Then the soundness theorem is that the probability for an adversary to produce a trace that is not a possible concretization of a possible symbolic trace is negligible.

Theorem 6.1 *Let Π be a valid protocol that uses an IND-CCA asymmetric encryption scheme $\mathcal{AE}$. Let $\mathcal{A}$ be an adversary. Then, the following probability is negligible as a function of η:*

$$Pr[\nexists t_f \in \mathsf{Traces}(\Pi) \cdot CMExec(\mathcal{A}, \Pi, \theta) \in Concr(t_f, \theta)]$$

Where $CMExec(\mathcal{A}, \Pi, \theta)$ gives the computational trace produced by an active adversary $\mathcal{A}$ against protocol Π, $\mathsf{Traces}(\Pi)$ the possible symbolic traces for Π and $Concr(t_f, \theta)$ the possible concretizations of symbolic trace t_f.

Protocol Π is composed of a role R and an initial knowledge IK. Let N denote the number of different atoms that occur in Π. Then the number of asymmetric keys used by Π is lower than N, the number of different nonces is also lower than N. This bound is not tight however this is sufficient to achieve the proof.

The proof of this theorem is by reduction to the security of the underlying cryptographic schemes. That is, let $\mathcal{A}$ be an adversary interacting with the protocol, we build an adversary $\mathcal{B}$ playing against the asymmetric encryption scheme $\mathcal{AE}$, more precisely against a N-PAT-IND-CCA criterion such that the above probability is bounded by the advantage of $\mathcal{B}$. As the encryption scheme $\mathcal{AE}$ is supposed to satisfy IND-CCA, this advantage is negligible (this is immediate when applying proposition 5.4). This proof is an adaptation from the proofs that appear in [War03, MW04c]. However, these proofs only worked for protocols where secret keys cannot be sent in messages. This hypothesis is not required in this section.

The Adversary $\mathcal{B}$

Let us now explain the construction of adversary $\mathcal{B}$. As this adversary plays against the N-PAT-IND-CCA criterion, so he has access to the following oracles for each key:

1. A left-right pattern encryption oracle
2. A public key oracle
3. A decryption oracle

Intuitively the idea underlying the construction of $\mathcal{B}$ is as follows. First, the adversary $\mathcal{B}$ randomly guesses which keys are going to remain secret and which keys are not. As the number of keys is bounded by N, the probability that $\mathcal{B}$ correctly guesses the set CK of secret keys is 2^{-N} and is not negligible. Keys whose inverses are in $SecK$ correspond to challenge keys and the oracles from N-PAT-IND-CCA are used on these keys.

Then $\mathcal{B}$ uses the protocol adversary $\mathcal{A}$ as a subroutine. To do so, $\mathcal{B}$ has to answer $\mathcal{A}$'s queries to the protocol. Since $\mathcal{B}$ does not have the challenge keys to answer these queries, he uses the different oracles given by the N-PAT-IND-CCA criterion. While doing so, $\mathcal{B}$ checks that the computational trace in construction corresponds to a symbolic trace. When this is not the case, $\mathcal{B}$ analyzes the produced messages and answers one of the challenges of the N-PAT-IND-CCA criterion, i.e., guesses the bit b. We consider that $\mathcal{B}$ also outputs a final status. The possible final status are:

1. **DY** means that $\mathcal{B}$ managed to execute $\mathcal{A}$ confronted to the protocol, and that $\mathcal{A}$ has produced a correct symbolic trace. In this situation, $\mathcal{B}$ tries to randomly guess the challenge bit.
2. **NDY** means that $\mathcal{A}$ has produced an incorrect symbolic trace and $\mathcal{B}$ has managed to use that in order to deduce the value of bit b.
3. **Nonce Collision** means that $\mathcal{B}$ found a collision between two nonces. The probability of this event is proved to be negligible later in this document.

State of $\mathcal{B}$ Besides the list of instructions of the protocol, $\mathcal{B}$ uses two mappings and a symbolic trace:

1. θ is a mapping that associates bit-strings with symbolic atoms. Thus, θ is a partial concretization relation from symbolic terms to computational values. At the beginning, $\mathcal{B}$ randomly generates values for nonces, public keys that are not challenge keys and agent identities and stores them in θ. This initial value of θ, completed with the public keys from N-PAT-IND-CCA, is available to $\mathcal{A}$ exactly as in $Exec$ (i.e. $\mathcal{A}$ can observe the value of messages from his initial knowledge).
2. σ is a mapping from variables to symbolic messages. This is used in order to construct the symbolic trace and check whether it is valid or not.
3. t_f is the valid symbolic trace whose concretization corresponds to the computational trace produced so far.

4. For each challenge key, $\mathcal{B}$ stores the list of oracle requests and the returned values.

When $\mathcal{B}$ executes a send statement, θ is used to generate the bit-string corresponding to the symbolic message. When $\mathcal{B}$ executes a receive statement, the bit-string returned by $\mathcal{A}$ is parsed according to θ. If a new variable is assigned to during this statement, θ and σ are updated. Moreover, as we consider the ACM, this variable is assigned an atom. Hence $\mathcal{B}$ can compare the bit-string bs corresponding to the variable y with all the different bit-strings that are stored in θ. If bs already occurs in θ where it is linked to an atom a, then $\mathcal{B}$ stores in σ that y is linked to a. During the execution of each statement, t_f is updated by adding a new element to the trace. This element corresponds to the current statement after application of substitution σ. Thus, t_f always stores a symbolic trace that correctly abstracts the computational trace produced by the interaction of $\mathcal{A}$ with the protocol. Finally, the list of oracle queries is useful because $\mathcal{B}$ cannot ask decryption of outputs of the left-right encryption oracle. Thus if $\mathcal{B}$ produces a bit-string bs using this oracle and gives it to $\mathcal{A}$, suppose that $\mathcal{A}$ sends bs back to $\mathcal{B}$ then $\mathcal{B}$ cannot parse it using his decryption oracle. However $\mathcal{B}$ knows what is in bs as he has forged it with his oracles.

Choosing challenge nonces and challenge keys Adversary $\mathcal{B}$ also randomly chooses a subset of the keys CK used in the protocol. These keys are called challenge keys and $\mathcal{B}$ uses its oracles to compute messages related to these keys.

Let $N_1, \cdots, N_n$ be the nonces in $atom(\Pi) \setminus IK$. These nonces are originally not known by the adversary. The idea is that if the adversary manages to produce an incorrect symbolic trace, then either he guessed the value of one of these nonces (and this nonce always circulated protected by a key whose inverse is not deducible) or he guessed the value of one of the secret keys. Before starting the execution of $\mathcal{A}$ and the protocol, $\mathcal{B}$ randomly chooses one of these nonces. This nonce Ni is called the challenge nonce and $\mathcal{B}$'s guess is that he will get the value of Ni if $\mathcal{A}$ produces an incorrect trace. Thus adversary $\mathcal{B}$ generates two random bit-string values Ni_0 and Ni_1 for nonce Ni. These values are used to construct the messages that are submitted to left-right oracles, Ni_0 for left messages and Ni_1 for right messages. Now, if $Ni_0 = Ni_1$ then $\mathcal{B}$ aborts immediately and answers **Nonce Collision**. He also generates a random bit as his output. Otherwise, $\mathcal{B}$ proceeds by considering the different actions of the protocol.

$\mathcal{B}$'s behavior Before describing the behavior of $\mathcal{B}$, we have to introduce the following notation. Consider a symbolic substitution σ as above. Then we denote by $\mathcal{C}(\sigma)$ the extension of σ to all variables, i.e. $\mathcal{C}(\sigma)$ is a variable assignment that coincides with σ on σ's domain and assigns a nonce N_x to each variable $x \notin sup(\sigma)$.

We now describe the actions of $\mathcal{B}$ for each instruction $inst$ of the protocol. As $\mathcal{B}$ does not know all the secret keys nor the challenge bit b, $\mathcal{B}$ uses the procedures $concr'$ and $parse'$ explained below. If $\mathcal{B}$ finishes normally with all instructions, he returns **DY** and tries to randomly guess the value of bit b, thus his output is a random bit b'.

As we only investigate the case of asymmetric encryption, the only possible actions are sending and reception of messages (signature verification is not used).

1. Consider the case of an input instruction $\textsc{Recv}(t)$. Suppose that the bit-string sent by $\mathcal{A}$ is bs. Then, t_f is updated as follows $t_f := t_f \cdot \textsc{Recv}(t)$. Moreover, $\mathcal{B}$ calls the procedure $parse'(bs, t)$ given below. If $parse'(bs, t)$ returns abnormally then $\mathcal{B}$ assigns back to t_f its initial value, i.e. cancels the last concatenation $\textsc{Recv}(t)$, and returns **DY**.

2. The other type of instruction to consider is an output $\textsc{Send}(t)$. $\mathcal{B}$ computes $concr'(t)$, outputs $concr'(t)$ and updates t_f as follows $t_f := t_f \cdot \textsc{Send}(t)$.

The procedure $parse'$ Procedure $parse'$ first checks whether the types of bs and t are compatible. If it is not the case then it aborts and adversary $\mathcal{B}$ returns **DY** as final status. The behavior of $parse'$ is close to the behavior of $parse$ the main difference is that whenever $parse'$ has to associate a value bs to a variable for the first time, it tests the equality between bs and any

variable or atom from θ. If one of such equalities holds, the symbolic substitution σ is updated in order to reflect this equality. Moreover, $parse'$ also tests the equality between bs and Ni_0 and bs and Ni_1. If one of this equality holds, $\mathcal{B}$ uses this to deduce the value of bit b: if the first equality holds, $\mathcal{B}$ outputs 0. Respectively, if the second equality holds, $\mathcal{B}$ outputs 1. Adversary $\mathcal{B}$ can also win his challenge if t is the inverse of a challenge key.

Algorithm $parse'(bs, t, \theta, \sigma)$:
  if $t = Ni$ **then**
    if $bs = Ni_0$ **then** $\mathcal{B}$ outputs **NDY**, 0
    if $bs = Ni_1$ **then** $\mathcal{B}$ outputs **NDY**, 1
    else raise parse-error
  if t is the inverse of one of the challenge keys **then**
    $\mathcal{B}$ outputs the value **NDY** and the value of the challenge bit
  if $\mathsf{type}(bs)$ and t are compatible **then**
    match t **with**
    $[x]$ Variable or Atom
      if $x \in sup(\theta)$ **then**
        if $bs = x\theta$ **then return** θ
        else raise parse-error
      else $x\theta := bs$
        if $bs \in range(\theta)$ **then** $x\sigma := find(bs, \theta)$
    $[enc_a(t_1, t_2)]$ Asymmetric Encryption
      if t_2 is not a challenge key **then**
        $parse'\big(\mathcal{D}^a(bs, concr(\eta, t_2, \theta)), t_1, \theta\big)$
      else if bs was not produced by the left-right oracle
        $parse'\big(\mathcal{O}_{t_2}(bs), t_1, \theta\big)$
      else
        t is unified with the symbolic message that lead to use the left-right oracle
        σ and θ are modified according to this

Oracle $\mathcal{O}_{t_2}$ designates the decryption oracle related to key from t_2.

The advantage of the ACM model appears clearly in this procedure: as variables are only used to store atoms, it is possible to easily test that a fresh atom value is "really" fresh and if it is not the case we just have to store in σ that there is an equality between the value of two atoms. When considering the standard computational model, parsing is more complicated. For example, $\mathcal{B}$ can receive a bit-string bs that he links to a variable x. Later $\mathcal{B}$ can realize that bs is an encryption using a key whose inverse was not known by $\mathcal{B}$ previously. The information encrypted in bs might have been useful to $\mathcal{B}$ in order to win his challenge when he received bs but this information can be useless when $\mathcal{B}$ receives the decryption key that allows him to understand the structure of bs. This is called the *late commitment problem* and is very difficult to handle.

Example 6.2 *Let us give a concrete example of this late commitment problem. Thus let us consider the protocol that contains the following role;*

$$\textsc{Recv}(x).\textsc{Send}(N).\textsc{Recv}(y).\textsc{Recv}(\{N\}_{y^{-1}})$$

Then let us consider that adversary $\mathcal{A}$ against the protocol plays as follows. He sends a bit-string bs which is linked to x. $\mathcal{B}$ has to give him the value of N. After that, $\mathcal{A}$ sends a secret key sk and once more the bit-string bs. At this point $\mathcal{B}$ parses bs as the encryption of N using the public key related to sk. The corresponding trace is:

$$\textsc{Recv}(\{N\}_{pk}).\textsc{Send}(N).\textsc{Recv}(sk).\textsc{Recv}(\{N\}_{pk})$$

This trace is not valid in the symbolic world. However it is difficult for $\mathcal{B}$ to win. Indeed $\mathcal{B}$ cannot use N as his challenge nonce as he has to send N in the second action.

The JCM also allows us to get rid of this problem as the adversary $\mathcal{A}$ has to give all the components that are used in the produced bit-string. Thus it is not possible that $\mathcal{B}$ learns some informations on the bit-string too late.

The procedure *concr′* Let us first introduce some notations. We let $p(\theta, t)$ denote the pattern obtained from t by replacing each variable x in t by $\theta(x)$ and each non-challenge atomic message a by $\theta(a)$. Notice that since the protocol is executable, $\theta(x)$ must be defined. As we suppose that the protocol satisfies the acyclicity condition, the obtained pattern does not contain cycles.

Now, there are the following cases according to t:

1. If t is a variable x, then *concr′* returns $\theta(x)$.

2. If t is an atom a that is not the challenge nonce and not a challenge key, then *concr′* returns $\theta(a)$.

3. If t is an encryption $enc_a(t_1, t_2)$. The following cases can be considered:

   (a) t_1 does not contain any challenge (either the challenge nonce or a secret challenge key) as sub-message. Then *concr′* can compute the bit-string value of t_1 using a simple recursive function, i.e. it outputs $\mathcal{E}_a(concr'(t_1), concr'(t_2))$.

   (b) t_1 contains a challenge as sub-message and t_2 is a challenge key. Then, *concr′* computes two new patterns p_0, respectively p_1, by substituting in $p(\theta, t_1)$ nonce Ni by Ni_0, respectively Ni_1. Note that if the challenge nonce does not appear in t_1, patterns p_0 and p_1 are equal. Then, it calls the left-right encryption oracle corresponding to t_2 on $\langle p_0, p_1 \rangle$. Finally, *concr′* outputs the result produced by the left-right oracle.

   (c) t_1 contains a challenge as sub-message and t_2 is not a challenge key. Then, *concr′* proceeds using a simple induction: let $bs_1 := concr'(t_1)$ and $bs := \mathcal{E}_a(bs_1, concr'(t_2))$. *concr′* returns bs.

4. There are no other cases as symmetric encryption and digital signature are forbidden for now.

It is important to note that the *concr′* function may fail. For example, if it is called on the challenge nonce or on a secret challenge key. This means that $\mathcal{B}$ made an incorrect guess when generating his challenge keys and nonces.

Now that we have properly defined the *parse′* and the *concr′* function, the behavior of $\mathcal{B}$ can be detailed. This is done in figure 6.1.

Example 6.3 *To illustrate the behavior of adversary $\mathcal{B}$, we give a first example. This example is one of the simplest protocols one can imagine consisting of the action* $\textsc{Recv}(N)$.

Initially θ and σ are the empty mappings and t_f is the empty sequence. The only possible challenge nonce is N. To perform the simulation, $\mathcal{B}$ randomly generates Ni_0 and Ni_1. Adversary $\mathcal{B}$ executes instruction $\textsc{Recv}(N)$ *therefore he waits for $\mathcal{A}$ to output a bit-string bs. Suppose first that $bs = Ni_0$. Now, $\mathcal{B}$ sets $t_f := [\textsc{Recv}(N)]$, which is obviously not a valid symbolic trace, and calls $parse'(bs, N)$. The first test of $parse'$ (comparing the bit-string bs to Ni_0) is a success. Hence $\mathcal{B}$ outputs 0 as he thinks that Ni_0 was used in the protocol and a cryptographic primitive was broken by $\mathcal{A}$. However since no left-right oracle has been called, $\mathcal{B}$ does not get any significant advantage. This case has a negligible probability to happen as this means that random nonce has been guessed by $\mathcal{A}$ without any information on it.*

Assume now that $bs \neq Ni_0, Ni_1$. Then, $parse'$ is applied again but here $\mathcal{B}$ aborts (as bs does not have a value that is compatible with N). This is the right thing to do as t_f is obviously not a valid symbolic trace.

Adversary $\mathcal{B}(\eta)$:
randomly generate CK and Ni
generate Ni_0 and Ni_1
initialize all the remaining nonces and keys
for i **from** 1 **to** k
match $inst_i$ **with**
[RECV(t)] Message reception
$(bs, mem) := \mathcal{A}(mem)$
$\theta := parse'(bs, t, \theta, \sigma)$
$trace := trace :: \text{RECV}(t)$
[SEND(t)] Message emission
$trace := trace :: \text{SEND}(t)$
$mem := \mathcal{A}(concr'(t, \theta), mem)$
endmatch
endfor

Figure 6.1: Adversary $\mathcal{B}$

Advantage of $\mathcal{B}$

The objective is now to relate the advantage of $\mathcal{B}$ to the probability that $\mathcal{A}$ creates an invalid symbolic trace. The intuition is that whenever $\mathcal{A}$ outputs a trace that cannot be abstracted to a correct symbolic trace, there is (at least) one possible choice of challenge keys and of the challenge nonce that allows $\mathcal{B}$ to deduce challenge bit b. This is the reason why $\mathcal{B}$ produces a symbolic trace t_f. If $\mathcal{B}$ has to append a non-deducible message to this trace, he is able to win his challenge.

Non-Dolev-Yao Messages Whenever the $parse'$ function is called, adversary $\mathcal{B}$ tests that the resulting message is deducible from previously exchanged messages and his initial knowledge. According to the hypotheses we made over protocols, the initial knowledge of the intruder contains every public keys and agent identities. Hence, let m be a non-deducible message. Then m contains a nonce N that has always been encrypted in previously sent messages. The idea is that with non-negligible probability, this nonce is the trapped nonce Ni. Thus, by getting the value Ni_0 or Ni_1, $\mathcal{B}$ is able to deduce bit b and to win his challenge. Another possibility is that m contains a secret key pk^{-1} that was always encrypted by secure keys in the previously exchanged message. The idea here is that pk is a challenge key with non negligible probability. Knowing pk^{-1}, adversary $\mathcal{B}$ can easily deduce the challenge bit b and win his challenge.

Let us formalize this in the following proposition.

Proposition 6.1 *Let m, m_1,... ,m_k be $k+1$ messages. Let IK be a set that contains every public keys and identities that appear either in m or in one of the m_i.*

Let us suppose that m is not deducible from $E = IK, m_1, ..., m_k$. Then there exists a position p such that $m_{|p}$ is either a nonce, or a secret key which is not deducible from E. Moreover, for any prefix p' of p, $m_{|p'}$ is not deducible from E.

Proof: This proof can be done using a structural induction on m. For this purpose, we study the following different cases:

1. If m is an atom a, then as IK contains all the identities and public keys m is either a nonce or a secret key. a is not deducible from E thus the conclusion is immediate by taking $p = \epsilon$.

2. If m is a pair $\langle m_0, m_1 \rangle$, then either m_0 or m_1 is not deducible from E. By symmetry, let us suppose that m_0 is not deducible. The induction hypothesis applies to m_0, hence we get a position p in m_0 that is suitable for m_0. Hence position $0.p_0$ fits for m. Then we just have to verify that m_ϵ is not deducible and this is true as m_ϵ is m.

3. If m is an asymmetric encryption $\{n\}_{pk}$, then as pk appears in IK, n is not deducible (otherwise m would also be deducible). Therefore, the induction hypothesis applies on n which has a suitable position p. Let us consider position $0.p$ for m. Then we just have to verify that m_ϵ is not deducible. However m_ϵ is m hence it is not deducible.

■

Thus if a message is non-deducible, it contains (with non-negligible probability) a secret that allows $\mathcal{B}$ to guess the value of challenge bit b. Furthermore, any encryption that could hide this secret is non-deducible, thus such encryptions have not been forged by the encryption oracles and the decryption oracles can be used to recover the secret.

Probabilities Let us define the following sets of events corresponding to the different outputs of $\mathcal{B}$:

1. E_0: **DY**, the execution behaved correctly but the final t_f is a valid symbolic trace.
2. E_1: **NDY**, adversary $\mathcal{A}$ produced a non-deducible message, t_f is not a valid symbolic trace but bit b has been found.
3. E_2: **Nonce Collision**, two nonces (generated in the initialization of $\mathcal{B}$) have the same bit-string representation.

As each of these events corresponds to a possible output of $\mathcal{B}$, these events are disjoint and we have that:

$$Pr[E_0] + Pr[E_1] + Pr[E_2] = 1$$

Hence in the case of E_0 and E_1, the nonces generated by $\mathcal{B}$ have to be different one from another. As detailed later in this section, the probability of nonce collision is negligible hence $Pr[E_2]$ is negligible. Let us now detail the advantage of $\mathcal{B}$: $\mathcal{B}$ correctly guesses the value of challenge bit b when event E_1 occurs. For the two other events, $\mathcal{B}$ uses a random bit as its output. Hence the advantage of $\mathcal{B}$ against N-PAT-IND-CCA (denoted by γ_N) is given by:

$$\begin{aligned} \mathbf{Adv}_{\mathcal{B}}^{\gamma_N}(\eta) &= 2Pr[E_1] + Pr[E_0] + Pr[E_2] - 1 \\ &= Pr[E_1] \end{aligned}$$

We suppose that the asymmetric encryption scheme used to implement the protocol is secure against IND-CCA. Then according to proposition 5.4, this encryption scheme is also secure against N-PAT-IND-CCA. Therefore the advantage of $\mathcal{B}$ against the N-PAT-IND-CCA criterion is negligible and the probability of E_1 is also negligible.

Let us know consider the probability p that $\mathcal{A}$ produces a trace that is incorrect in the symbolic setting. Probability p is defined by:

$$p = Pr[\nexists t_f \in \mathsf{Traces}(\Pi) \cdot Exec(\mathcal{A}, \Pi, \theta) \in Concr(t_f, \theta)]$$

Then p is the probability of an event E which denotes that $\mathcal{A}$ faced to Π produced a trace that cannot be abstracted to a valid symbolic trace. In this case, as trace t_f output by $\mathcal{B}$ is a correct abstraction, the last message received by $\mathcal{B}$ is not deducible from previous ones. According to proposition 6.1, $\mathcal{B}$ wins his challenge in such cases. However, this only occurs if $\mathcal{B}$ has correctly chosen the challenge keys and the challenge nonce but this happens with non-negligible probability.

$$Pr[E_1] \geq \frac{1}{2^N N} p$$

This relation is not an equality as multiple choices of challenge keys and the challenge nonce can lead to the victory of $\mathcal{B}$. Finally, we get that p can be bounded by:

$$p \leq 2^N . N . \mathbf{Adv}_{\mathcal{B}}^{\gamma_N}(\eta)$$

As p is positive, this probability is negligible in η. This concludes the proof of theorem 6.1.

Nonces are Probably Different We consider that anytime a computational adversary picks up some nonces, they are different one from another. The adversary can only get a number m of nonces that is polynomial in η and we suppose that the number n of possible nonces is exponential in η (so $m < n$). Let P be the probability that the adversary gets twice the same nonce.

$$1 - P = \frac{n}{n}\frac{n-1}{n}...\frac{n-(m-1)}{n}$$

Thus, we have the following inequalities:

$$0 \leq P \leq 1 - \big(1 - \frac{m-1}{n}\big)^m$$

Proposition 6.2 *For any $x \in [0,1[$ and $a \geq 1$,*

$$\big(1-x\big)^a \geq 1 - x.a$$

Proof: Consider the function $f(x) = \big(1-x\big)^a - 1 + x.a$. Derive it twice to get the result. ∎

Applying the proposition, we get:

$$0 \leq P \leq \frac{m.(m-1)}{n}$$

As m is polynomial and n is exponential in η, P is negligible in η. When considering an adversary that has a non-negligible advantage against something, this adversary still has his advantage if we consider only executions where nonces are distinct.

6.2.3 Adding more Cryptographic Primitives

In this section, we generalize the result from the previous section to a more general class of protocols. This is done by considering more cryptographic primitives such as symmetric encryption. However we still consider the ACM to simplify things. Within the ACM, considering digital signature is useless: variables can only be used to receive atoms, hence it is not possible to receive new signatures. The only verification that can be done by a role involves a signed message that is forged by the same role, therefore the result of the verification can be statically determined. For this reason, we only consider the addition of symmetric encryption.

Theorem 6.2 *Let Π be a valid protocol that uses a safe cryptographic library $\mathcal{CL}$. Let $\mathcal{A}$ be an adversary. Then, the following probability is negligible as a function of η:*

$$Pr[\nexists t_f \in \mathsf{Traces}(\Pi) \cdot Exec(\mathcal{A}, \Pi, \theta) \in Concr(t_f, \theta)]$$

The proof of this theorem is very close to the proof given for theorem 6.1.

Let us use the same notations: protocol Π is composed of a role R and an initial knowledge IK. Integer N denotes the number of different atoms that occur in Π. We still have that the number of asymmetric keys used by Π is lower than N, the number of different nonces and the number of symmetric keys used by Π are also lower than N.

As earlier, the proof of this theorem is done by reduction to the security of the underlying cryptographic schemes. Let $\mathcal{A}$ be an adversary interacting with the protocol, we build an adversary $\mathcal{B}$ playing against the N-PASS criterion (for cryptographic library $\mathcal{CL}$) such that the above probability is bounded by the advantage of $\mathcal{B}$.

The cryptographic library used to implement the protocol is composed by an asymmetric encryption scheme $\mathcal{AE}$, a symmetric encryption scheme $\mathcal{SE}$ and a digital signature scheme $\mathcal{SS}$. As $\mathcal{CL}$ is supposed to be safe, $\mathcal{AE}$ is secure against IND-CCA, $\mathcal{SE}$ is secure against SYM-CPA and $\mathcal{SS}$ is secure for UNF (although $\mathcal{SS}$ is not used in this situation). By using proposition 5.8, we get that the cryptographic library is secure for N-PASS.

Adversary $\mathcal{B}$ The design of adversary $\mathcal{B}$ is similar to the one proposed in the previous section. However functions $parse'$ and $concr'$ have to be adapted in order to handle symmetric encryptions. The main difference with respect to asymmetric encryption is that there are no decryption oracles for symmetric keys that the $parse'$ function may use. Remember that the decryption oracle was used whenever the protocol had to parse an encryption using one of the challenge key and this encryption had not been produced by the left-right encryption oracle. In this situation for symmetric encryption, there is no need to decrypt the message: as this message is a fresh encryption using a challenge key, this allows adversary $\mathcal{B}$ to win his challenge against forgeability of symmetric encryption.

The $concr'$ algorithm has to be modified in order to use the left-right encryption oracle for symmetric keys from CK. When $concr'$ is called on a symmetric encryption $\{t_1\}_{t_2}$ where t_2 is a challenge key (i.e. appears in CK), then $concr'$ generates two patterns corresponding to t_1 in the case where Ni is instantiated with Ni_0 and in the case where it is instantiated with Ni_1. These patterns are submitted to the left-right encryption oracle and $concr'$ returns the resulting bit-string.

Non-deducible Messages and Probabilities Whenever $\mathcal{B}$ has to append a non-deducible bit-string to trace t_f, $\mathcal{B}$ is able to win by either guessing the value of challenge bit b or by forging a fresh symmetric encryption. There are different possibilities that can make $\mathcal{B}$ guess b: the message can be non-deducible because of a nonce (and with non-negligible probability, this is nonce Ni). It can be non-deducible because of a secret or symmetric key then it is easy to get the value of b.

The following proposition describes why a message can be non-deducible.

Proposition 6.3 *Let m, m_1,... ,m_k be $k+1$ messages. Let IK be a set that contains every public keys and identities that appear either in m or in one of the m_i.*

Let us suppose that m is not deducible from $E = IK, m_1, ..., m_k$. Then there exists a position p such that $m_{|p}$ is either a nonce, a symmetric key, a secret key or a symmetric encryption which is not deducible from E. Moreover, for any prefix p' of p, $m_{|p'}$ is not deducible from E.

Proof: This proof can be done using a structural induction on m. For this purpose, we study the different following cases:

1. If m is an atom a, then as IK contains all the identities and public keys m is either a nonce, a symmetric key or a secret key, a is not deducible from E thus the conclusion is immediate by taking $p = \epsilon$.

2. If m is a pair $\langle m_0, m_1 \rangle$, then either m_0 or m_1 is not deducible from E. By symmetry, let us suppose that m_0 is not deducible. The induction hypothesis applies on m_0, hence we get a position p in m_0 that is suitable for m_0. Hence position $0.p_0$ is suitable for m. Then we just have to verify that m_ϵ is not deducible and this is true as m_ϵ is m.

3. If m is an asymmetric encryption $\{n\}_{pk}$, then as pk appears in IK, n is not deducible (otherwise m would also be deducible). Therefore, the induction hypothesis applies on n which has a suitable position p. Let us consider position $0.p$ for m. Then we just have to verify that m_ϵ is not deducible. However m_ϵ is m hence it is not deducible.

4. If m is a symmetric encryption $\{n\}_k$, then as m is not deducible, ϵ can be used for position p.

■

Thus if a message is non-deducible, it contains a secret that allows $\mathcal{B}$ to win his game. Furthermore, any encryption that could hide the secret is non-deducible, thus such encryptions have not been forged by the left-right encryption oracles. In the case of asymmetric cryptography, the decryption oracle can be used to recover the secret. In the case of symmetric cryptography, $\mathcal{B}$ directly wins his game by forging a fresh symmetric encryption.

As in the previous proof, we define three different events corresponding to the output of $\mathcal{B}$:

1. E_0: **DY**, the execution behaved correctly but the final t_f is a valid symbolic trace.
2. E_1: **NDY**, adversary $\mathcal{A}$ produced a non-deducible message, t_f is not a valid symbolic trace but bit b has been found or a fresh symmetric encryption has been forged.
3. E_2: **Nonce Collision**, two nonces (generated in the initialization of $\mathcal{B}$) have the same bit-string representation.

As each of these events corresponds to a possible output of $\mathcal{B}$, these events are disjoint and we have that:

$$Pr[E_0] + Pr[E_1] + Pr[E_2] = 1$$

Hence in the case of E_0 and E_1, the nonces generated by $\mathcal{B}$ have to be different one from another. As we saw in the previous section, the probability of nonce collision is negligible hence $Pr[E_2]$ is negligible.

Let us now give the advantage of $\mathcal{B}$ if we suppose that $\mathcal{B}$ correctly chose CK and Ni: $\mathcal{B}$ correctly guesses the value of challenge bit b or forges a fresh symmetric encryption when event E_1 occurs. For the other two events, $\mathcal{B}$ tries to guess the value of b by outputting a random bit. Event E_1 is split in two: event E_1^b when $\mathcal{B}$ guesses the value of b and event E_1^s when $\mathcal{B}$ forges a fresh symmetric encryption. Hence the advantages of $\mathcal{B}$ against N-PASS/IND (denoted by γ_N) and N-PASS/UNF (denoted by δ_N) are given by:

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{B}}^{\gamma_N}(\eta) &= 2Pr[E_1^b] + Pr[E_1^s] + Pr[E_0] + Pr[E_2] - 1 \\
&= Pr[E_1^b] \\
\mathbf{Adv}_{\mathcal{B}}^{\delta_N}(\eta) &= 2Pr[E_1^s] - PrRand^{UNF}
\end{aligned}$$

Where $PrRand^{UNF}$ denotes the probability to generate a valid symmetric encryption (for a randomly sampled key) without any oracle. The cryptographic library used to implement the protocol is supposed to be safe. Hence the advantage of $\mathcal{A}$ against γ_N and δ_N is negligible. Moreover $PrRand^{UNF}$ is negligible, thus we get that $Pr[E_1^b]$ and $Pr[E_1^s]$ are negligible and so $Pr[E_1]$ is also negligible.

From here, the theorem's proof can be completed by using the same reasoning as described at the end of the previous section.

6.2.4 Considering the General Model

Although the ACM can be used to analyse many simple protocols such as the Needham-Schroeder-Lowe protocol, it is not suited to represent protocols that deal with digital signature or message forwarding. For this reason, we want to extend our soundness result to the case of the JCM and of the full computational model.

The JCM allows for simpler proofs but is not representative of reality: in real life situations, adversaries do not have to justify the attacks they are performing. Hence in this section we mainly focus on the full computational model. However the previous soundness proof can be adapted to handle the JCM as parsing can be entirely achieved when receiving a bit-string.

The main complication when considering the full computational modal compared to the ACM comes from the emission of secret keys in the case of asymmetric cryptography and from the emission of keys in the case of symmetric cryptography. This complication is linked to the proof strategy we used previously. This strategy supposes that each time the adversary sends a bit-string, this bit-string can be totally parsed. However sending secret keys raises the issue of the *late commitment problem* as described previously. It is possible that the adversary sends first a bit-string then later sends the corresponding decryption key. This decryption keys allows us to parse the first bit-string but if adversary $\mathcal{B}$ only notices at that point that the corresponding symbolic message is non deducible, it might be impossible for $\mathcal{B}$ to win the challenge because of the messages that he had to send between the first bit-string and the key.

Let us give another concrete example of this, consider the following protocol where A is impersonated by the adversary.

$$\begin{array}{lcl} B \to A & : & \{N\}_{k'} \\ A \to B & : & x \\ B \to A & : & N \\ A \to B & : & x_k \\ A \to B & : & \{N\}_{x_k} \end{array}$$

Let $\mathcal{A}$ be a computational adversary against this protocol, let us imagine an adversary $\mathcal{B}$ close to the one proposed earlier in this section. $\mathcal{B}$ uses $\mathcal{A}$ as a subroutine and tries to break symmetric encryption (in this case, the indistinguishability part and not the authentication). Therefore $\mathcal{B}$ generates two possible values bs_0 and bs_1 for nonce N. Using his left-right encryption oracle, $\mathcal{B}$ produces $\mathcal{E}(bs_b, k')$ and gives it to adversary $\mathcal{A}$. Then $\mathcal{A}$ outputs a bit-string bs which corresponds to x. At this point, $\mathcal{B}$ has to give bs_b to $\mathcal{A}$ hence he cannot terminate this simulation. We would like to argue that N cannot be the reason why the trace output by $\mathcal{A}$ is not possible in the symbolic setting however this is not the case. Namely if we managed to send the value of N to $\mathcal{A}$, $\mathcal{A}$ may have output a bit-string bs_k for key x_k and then bit-string bs for message $\{N\}_{x_k}$. $\mathcal{B}$ can obtain the value of N but because he has sent N before, this cannot be used to break symmetric encryption. The corresponding symbolic trace is:

$$\begin{array}{lcl} B \to A & : & \{N\}_{k'} \\ A \to B & : & \{N\}_{k} \\ B \to A & : & N \\ A \to B & : & k \\ A \to B & : & \{N\}_{k} \end{array}$$

This trace is impossible in the symbolic setting but there is no easy way to break symmetric encryption by using $\mathcal{A}$. When $\mathcal{B}$ receives $\{N\}_k$, he does not know the corresponding decryption key and hence he cannot get the value of N and wins his challenge. The value of N can only be discovered when $\mathcal{A}$ sends k but this is to late as $\mathcal{B}$ had to give N already.

A similar problem is raised in the work of Backes, Pfitzmann and Waidner [BPW03b]. This issue is solved by assuming that *the overall protocol ensures that keys are not sent after being used.* In their context, this assumption clearly eliminates the commitment problem. According to their work, this assumption is generally met when considering protocols from the Clark and Jacob's survey [CJ97]. Only one out of the fifty protocols does not fulfill this requirement: the Wide Mouthed Frog protocol, which is flawed. Hence this restriction seems quite fair even for real-world protocols. However it is not clear how this restriction can be adapted to fit in our model. It is possible to imagine more complex protocols where the issue appears but the use of the key before its sending comes from a collision.

In our context, a simple solution is to forbid sending secret keys. Then parsing of a message cannot change because of new knowledge. Hence it is possible to know when receiving a bit-string if the corresponding message is non-deducible. If it is the case, security of one of the cryptographic scheme can be broken. However, this hypothesis is unrealistic for most security protocols as key sending is often used.

Another possible way of handling key sending is to consider stronger cryptographic assumptions than the usual semantic security. This is developed in Romain Janvier's thesis [Jan06]. The soundness result is proven in the full computational model. However Janvier has to add new cryptographic requirements which are closely related to commitment. It is not clear whether these requirements are implied by classical requirements or not.

6.2.5 Extending the Results

In this section, we consider two extensions of the previous results. In the first one, we add a new primitive close to hashing. In the second part, we show how it is possible to modify the Dolev-Yao model when considering weaker computational primitives, this is demonstrated on a block cipher.

Hash Functions

Hashing is a cryptographic primitive that is very commonly used in security protocols. Its classical use relies on two particularities: hashing is deterministic, therefore it is possible to authenticate a message m knowing its hashed value; hashing is supposed to be collision-resistant, hence authentication provided by a hashing is reliable. Probabilistic hash functions have been proposed recently [Can97], however we only consider deterministic keyed hash functions in this section.

Symbolic Model

First let us specify how hashing is modeled in the symbolic setting. The definition of message is modified by adding a new function h of arity 1. For example, hashing of the pair of nonces N and N' is represented by message $h(\langle N, N' \rangle)$.

Protocols are also modified: there is a new instruction which allows an agent to test that a value is the hash of a message, this new instruction is denoted by $[x = h(t)]$ where x is a variable and t is a term. The modified semantics is pretty straightforward, a substitution σ is possible iff $x\sigma = h(t\sigma)$ where the $=$ relation here denotes syntactic equality.

The last modification for the symbolic model concerns the intruder deduction relation. A new rule is added to the inductive definition of this relation:

- If $E \vdash m$, then $E \vdash h(m)$.

If a message is deducible by the adversary, then its hashing is also deducible. As this is the only modification to the deduction relation, the adversary cannot deduce anything on m from $h(m)$.

Computational Requirements In the computational world, a *keyed hashing algorithm* is a polynomial deterministic algorithm that computes from a key k and a bit-string bs another bit-string of size η. The key generation algorithm is not important and one can suppose that k is chosen randomly among strings of size η. The same key is used for all applications of the hash function and this key is public. The only use of this key is to make collisions harder to find (it is impossible for an adversary against collision-freeness to output fixed collisions as these collisions depend on the randomly chosen key).

Usual security requirements for a hashing algorithm $\mathcal{H}$ are one-wayness and collision resistance. One-wayness is a way to represent secrecy, it is impossible from $\mathcal{H}(bs)$ to deduce the bit-string bs. Formally, the following advantage is negligible for any adversary $\mathcal{A}$:

$$\mathbf{Adv}^{OW}_{\mathcal{A}}(\eta) = Pr[k \xleftarrow{R} [0,1]^\eta \ , \ bs \xleftarrow{R} [0,1]^\eta \ , \ bs = \mathcal{A}(\eta, \mathcal{H}(k, bs))]$$

Collision resistance means that finding a collision is hard:

$$\mathbf{Adv}^{CR}_{\mathcal{A}}(\eta) = Pr[k \xleftarrow{R} [0,1]^\eta \ , \ (bs_1, bs_2) := \mathcal{A}(\eta) \ , \ \mathcal{H}(k, bs_1) = \mathcal{H}(k, bs_2)]$$

Note that it is not possible to statically encode collisions in $\mathcal{A}$ as the value produced by algorithm $\mathcal{H}$ depends on k which is randomly generated.

Let us discuss now the requirements we have to put on hashing in order to adapt the proof of the soundness theorem 6.2. For secrecy, hashing has to satisfy an indistinguishability criterion as one-wayness is not sufficient. Hashing has to satisfy an unforgeability criterion as it is not possible in the symbolic world to compute a fresh hashing using old ones. Finally equality tests for hashing are modeled by bit-string tests in the computational setting, hence we also ask our hashing algorithm to satisfy collision resistance. The main difficulty when formulating these

criteria is that the hashing algorithm is deterministic. Thus the indistinguishability criterion is not straightforward: the left-right hashing oracle takes as arguments two patterns which must ask for inclusion of a secret nonce N^H by using pattern variable $[N^H]$. If we do not ask that, the adversary can submit the pair $\langle 0, 1\rangle$ to the left-right oracle and compute the values of $\mathcal{H}(k, 0)$ and $\mathcal{H}(k, 1)$ (all the participants know the value of hash key k, even the adversary). Moreover, the adversary cannot submit the same pattern twice (otherwise it is easy to design adversaries with non-negligible advantages).

The HASH Criterion The HASH criterion is a combination of an IND-CCA criterion, an UNF criterion and a collision free criterion. A hashing algorithm needs to verify three properties to be secure. First an adversary cannot obtain information on a bit-string bs when looking at $\mathcal{H}(k, bs)$. The second property is that if an adversary does not know a bit-string bs, he cannot produce $\mathcal{H}(k, bs)$ even if he knows hashing of messages meaningfully related to bs. Finally, it must be hard for an adversary to find two different messages which have the same hash for a given key. More details about criteria related to HASH can be found in the excellent book of Bellare and Rogaway [BR03].

The HASH criterion γ takes as argument a hash function $\mathcal{H}$ and returns $(\Theta; F; V)$. Θ generates a bit b, a key k and a random bit-string N^H of size η. Oracle F gives access to two oracles: an oracle which gives the value of key k and a left-right hashing oracle which takes as input a pair $\langle pat_0, pat_1\rangle$ of hollow patterns (an *hollow* pattern can ask for inclusion of N^H and have to ask for it at one position at least) and outputs $\mathcal{H}(k, pat_b[N^H])$. Moreover, each pattern can only be submitted once to this oracle in order to avoid guessing attacks. Verifier V contains three parts: V_{IND} returns true if the adversary outputs the challenge bit b; V_{UNF} returns true if the adversary outputs a pair $\langle h, pat\rangle$ such that $h = \mathcal{H}(k, pat[N^H])$ and h was not produced by F; V_{CF} returns true if the adversary outputs a pair $\langle bs_0, bs_1\rangle$ such that $\mathcal{H}(k, bs_0) = \mathcal{H}(k, bs_1)$, and bit-strings bs_0 and bs_1 are different.

A hashing algorithm is said HASH if it verifies γ.

The criterion related to IND $(\Theta; F; V_{IND})$ (resp. to UNF $(\Theta; F; V_{UNF})$) is denoted by HASH/IND (resp. HASH/UNF). The last criterion related to collision free is denoted HASH/CF.

In practice, we only ask the $\mathcal{H}$ algorithm to satisfy the indistinguishability criterion and collision resistance as unforgeability is a consequence of these two criteria.

Proposition 6.4 *If an algorithm $\mathcal{H}$ verifies HASH/IND and HASH/CF and $PrRand^{CF}$ and $PrRand^{UNF}$ are negligible, then $\mathcal{H}$ verifies HASH/UNF and HASH.*

Proof: We first state two technical lemmas:

Lemma 6.1 *Let $(a_i)_{1\leq i\leq n}$ be n real numbers. Then*

$$\sum_{1\leq i\leq n} a_i^2 \geq \frac{1}{n} \sum_{1\leq i,j\leq n} a_i a_j$$

Proof: By developing $(a_i - a_j)^2 \geq 0$, we obtain

$$a_i^2 + a_j^2 \geq 2a_i a_j$$

$$\sum_{1\leq i,j\leq n} a_i^2 + a_j^2 \geq 2 \sum_{1\leq i,j\leq n} a_i a_j$$

$$2n \sum_{1\leq i\leq n} a_i^2 \geq 2 \sum_{1\leq i,j\leq n} a_i a_j$$

$$\sum_{1\leq i\leq n} a_i^2 \geq \frac{1}{n} \sum_{1\leq i,j\leq n} a_i a_j$$

■

Using this first lemma, we are able to prove the following inequality between probabilities.

Lemma 6.2 *Let X, Y and Y' be three independent random variables. X is chosen randomly in a finite set S_X, Y and Y' are chosen randomly in the finite set S_Y. Let E be a predicate over $S_X \times S_Y$, then*

$$pr\big(E(X,Y) \wedge E(X,Y')\big) \geq \big[pr\big(E(X,Y)\big)\big]^2$$

Proof: To prove this lemma, let p be the left probability. Hence,

$$\begin{aligned} p &= pr\big(E(X,Y) \wedge E(X,Y')\big) \\ &= \frac{1}{|S_X|} \sum_{x \in S_X} pr\big(E(x,Y) \wedge E(x,Y')\big) \\ &= \frac{1}{|S_X|} \sum_{x \in S_X} pr\big(E(x,Y)\big).pr\big(E(x,Y')\big) \\ &= \frac{1}{|S_X|} \sum_{x \in S_X} pr\big[\big(E(x,Y)\big)\big]^2 \end{aligned}$$

Then, using lemma 6.1, we get:

$$\begin{aligned} p &\geq \frac{1}{|S_X|^2} \sum_{x,x' \in S_X} pr\big[\big(E(x,Y)\big)\big].pr\big[\big(E(x',Y)\big)\big] \\ &= \Big(\frac{1}{|S_X|} \sum_{x \in S_X} pr\big[\big(E(x,Y)\big)\big]\Big)^2 \\ &= \Big(pr\big[\big(E(X,Y)\big)\big]\Big)^2 \end{aligned}$$

■

Let $\mathcal{H}$ be a hash function that verifies HASH/IND and HASH/CF. Let us suppose that there exists an adversary $\mathcal{A}$ against HASH/UNF (also written UNF in the following) whose advantage is not negligible. Then we build the adversary $\mathcal{B}$ against HASH/IND (also written IND in the following) which executes $\mathcal{A}$ ($\mathcal{A}$ uses directly oracles given to $\mathcal{B}$).

Adversary $\mathcal{B}(\eta)/\mathcal{O}_k, \mathcal{O}_{LR}$:
$pat, bs := \mathcal{A}(\eta)/\mathcal{O}_k, \mathcal{O}_{LR}$
$N' \xleftarrow{R} [0,1]^\eta$
$pat' := \langle [], N' \rangle$
$bs' = \mathcal{O}_{LR}(pat, pat')$
if $bs = bs'$ **return** 0
else $b' \xleftarrow{R} [0,1]$
return b'

The advantage of $\mathcal{B}$ against IND is given by:

$$\mathbf{Adv}_{\mathcal{B}}^{IND} = Pr[\mathcal{B} \to 0 \text{ in } \mathbf{G}_{\mathcal{B}}^{IND} | b = 0] - Pr[\mathcal{B} \to 0 \text{ in } \mathbf{G}_{\mathcal{B}}^{IND} | b = 1]$$

We can now compare the game $\mathbf{G}_{\mathcal{B}}^{IND} | b = 0$ and the game $\mathbf{G}_{\mathcal{A}}^{UNF}$.

Game $\mathbf{G}_{\mathcal{B}}^{IND}(\eta)|b=0$:
$N^H \xleftarrow{R} [0,1]^\eta$
$k \xleftarrow{R} [0,1]^\eta$
$pat, bs := \mathcal{A}(\eta)/\lambda().k, \lambda\langle pat_0, pat_1\rangle\mathcal{H}(k, pat_0[N^H])$
$N' \xleftarrow{R} [0,1]^\eta$
$pat' := \langle [], N'\rangle$
$bs' = \mathcal{H}(k, pat[N^H])$
if $bs = bs'$ **return** 1
else $b' \xleftarrow{R} [0,1]$
return $b' = 0$

Game $\mathbf{G}_{\mathcal{A}}^{UNF}(\eta)$:
$N^H \xleftarrow{R} [0,1]^\eta$
$k \xleftarrow{R} [0,1]^\eta$
$pat, bs := \mathcal{A}(\eta)/\lambda().k, \lambda\langle pat_0, pat_1\rangle\mathcal{H}(k, pat_b[N^H])$
return $\mathcal{H}(k, pat[N^H]) = bs$

Thus we get that the advantage of $\mathcal{B}$ can be related to the advantage of $\mathcal{A}$ and to the advantage of another adversary $\mathcal{A}'$ against UNF.

$$\mathbf{Adv}_{\mathcal{B}}^{IND} = Pr[\mathbf{G}_{\mathcal{A}}^{UNF} = true] + \frac{1}{2}Pr[\mathbf{G}_{\mathcal{A}}^{UNF} = false] - Pr[\mathbf{G}_{\mathcal{A}'}^{UNF} = true] - \frac{1}{2}Pr[\mathbf{G}_{\mathcal{A}'}^{UNF} = false]$$

Where $\mathcal{A}'$ is an adversary against UNF defined by:

Adversary $\mathcal{A}'(\eta)/\mathcal{O}_k, \mathcal{O}_{LR}$:
$pat, bs := \mathcal{A}(\eta)/\mathcal{O}_k, \mathcal{O}_{LR}$
$N' \xleftarrow{R} [0,1]^\eta$
$pat' := \langle [], N'\rangle$
return pat', bs

We obtain

$$2\mathbf{Adv}_{\mathcal{B}}^{IND} = \mathbf{Adv}_{\mathcal{A}}^{UNF} - \mathbf{Adv}_{\mathcal{A}'}^{UNF}$$

Hence, as $\mathbf{Adv}_{\mathcal{B}}^{IND}$ is negligible and $\mathbf{Adv}_{\mathcal{A}}^{UNF}$ is not, $\mathcal{A}'$ has a non negligible advantage against HASH/UNF.

Finally, we build from $\mathcal{A}$ an adversary $\mathcal{C}$ against collision-freeness whose advantage is related to the advantage of $\mathcal{A}'$. For this purpose, $\mathcal{C}$ generates a nonce N^H in order to simulate with a function ρ the hash oracle used by $\mathcal{A}$.

Adversary $\mathcal{C}(\eta)/\mathcal{O}_k, \mathcal{O}_{LR}$:
$N^H \xleftarrow{R} [0,1]^\eta$
$pat, bs := \mathcal{A}(\eta)/\mathcal{O}_k, \rho$
$N' \xleftarrow{R} [0,1]^\eta$
$N'' \xleftarrow{R} [0,1]^\eta$
$pat' := \langle [], N'\rangle$
$pat'' := \langle [], N''\rangle$
return $pat'[N^H], pat''[N^H]$

Then, as $PrRand^{CF}$ is negligible, the probability that $\mathcal{C}$ finds a collision is negligible. Moreover, this probability is greater than the probability that $\mathcal{C}$ finds a collision and the hash of $pat'[N^H]$ is equal to the bs produced by $\mathcal{A}$. In the following, events like $\mathcal{H}(pat'[N^H]) = bs$ means: after the random execution of $\mathbf{G}_{\mathcal{A}'}^{UNF}$, we obtain pat', N^H and bs such that this equality holds. To deduce the second inequality, we use lemma 6.2 that is given just after this proof.

$$\begin{aligned} Pr[\mathbf{G}_{\mathcal{C}}^{CF} = true] &\geq pr\big(\mathcal{H}(pat'[N^H]) = bs = \mathcal{H}(pat''[N^H])\big) \\ &\geq Pr[\mathcal{H}(pat'[N^H]) = bs] \\ &\quad .Pr[\mathcal{H}(pat''[N^H]) = bs] \\ &\geq \big(Pr[\mathbf{G}_{\mathcal{A}'}^{UNF} = true]\big)^2 \end{aligned}$$

There is a contradiction as $\mathcal{A}'$ has a non negligible advantage and $PrRand^{UNF}$ is negligible. Hence $\mathcal{H}$ verifies HASH/UNF. ■

Usual hash functions (MD5, SHA1) produce a fixed-length output. This makes it possible to find collisions in a constant time (in η), clearly such primitives are not appropriate. It is not clear to us if there exists an algorithm that verifies $HASH$.

Hypotheses to Prove Soundness In order to extend our soundness result to protocols using hashing, the first hypothesis concerns the hashing algorithm which has to satisfy the HASH criterion. $PrRand^{HASH/UNF}$ and $PrRand^{HASH/CF}$ are negligible. The same hash key is used through the whole protocol execution, this allows protocols to use the unary notation $h(m)$.

The other requirement holds on protocols: each sent hash message contains a nonce that remains secret, hash messages cannot contain any secret keys: private keys for asymmetric encryption, symmetric keys and signature keys.

With these requirements, it seems possible to apply the same proof technique as the one used to prove theorem 6.2 and thus to obtain the same soundness result, see [Jan06] for details.

Block Ciphers

The objective of this section is to weaken the perfect cryptography hypothesis. This is first done in the formal model by adding a new rule to possible deductions. Then in the computational model, the classical IND-CCA criterion has to be modified. Hence, a soundness theorem can show that the extended Dolev-Yao model is still a safe abstraction of the computational model if we suppose that the encryption scheme verifies the modified IND-CCA property.

To enhance the adversary's capacity in both models, let us consider a pair of functions: δ_f takes a message as argument and outputs another message and δ_c uses a string of bits to produce another string of bits. Furthermore, these two functions have to be related: they must modify their arguments in the same way.

Definition 6.5 (DY-weakening) *A DY-weakening is a pair of function (δ_f, δ_c) such that for any computational substitution θ and any message m,*

$$\delta_c\left(concr(m,\theta)\right) = concr\left(\delta_f(m),\theta\right)$$

Where concr is the concretization algorithm. We also ask that given a string s, the set $S' = \left\{\delta_c^n(s), n>0\right\}$ can be computed in polynomial time.

This last hypothesis is useful when defining the computational criterion.

To illustrate this definition, we use the equational theories related to Block Ciphers (BC). We do not consider a precise implementation of BC but rather consider encryption schemes verifying similar equational theories. BC allows the adversary to make some deductions using a property verified on some block encryption schemes.

Example 6.4 (Block Ciphering) *BC is a DY-weakening that allows the adversary to deduce from an encoding of message m the encoding of a prefix of m. More formally,*

$$\begin{aligned}\delta_f(\{\langle m,n\rangle\}_{pk}) &= \{m\}_{pk}\\ \delta_c(\mathcal{E}(s.s',pk)) &= \mathcal{E}(s,pk)\end{aligned}$$

It is easy to verify that (δ_f,δ_c) constitutes a correct DY-weakening.

The Extended Dolev-Yao Model In the DY model, the deductibility relation $E \vdash m$ is defined by inference rules. We add a new inference which represents that an adversary can deduce from a message its image after applying δ_f.

$$\frac{E \vdash m}{E \vdash \delta_f(m)}$$

This defines an Extended Dolev-Yao model denoted by EDY. Note that usually, δ_f is defined on a subset of the set of messages. If δ_f cannot be applied to a message m, then our inference adds no new deductions starting from m.

Example 6.5 (Block Ciphering) *Using δ_f given in the previous example, we obtain the classical prefix rule:*

$$\frac{T \vdash \{\langle m, n\rangle\}_{pk}}{T \vdash \{m\}_{pk}}$$

The δ_c-IND-CCA Criterion If we consider an IND-CCA encryption scheme, then EDY is still a safe abstraction of the computational model as EDY is an extension of DY (formally, $T \vdash_{DY} m$ implies $T \vdash_{EDY} m$). However, EDY can still be a safe abstraction when considering an encryption scheme that is not IND-CCA. The encryption scheme has to be secure against a new criterion that uses δ_c, this criterion is called δ_c-IND-CCA.

δ_c-IND-CCA is defined as a criterion $(\mathcal{KG}, \mathcal{E}_{pk}, \mathcal{D}_{sk})$ similar to IND-CCA. The only difference is that in IND-CCA the decryption oracle does not apply to strings produced by the encryption oracle (stored in a mutable field mem), here it does not apply to strings that are in $mem' = \left\{\delta_c^n(s), s \in mem, n > 0\right\}$. As this set can be computed in polynomial time, the decryption oracle can be implemented using a PRTM.

It is easy to extend δ_c-IND-CCA to N-δ_c-IND-CCA and a trivial application of the partition theorem gives the following property:

Proposition 6.5 *Let $\mathcal{AE}$ be an asymmetric encryption scheme. Then for any $N > 0$, $\mathcal{AE}$ is $N - \delta_c$-IND-CCA iff $\mathcal{AE}$ is δ_c-IND-CCA.*

Example 6.6 (Block Ciphering) *Proof of the existence of a δ_c-IND-CCA encryption scheme for BC can be done as follows.*

Let bs be a positive integer representing a block size. Let us consider an IND-CCA encryption scheme $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ and a SYM-CPA encryption scheme $\mathcal{AE}' = (\mathcal{KG}', \mathcal{E}', \mathcal{D}')$. Then, $\mathcal{AE}''$ is the encryption scheme $(\mathcal{KG}, \mathcal{E}'', \mathcal{D}'')$ where $\mathcal{E}''(s, pk)$ is defined as:

$$\mathcal{E}''(s, pk) = \mathcal{E}(s_1 \cdot k_1, pk) \cdot \mathcal{E}'(s_2 \cdot k_2, k_1) \cdot \ldots \cdot \mathcal{E}'(s_n, k_{n-1})$$

when s is composed by blocks s_1 to s_n of size bs and keys k_i are freshly generated symmetric keys. The decoding algorithm allows to retrieve the original message by applying a similar modification.

This encryption scheme is not IND-CCA but is δ_c-IND-CCA. To prove that, let us assume that it is not the case. Then there exists an adversary $\mathcal{A}$ against this criterion. We use $\mathcal{A}$ to break Q-PAS (for Pattern Asymmetric Symmetric semantic security, this criterion represents joint security of asymmetric and symmetric encryption with patterns and is a restriction of Q-PASS): for this purpose, $\mathcal{A}$ is executed and his queries to oracles are simulated using IND-CCA oracles. Then, if $\mathcal{A}$ asks for the encryption of $s_1 \cdot \ldots \cdot s_n$, the new adversary asks for encryption of s_1, creation of a fresh challenge symmetric key SYM-CPA and so on. When $\mathcal{A}$ calls his decryption oracle, the argument m is not the prefix of any output of the new encryption oracle. Let us suppose that $m = m_1 \cdot \ldots \cdot m_n$. Then if m_1 has not been produced by the left-right asymmetric encryption oracle, it is possible to decrypt the whole message. Otherwise, let i be the minimal index such that m_i has not been produced by an encryption oracle. Then either m_i allows to win SYM-CPA/UNF or the m was not a correct encryption. Finally, if any m_i has been produced by an encryption oracle, then m is a prefix of the output of the new encryption oracle.

Soundness Result In the formal world, the definition of extended possible traces is exactly similar to the definition of possible traces. For a given protocol Π, this defines a set $etraces(\Pi)$. For computational semantics, the different exec algorithms are not modified.

The main result is stated in the following theorem:

Theorem 6.3 *Let Π be a protocol. Let $\mathcal{AE}$ be the encryption scheme used in Π. If $\mathcal{AE}$ is δ_c-IND-CCA then for any concrete adversary $\mathcal{A}_c$:*

$$Pr\big[\ \nexists t_f \in traces(\Pi) \cdot Exec(\mathcal{A}, \Pi, \theta) \in Concr(t_f, \theta)\big]\ \textit{is negligible}$$

The proof is close to the one of the previous soundness theorem, an adversary against the protocol can be transformed to an adversary against the encryption scheme as soon as the trace is NEDY (not Extended Dolev-Yao).

- Parsing is done as before except that messages in mem' cannot be decoded anymore. For this purpose, messages in mem' are stored as long as their formal equivalent to allow their parsing. Hence, it is possible to simulate the execution of the protocol.
- Messages that are NEDY are elements of the following grammar where m represents any message:

  $$n ::= N | \langle n, m\rangle | \langle m, n\rangle | \{n\}_{pk}$$

  If n is a nonce, winning the challenge can be done as before. If n is a pair, then one of its component is NEDY. Finally, if n is an encoding $\{n'\}_{pk}$ then the decryption oracle can be called to get n', otherwise $n \in mem'$ but any message in mem' is accessible in the trace and so is EDY.

Example 6.7 (Block Ciphering) *Verification of protocols with BC is possible when considering a bounded number of sessions (but the problem is in general NP-complete [CKRT03a]). Thus, it is possible to verify a protocol with this hypothesis in the formal world, if the protocol is safe (regarding for example trace properties), then it is safe in the computational world.*

Extensions Here, the weakening functions operate on messages/strings and produces a message/string. Although this hypothesis makes the results easier to understand, it is quite restrictive. This is why, we consider functions δ_c and δ_f which work on and produce sets of messages/strings. A similar soundness result can obtained in a straightforward way.

Example 6.8 (Extended BC) *Let us consider an extension of BC where the adversary is able to deduce a prefix as before but also a suffix. This is encoded in the following definition:*

$$\begin{aligned} \delta_f(\{\langle m, n\rangle\}_{pk}) &= \{\{m\}_{pk}, \{n\}_{pk}\} \\ \delta_c(\mathcal{E}(s.s', pk)) &= \{\mathcal{E}(s, pk), \mathcal{E}(s', pk)\} \end{aligned}$$

The EDY is modified with the two following inferences:

$$\frac{T \vdash \{\langle m, n\rangle\}_{pk}}{T \vdash \{m\}_{pk}} \qquad \frac{T \vdash \{\langle m, n\rangle\}_{pk}}{T \vdash \{n\}_{pk}}$$

Proof of the existence of a δ_c-IND-CCA encryption scheme for extended BC is presented here: Let bs be a positive integer representing a block size. Let us consider an IND-CCA encryption scheme $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$. Then, $\mathcal{AE}'$ is the encryption scheme $(\mathcal{KG}, \mathcal{E}', \mathcal{D}')$ where $\mathcal{E}'(s, pk)$ is defined as:

$$\mathcal{E}'(s, pk) = \mathcal{E}(s_1, pk) \cdot \mathcal{E}(s_2, pk) \cdot ... \cdot \mathcal{E}(s_n, pk)$$

when s is composed by blocks s_1 to s_n of size bs. The decoding algorithm allows to retrieve the original message by applying a similar modification.

Then, $\mathcal{AE}'$ is not an IND-CCA encryption scheme but is a δ_c-IND-CCA encryption scheme. It is not IND-CCA as an adversary can submit message 0^{2bs} and message 1^{2bs} to the encryption oracle. The oracle outputs $\{b^{2bs}\}_{pk} = \{b^{bs}\}_{pk}.\{b^{bs}\}_{pk}$ and so the adversary submits b^{bs} to the decryption oracles (this is possible as this message has not been produced by the encryption oracle) and obtains the value of bit b. The restriction made on the decryption oracle aims at forbidding such attacks.

If $\mathcal{AE}'$ is not a δ_c-IND-CCA encryption scheme, then there exists an adversary $\mathcal{A}$ against the criterion with δ_c. As usual, this adversary can be used to create an adversary against IND-CCA for algorithm $\mathcal{AE}$: oracles related to $\mathcal{A}$ are simulated using $\mathcal{AE}$ (this is trivial for encryption and possible for decryption because of the form of δ_c).

This approach works well for equational theories that do not modify encryption keys. This is not the case if we consider an encryption scheme with key commutation (usual when considering modular exponentiation). For any keys pk_1, pk_2, and any message m:

$$\left\{\{m\}_{pk_2}\right\}_{pk_1} = \left\{\{m\}_{pk_1}\right\}_{pk_2}$$

In this case, if a bit-string bs has been submitted to an encryption oracle that returned bs', then no encoding (with any other public key) of bs' can be submitted to the decryption oracle. Hence the set mem' cannot be computed in polynomial time.

6.3 Relating Symbolic and Computational Properties

In this section, we study computational soundness of symbolic properties. We mainly investigate two different classes of properties: trace properties like authentication and weak secrecy and strong secrecy properties like SecNonce or SecKey. More complex properties based on opacity (a symbolic version of computational indistinguishability) are examined in chapter 9.

6.3.1 Trace Properties

Let us first consider the case of trace properties. The computational version P_c of a trace property is given by a set of computational traces and the symbolic version P_f is given by a set of symbolic traces. We say that P_f is a *faithful abstraction* of P_c for protocol Π, if the probability that a concretization of a symbolic trace in P_f is not in P_c is negligible. In other words, the following probability has to be negligible for any adversary $\mathcal{A}$:

$$Pr[\exists t_f \in traces(\Pi) : t_f \in P_f \wedge Exec(\mathcal{A}, \Pi, \theta) \in Concr(t_f) \wedge Exec(\mathcal{A}, \Pi, \theta) \notin P_c].$$

In particular, faithfulness is always verified if the set of possible concretizations of traces from P_f is included in P_c which means that any concretization of a symbolic trace verifying the property in the symbolic world also has to verify the property in the computational setting.

The following proposition is a preservation result for faithful trace properties. It states that if the symbolic property is a faithful abstraction of the computational property and it is satisfied in the symbolic model then the concrete property is satisfied in the computational model. It has been applied to mutual authentication in [MW04c] in which there is also a longer discussion about symbolic/computational properties.

Proposition 6.6 *Let P_f be a symbolic property and P_c be a computational property. Let Π be a well-formed protocol that uses a PASS secure library. If P_f is a faithful abstraction of P_c for Π and if $\Pi \models_f P_f$, then $\Pi \models_c P_c$. In other words, if the protocol satisfies the property P_f in the symbolic model then it satisfies P_c in the computational model.*

Proof: This proposition is a consequence of theorem 6.1. Indeed, we have

$$\begin{aligned} Pr[Exec(\mathcal{A}, \Pi, \theta) \notin P_c] \;=\; & Pr[\exists t_f \in \mathsf{Traces}(\Pi) : t_f \in P_f \wedge \\ & \quad Exec(A, \Pi, \theta) \in Concr(t_f) \setminus P_c] + \\ & Pr[\exists t_f \in \mathsf{Traces}(\Pi) : t_f \notin P_f \wedge \\ & \quad Exec(A, \Pi, \theta) \in Concr(t_f)] + \\ & Pr[\nexists t_f \in \mathsf{Traces}(\Pi) : Exec(A, \Pi, \theta) \in Concr(t_f)] \\ =\; & \nu(\eta) + 0 + \nu'(\eta) \end{aligned}$$

where ν and ν' are negligible. ∎

Similar results also exist for protocols that only use a subset of the different cryptographic primitives proposed here. Moreover, it is also possible to generalize this result in order to handle hashing and the full computational model (see [Jan06] for details).

Let us now illustrate how this last proposition can be used. For this purpose, we consider two different properties: authentication and weak secrecy. These are trace properties hence the proposition applies directly whereas for more complex properties (e.g. SecNonce) an intermediate proof is needed.

Authentication

Let us first deal with authentication properties. These properties can be formulated as trace properties. Namely, given a protocol and a cryptographic library, there is a set of computational properties representing executions where authentication is verified in the computational setting. There also exists a set of symbolic traces which represent executions where authentication is verified in the symbolic world.

Proposition 6.7 *Let* Π *be a well-formed protocol that uses a PASS secure library. If* Π *satisfies an authentication property in the symbolic setting, then the same authentication property is verified in the computational world.*

Proof: For any of the authentication properties, it is sufficient to notice that if a computational execution abstracts on a symbolic trace where authentication is verified, then authentication is also verified in the computational setting. Hence the symbolic version of authentication is clearly a faithful abstraction of the computational version. Then it is possible to apply proposition 6.6. ■

Weak Secrecy

Weak secrecy is expressed as a trace property in section 6.1.3. Hence it is possible to use property 6.6 in order to deal with soundness of symbolic secrecy. The main result is that if a protocol ensures secrecy for nonce N in the symbolic setting, then weak secrecy of N is ensured in the computational setting. This is stated through the next proposition.

Proposition 6.8 *Let* Π *be a well-formed protocol that uses a PASS secure library. If* Π *satisfies secrecy for message* m *in the symbolic setting, then the same weak secrecy of* m *is verified in the computational world.*

Proof: The proof is exactly the same as for 6.7. It is sufficient to notice that if a computational execution abstracts on a symbolic trace where weak secrecy is verified, then weak secrecy is also verified in the computational setting. Then it is possible to apply proposition 6.6. ■

Weak secrecy correctly abstracts on symbolic secrecy. However this version of secrecy is too weak to be used in the computational setting. For example, it is possible that a protocol satisfies weak secrecy for a nonce N and that an adversary can deduce half of the bits of N. As the adversary is not able to build N entirely, there is no contradiction. If N is used for authentication, this is not a problem as the adversary has to submit the whole N in order to perform an attack. But if N is for example an encoding of a credit card number and a payment date, then if the adversary is able to recover the credit card information, this becomes a real problem. At this point, weak secrecy cannot guarantee anything on partial information leaks.

From a computational point of view, we would like to ensure that no information at all can be extracted from the secret nonce. This is captured by the SecNonce property: after executing the protocol, the adversary should not be able to distinguish the secret nonce value from a randomly sampled value with non-negligible probability.

6.3.2 Secrecy

Some properties cannot be expressed by a predicate over traces. In this situation, proposition 6.6 cannot be applied directly. Hence, simulatability approaches [BPW03a, CH04] are better suited to

handle directly these properties. However soundness of the symbolic model can be proven for some "non-trace" properties: this is usually the case for indistinguishability properties and the proofs have to be adapted from for example the proof of theorem 6.1. For trace properties, this theorem just have to be applied but for "non-traces" properties, we have to understand and modify the proof of this theorem. This is done for example in [CW05] for strong secrecy of nonces (i.e. the SecNonce property presented in section 6.1.3). Precisely, the main result of this paper is that if a nonce N is secret for a protocol Π in the symbolic setting, if the encryption scheme used to implement it is secure against IND-CCA and if the signature scheme is secure for UNF, then the SecNonce property for N in Π is verified. The proof proceeds by directly creating an adversary against the cryptographic primitives that uses the adversary against SecNonce.

This result can easily be extended to the case of protocols that involve asymmetric encryption, symmetric encryption and digital signature. However, as we want proofs to remain readable, we still consider in this section the simplest model possible: the ACM model where the only allowed cryptographic primitive is asymmetric encryption.

Strong Secrecy for Nonces: SecNonce

Note that here, the model is simpler than the one used in [CW05]. However the result given here cannot be seen as a direct consequence of the main result of this paper as we allow key emission. Moreover, the proof presented here can be adapted to handle the general computational model as well as other cryptographic primitives [Jan06].

Theorem 6.4 *Let Π be a protocol and N be a nonce. Let us suppose that the asymmetric encryption scheme used to implement this protocol is secure against IND-CCA. If protocol Π preserves secrecy of nonce N in the symbolic setting, then Π verifies SecNonce for nonce N.*

Proof: The proof of this theorem is very close to the proof of theorem 6.1. Let $\mathcal{A}$ be an adversary against SecNonce for protocol $\Pi = (R, IK)$ on nonce N. Let n be the number of nonces used by the protocol. Then we build an adversary $\mathcal{B}$ against n-PAT-IND-CCA. Adversary $\mathcal{B}$ is built in the same way as in the proof of theorem 6.1. Nonce N is trapped, hence two values bs_0 and bs_1 for N are randomly sampled. Adversary $\mathcal{B}$ uses $\mathcal{A}$ as a subroutine. Hence $\mathcal{B}$ has to simulate the protocol execution so that $\mathcal{A}$ can perform his attack. This attack allows $\mathcal{B}$ to find his challenge bit's value. Whenever $\mathcal{B}$ has to produce the concretization of an encryption of a message m containing N, $\mathcal{B}$ creates two patterns: pat_0 where N is instantiated with bs_0 and pat_1 where N is instantiated with bs_1. Then $\mathcal{B}$ uses his left-right encryption oracles on the pair $\langle pat_0, pat_1 \rangle$ and hence receives the encryption of (the concretization of) pat_b. As N is secret in the symbolic world, $\mathcal{B}$ cannot be asked to produce a message where N is unprotected (otherwise, $\mathcal{A}$ has produced a trace that is not possible in the symbolic setting and so another adversary would be able to break the cryptographic primitives using $\mathcal{A}$). Hence $\mathcal{B}$ perfectly simulates the execution of the protocol with adversary $\mathcal{A}$.

Finally, $\mathcal{B}$ gives to $\mathcal{A}$ the bit-strings bs_0 and bs_1 and $\mathcal{A}$ has to tell which bit-string was used in the protocol. $\mathcal{B}$ makes the same answer as $\mathcal{A}$ for his challenge bit. Therefore the experiment involving $\mathcal{A}$ against SecNonce and $\mathcal{B}$ against n-PAT-IND-CCA are the same (except for negligible probability if $\mathcal{B}$ has to reveal N). Thus Π verifies SecNonce for nonce N. ■

Secrecy for Keys

In this section, we introduce a definition for strong secrecy of keys. Weak secrecy of keys can be defined as weak secrecy of nonces: an adversary wins if he is able, after execution of the protocol, to output the bit-string value of the key. However, it is not possible to define strong secrecy for keys as in SecNonce. Indeed, strong – Real or Random – secrecy is too strong when applied to keys. Our basic criteria cannot ensure real or random secrecy of secret keys: let us consider for example the protocol constituted by the only action $\text{SEND}(\{A\}_k)$. Then secrecy of key k^{-1} is ensured in the symbolic setting as k^{-1} is not deducible from $\{A\}_k$. In the computational world, the IND-CCA and SYM-CPA criteria do not imply the encryption to be which-key concealing. An algorithm is which-key concealing if it is impossible to retrieve any information on the key from cipher-texts.

Otherwise, it is which-key revealing. This notion is formally detailed in the following paragraph. As the encryption can be which-key revealing, it is possible for the adversary to distinguish the key used to compute $\{A\}_k$ from a randomly generated key. Hence real or random secrecy for keys is not implied by IND-CCA as it was for nonces.

Which-Key Concealing An encryption scheme (symmetric or asymmetric) is said to be *which-key concealing* if it is impossible for an adversary to deduce from a cipher-text any information on the related key. The related game is an indistinguishability criterion: the adversary is either confronted to two encryptions using the same key or to two encryptions with different keys and has to tell if he thinks the encryptions where performed with the same key or with different keys. Let $\mathcal{AE}$ be an asymmetric encryption scheme composed of the three algorithms $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$. Let $\mathcal{A}_1, \mathcal{A}_2$ be a two-stage adversary, then the advantage of $\mathcal{A}_1, \mathcal{A}_2$ against which-key concealing (wk) is given by:

$$\begin{aligned}\mathbf{Adv}^{wk}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) \quad = \quad & Pr[(pk, sk) := \mathcal{KG}(\eta);\ bs := \mathcal{A}_1(\eta);\ \mathcal{A}_2(\mathcal{E}(bs, pk), \mathcal{E}(bs, pk)) = 1] \\ & -Pr[(pk, sk) := \mathcal{KG}(\eta); (pk', sk') := \mathcal{KG}(\eta);\ bs := \mathcal{A}_1(\eta);\ \mathcal{A}_2(\mathcal{E}(bs, pk), \mathcal{E}(bs, pk')) = 1]\end{aligned}$$

This advantage definition can easily be adapted to handle the case of symmetric encryption. Let $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme, then the advantage of $\mathcal{A}$ against wk is given by:

$$\begin{aligned}\mathbf{Adv}^{wk}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) \quad = \quad & Pr[k := \mathcal{KG}(\eta);\ bs := \mathcal{A}_1(\eta);\ \mathcal{A}_2(\mathcal{E}(bs, k), \mathcal{E}(bs, k)) = 1] \\ & -Pr[k := \mathcal{KG}(\eta); k' := \mathcal{KG}(\eta);\ bs := \mathcal{A}_1(\eta);\ \mathcal{A}_2(\mathcal{E}(bs, k), \mathcal{E}(bs, k')) = 1]\end{aligned}$$

An encryption scheme is which-key concealing if for any adversary $\mathcal{A}_1, \mathcal{A}_2$, the advantage of $\mathcal{A}_1, \mathcal{A}_2$ against wk is negligible. Our main claim is that classical security assumptions for encryption schemes do not ensure that the schemes are which-key concealing.

Proposition 6.9 *Let us suppose that there exists an asymmetric encryption scheme that is secure against IND-CCA. Then there exists an asymmetric encryption scheme that is secure against IND-CCA and not which-key concealing.*

Proof: Let $\mathcal{AE}$ be an asymmetric encryption scheme defined by algorithms $\mathcal{KG}$, $\mathcal{E}$ and $\mathcal{D}$. Then asymmetric encryption scheme $\mathcal{AE}'$ is defined as triple $(\mathcal{KG}', \mathcal{E}', \mathcal{D}')$ where:

- The key generation algorithm is unchanged: $\mathcal{KG}' = \mathcal{KG}$.
- Encryption is performed by appending the public key after the encryption made by $\mathcal{E}$:

$$\mathcal{E}'(bs, pk) = \mathcal{E}(bs, pk).pk$$

- The decryption algorithm is modified in order to match the previous encryption algorithm:

$$\mathcal{D}'(bs.pk, sk) = \mathcal{D}(bs, sk)$$

Then if asymmetric encryption scheme $\mathcal{AE}$ is secure against IND-CCA, $\mathcal{AE}'$ is also secure against IND-CCA but is also which-key revealing. ∎

The case of symmetric encryption is little more complicated to handle as it is not possible to paste the key after encryptions. Hence, instead of appending the key itself, we append a hash of this key that does not give any useful information to the adversary. An example of such hash can be the parity of the random coins used to generate the key (because parity of the key itself may not bring any useful information).

Therefore, classical indistinguishability criteria cannot ensure real-or-random secrecy for keys. The important point is not for the adversary to get "any" information on the key but to get some "usable" information. Thus strong secrecy for keys is defined as the impossibility for an adversary to have collected enough information on the key so that he can perform an attack against IND-CCA (for asymmetric encryption). The main advantage of this strong secrecy notion is that it allows composition of protocols: after executing a first protocol, the adversary did not get enough information to break IND-CCA. Hence the key can be used once more in a second protocol (as safety of the protocol relies on IND-CCA, cf section 6.2) and so on.

Strong Secrecy for Asymmetric Encryption Keys Let $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme. Let Π be a protocol and pk be a public key that occurs in Π. The SecKey property is defined as the following challenge: an adversary is composed of to two stages $\mathcal{A}_1$ and $\mathcal{A}_2$. Stage $\mathcal{A}_1$ is executed as a protocol adversary and then can get information on pk. Finally, in the second stage, $\mathcal{A}_2$ is confronted to the usual IND-CCA challenge and has to guess the value of the challenge bit. Formally, the advantage of $(\mathcal{A}_1, \mathcal{A}_2)$ is defined by:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{SK(\Pi,pk)}(\eta) \quad = Pr[\quad & \theta := init(\eta, R)\\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}_1, \theta, \Pi)\\ & \mathcal{A}_2(mem)/(\text{IND-CCA oracles for pk with b=1}) = 1]\\ -Pr[\quad & \theta := init(\eta, R)\\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}_1, \theta, \Pi)\\ & \mathcal{A}_2(mem)/(\text{IND-CCA oracles for pk with b=0}) = 1]\end{aligned}$$

An important point is that the adversary cannot use his IND-CCA decryption oracle on cipher-texts produced during the protocol execution. Technically, such cipher-texts can be produced using the related left-right encryption oracle but also by using the public key oracle. Hence the decryption oracle may not work on these cipher-texts.

Protocol Π verifies SecKey for key pk if the advantage of any adversary is negligible in η.

Proposition 6.10 *Let Π be a protocol and pk be a key that is used in Π. Let us suppose that the cryptographic library used to implement this protocol is safe. If protocol Π preserves secrecy of pk^{-1} in the symbolic setting, then Π verifies SecKey for key pk.*

Proof: The idea of the proof is the simple. Let $\mathcal{A}$ be an adversary against SecKey, we build an adversary $\mathcal{B}$ against N-PAT-IND-CCA which uses $\mathcal{A}$ as a subroutine. $\mathcal{B}$ simulate the execution of adversary $\mathcal{A}$ confronted to protocol Π. For this purpose, he first randomly chooses a subset of his keys that includes pk called challenge keys, these keys are supposed to be kept secret by the protocol. He then generates the necessary atoms and uses them to simulate the protocol, $\mathcal{B}$ uses his oracles to perform encryptions with his challenge keys. This is done in the same way as in theorem 6.1. Note that the left-right encryption oracles are always used with the same patterns as arguments. Indeed in this situation, patterns are used to ask for encryption of secret keys but the left-right aspect is not useful here. Adversary $\mathcal{B}$ is able to perfectly simulates the protocol execution, the only problem occurs if $\mathcal{B}$ is asked to send pk^{-1} without protecting it with a safe encryption. However as this cannot happen in the symbolic world, this can only happen in the computational setting with negligible probability. Thus after simulating the protocol execution, adversary $\mathcal{B}$ answers $\mathcal{A}_2$ queries to the IND-CCA oracle using his own oracles. At this point, we understand why the decryption oracle cannot be used on cipher-text from the protocol simulation (which may have been produced by the left-right encryption oracle). Finally $\mathcal{B}$ outputs the same result as $\mathcal{A}_2$.

If $\mathcal{B}$ correctly chose his challenge keys (and this happen with non-negligible probability), then the advantage of $\mathcal{A}_1, \mathcal{A}_2$ and the advantage of $\mathcal{B}$ are equal up to some negligible function (representing executions where $\mathcal{B}$ is asked to output pk^{-1}). Thus the SecKey property is verified for pk. ■

Strong Secrecy for Symmetric Encryption Keys The previous definition is easy to adapt to symmetric encryption. Let $\mathcal{SE}$ be a symmetric encryption scheme. Let Π be a protocol that uses a symmetric key k. We still consider two-stages adversaries. The advantage of adversary

$(\mathcal{A}_1, \mathcal{A}_2)$ is given by (for the IND part):

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{SK(\Pi,k)}(\eta) \quad = Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}_1, \theta, \Pi) \\ & \mathcal{A}_2(mem)/(\text{IND-CPA oracles for k with b=1}) = 1] \\ -Pr[\quad & \theta := init(\eta, R) \\ & (t, \theta, mem) := CMexec(\eta, \mathcal{A}_1, \theta, \Pi) \\ & \mathcal{A}_2(mem)/(\text{IND-CPA oracles for k with b=0}) = 1]\end{aligned}$$

Protocol Π verifies SecKey for key k if the advantage of any adversary is negligible in η.

Proposition 6.11 *Let Π be a protocol and k be a symmetric key that is used in Π. Let us suppose that the cryptographic library used to implement this protocol is safe. If protocol Π preserves secrecy of k in the symbolic setting, then Π verifies SecKey for key k.*

Part III

Extensions

Chapter 7

Modular Exponentiation

Contents

The problem Diffie-Hellman schemes for authenticated key exchange protocols (AKE) are cryptographic primitives that allow users communicating over an unreliable network to establish shared secret keys that may be used to achieve security goals like data confidentiality and data integrity. They are widely used in practice (e.g. kerberos, SSH, TLS, video conferencing, etc...). There are two main families of AKE protocols that differ in the primitives used to provide authentication: password-based or long-lived key-based. In any case, the Diffie-Hellman key exchange scheme is a standard component of authenticated key exchange protocols. The design of AKE protocols is a difficult and error prone task (see [BGH+93, DvOW92, PQ01]). Hence, the interest in providing automatic verification tools that can cope with protocols that include Diffie-Hellman schemes and other cryptographic primitives. Such verification methods have been recently developed within the symbolic model for security protocols (also called formal model) [CKRT03a, MS03, BB03].

In this chapter, we study soundness of the symbolic verification methods for protocols that involve Diffie-Hellman schemes and symmetric encryption, i.e. we want to extend previous computational soundness results to protocols that use Diffie-Hellman exponentiation in order to generate new keys. We do not consider other cryptographic primitives like asymmetric encryption or digital signature as this would make notations more complex.

Contributions We make two main contributions in this chapter.

A Proof of the Correctness of the Symbolic Model We consider a symbolic model that deals with protocols using Diffie-Hellman exponentiation as well as symmetric encryption, essentially the model of [CKRT03a]. We prove computational soundness of this symbolic model in

the passive setting. This result is established under the Decisional Diffie-Hellman assumption and provided that the symmetric encryption scheme is secure and the protocol satisfies additional conditions usually met in practice. Our result also applies to (static) group Diffie-Hellman protocols. The purpose of these protocols is to establish a session key that is shared between the members of the group. The security of group protocols seems to require adapted methods different from the two-party case (e.g.[PQ01, BCPQ01, PQ03, PQ04]). Concerning the cryptographic primitives, the protocols we consider include Diffie-Hellman schemes and symmetric encryption and can easily be adapted to include digital signature, asymmetric encryption or hash functions.

We only consider the passive case as there exist automatic ways to transform key agreement protocols that are secure in the passive setting into authenticated key agreement protocols secure against active adversaries. This can be done by using the compiler introduced by Katz and Yung in [KY03]. Hence our result is an extension of the classical Abadi-Rogaway result [AR00]. Note that this result is not a subcase of the active setting: both results are incomparable as we consider an equivalence relation instead of just trace properties.

A New Diffie-Hellman Assumption The *Computational Diffie-Hellman assumption (CDH for short)*, respectively, the *Decisional Diffie-Hellman assumption (DDH for short)*, are often taken as a basis for proving security of protocols that use Diffie-Hellman schemes. CDH refers to the assumption that it is computationally infeasible to compute g^{xy}, given g, g^x and g^y. And the DDH assumption refers to the computational indistinguishability of the two distributions (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^r), i.e. $(g, g^x, g^y, g^{xy}) \sim (g, g^x, g^y, g^r)$. These assumptions are natural for two-party protocols. With the advance of multi-cast communication and group protocols the Diffie-Hellman scheme has been extended to allow more than two participants. *Group CDH* (GCDH for short) [STW96] and *Group DDH* (GDDH for short) [BCP02] have been introduced to deal with multi-party settings. The GCDH assumption refers to the infeasibility of computing $g^{x_1 \cdots x_n}$ given g^E for any strict subset $E \subsetneq \{x_1, \cdots, x_n\}$. The GDDH assumption refers to the indistinguishability of $g^{x_1 \cdots x_n}$ and g^r given g^E for any strict subset $E \subsetneq \{x_1 \cdots x_n\}$. While it has been shown that the DDH and GDDH assumptions are equivalent [STW96, BCP02]; it is unknown whether GCDH and CDH are equivalent.

In this chapter, we introduce the *Dynamic CDH* and *Dynamic DDH* assumptions (DCDH and 3DH for short). In contrast to the group DH-variations, in DCDH the adversary is not asked to compute $g^{x_1 \cdots x_n}$ but he can choose to compute g^p (or distinguish in case of 3DH) for any polynomial p. There are restrictions on the form of p, for example p should not have been given to the adversary. Thus in the decisional variation, the adversary can query the real value of g^p for any polynomial p or submit p to a left-right oracle that returns, depending on the value of a challenge bit b, g^p or g^r, for a randomly chosen r. The adversary has to guess the value of b.

While we are not able to provide an equivalence result between CDH and DCDH, we show that the 3DH assumption is equivalent to the DDH assumption. This assumption turns out to be very helpful in proving soundness of the symbolic model when group key exchange protocols are considered.

Related work concerning computational soundness Although lots of recent works studied the link between the symbolic and computational models, most of them do not apply to protocols that include Diffie-Hellman key exchange. In fact, very few results concerning this class of protocols have been published. An exception is the work of Gupta and Shmatikov [GS05] that extends the works of [DDMP03, DDM+05]. Their approach is based on a Hoare-like logic for reasoning about security protocols. The logic has two different interpretations with respect to a symbolic semantics and with respect to computational (complexity theoretic) semantics. Computational soundness of the logic, which informally means that proofs in the symbolic logic are valid in the computational interpretation is proved in [DDM+05]. The work in [GS05] adapts this result to protocols that use DH-exponentiation and digital signature. This work is also carried in the static corruption and finite-session setting. Moreover, it considers two-party protocols where authentication uses CMA-signature. We do not known about automated verification tools for the proposed logic. Jonathan Herzog presents in [Her03, Her04] an abstract model for DH key exchange protocols

based on the Strand-Space Model [THG98]. In Herzog's work the symbolic adversary is extended with the capability of applying arbitrary polytime functions. He shows computational soundness of the symbolic model. Hence none of these related works consider the case of polynomials used in exponentiations, as in the Burmester-Desmedt protocol [BD94].

Outline of this Chapter The next section introduces the classical Diffie-Hellman problems. In section 2, we define the dynamic version and show its equivalence to the decisional Diffie-Hellman problem. Section 3 introduces symbolic messages and the related equivalence relation. Section 4 discusses the soundness of the symbolic semantics in the passive case. Its main result is computational soundness of the equivalence relation. We illustrate this result on the Burmester-Desmedt protocol and prove its security in the passive case. Using the compiler described by Katz and Yung in [KY03], it is possible to build an adapted version of this protocol that is secure in the active setting. This seems to be contradictory with [PQ04] where Pereira and Quisquater prove that it is impossible to build an authenticated key-agreement protocol that only uses modular exponentiation and that is secure in the active setting. However there is no contradiction because the Katz and Yung compiler adds digital signatures to ensure authentication whereas Pereira and Quisquater consider protocols where authentication is provided using modular exponentiation. The original Burmester-Desmedt protocol only uses modular exponentiation but it is insecure in the active setting, we prove its security in the passive case.

7.1 The Diffie-Hellman Assumption

For the remainder of this chapter, let η be the security parameter. Let $\mathbb{G}$ be a cyclic group of prime order q and let g be a generator of $\mathbb{G}$. The operation of the group applied to x and y is denoted by $x.y$. The exponentiation of an element x is denoted by x^n. The order q is assumed large, i.e. its number of digits is linear in η. We suppose that everyone knows g, $\mathbb{G}$ and q.

7.1.1 The Diffie-Hellman Key Exchange Protocol

Let us first consider the simplest form of Diffie-Hellman scheme. Two agents A and B want to create a shared secret value. Agent A randomly chooses an element x in $[1, q]$ and sends g^x to B. Agent B also chooses an element y in $[1, q]$ and sends g^y to A. Then A and B can both compute the shared value g^{xy}: A takes the value u she received from B and computes u^x, B received v from A and computes v^y.

$$\begin{array}{lcl} A \rightarrow B & : & g^x \\ B \rightarrow A & : & g^y \\ A \rightarrow B & : & \{N\}_{g^{xy}} \end{array}$$

If we consider the case of a passive adversary (eavesdropper), then the adversary knows g^x and g^y. However, it should be hard for him to compute the shared secret g^{xy} using g^x and g^y. Thus this secret can be used to encrypt future communications between A and B. The last line represents the first use of such encrypted messages.

7.1.2 Computational and Decisional Diffie Hellman Problems

There are two different flavors of Diffie-Hellman assumptions related to the previous protocol. The first one, called the computational assumption, states that it should be hard for any adversary to produce g^{xy} if he only knows g^x and g^y. The second one, called the decisional assumption, is a strengthened version of the computational assumption. Knowing g^x and g^y, it should be hard for any adversary to distinguish between g^{xy} and g^r for some random value r.

Formally, the *Computational Diffie-Hellman* (CDH) assumption is that for any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ defined thereafter is negligible.

$$\mathbf{Adv}^{CDH}(\mathcal{A}) = Pr\Big[\mathcal{A}(g^x, g^y) \rightarrow g^{xy}\Big| x, y \stackrel{R}{\leftarrow} [1,q]\Big]$$

However, this computational assumption does not immediately guarantee any secrecy property on g^{xy}. It may be feasible to compute the first bits of g^{xy} but infeasible to compute its whole representation. Thus, there exists a presumably stronger assumption: from g^x and g^y, it is impossible to get any information on the shared secret g^{xy}.

The *Decisional Diffie-Hellman* (DDH) assumption is that for any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ defined thereafter is negligible.

$$\mathbf{Adv}^{DDH}(\mathcal{A}) = Pr\Big[\mathcal{A}(g^x, g^y, g^{xy}) \rightarrow 1\Big| x, y \stackrel{R}{\leftarrow} [1,q]\Big] - Pr\Big[\mathcal{A}(g^x, g^y, g^r) \rightarrow 1\Big| x, y, r \stackrel{R}{\leftarrow} [1,q]\Big]$$

If this assumption holds, an adversary is not able to distinguish the shared secret from a random group element with non-negligible probability. This means that an adversary cannot extract a single bit of information on g^{xy} from g^x and g^y.

Using the Criterion Formalism The CDH and DDH assumptions can easily be related to two security criteria. CDH is associated to $\gamma_{CDH} = (\Theta, F, V)$ where Θ randomly generates x and y in $[1,q]$, oracle F returns g^x and g^y and the verifier V returns true only for g^{xy}. DDH is associated to $\gamma_{DDH} = (\Theta, F, V)$ where Θ randomly generates x, y, r in $[1,q]$ and a bit b. If $b = 1$ then F returns g^x, g^y and g^{xy}, else F returns g^x, g^y and g^r. The adversary has to guess the value of bit b, i.e. $V(x)$ is true iff $x = b$.

Proposition 7.1 *The CDH assumption holds iff γ_{CDH} is safe. The DDH assumption holds iff γ_{DDH} is safe.*

Proof: Let $\mathcal{A}$ be an adversary against CDH, then it is easy to build an adversary $\mathcal{A}'$ against γ_{CDH} such that:

$$Pr\big[\mathcal{A}(g^x, g^y) \rightarrow g^{xy}\big| x, y \stackrel{R}{\leftarrow} [1,q]\big] = Pr[\mathbf{G}_{\mathcal{A}'}^{\gamma_{CDH}}(\eta) = true]$$

As $PrRand^{\gamma_{CDH}}$ is negligible, if γ_{CDH} is safe then the CDH assumption holds. The converse is immediate as it is easy to build from $\mathcal{A}'$ an adversary $\mathcal{A}$ against CDH such that the previous equality is still true.

Let $\mathcal{A}$ be an adversary against DDH, then it is easy to build an adversary $\mathcal{A}'$ against γ_{DDH} such that:

$$Pr\big[\mathcal{A}(g^x, g^y, g^{xy}) \rightarrow 1\big| x, y \stackrel{R}{\leftarrow} [1,q]\big] = Pr[\mathbf{G}_{\mathcal{A}'}^{\gamma_{DDH}}(\eta) = true | b = 1]$$

$$Pr\big[\mathcal{A}(g^x, g^y, g^r) \rightarrow 0\big| x, y, r \stackrel{R}{\leftarrow} [1,q]\big] = Pr[\mathbf{G}_{\mathcal{A}'}^{\gamma_{DDH}}(\eta) = true | b = 0]$$

Hence, we get that,

$$Pr\big[\mathcal{A}(g^x, g^y, g^{xy}) \rightarrow 1\big| x, y \stackrel{R}{\leftarrow} [1,q]\big] + Pr\big[\mathcal{A}(g^x, g^y, g^r) \rightarrow 0\big| x, y, r \stackrel{R}{\leftarrow} [1,q]\big] = 2Pr[\mathbf{G}_{\mathcal{A}'}^{\gamma_{DDH}}(\eta) = true]$$

Then as $PrRand^{\gamma_{DDH}} = 1/2$, we get

$$\mathbf{Adv}^{DDH}(\mathcal{A}) = \mathbf{Adv}_{\mathcal{A}'}^{\gamma_{DDH}}(\eta)$$

Therefore, if γ_{DDH} is safe then the DDH assumption holds. The converse is still true as it is also possible to build $\mathcal{A}'$ from $\mathcal{A}$. ■

Hence, we properly defined the classical Diffie-Hellman assumption using our game formalism. An immediate property is that the decisional version of the assumption is stronger than the computational one.

Proposition 7.2 *Criterion γ_{DDH} is safe implies that criterion γ_{CDH} is safe. Thus if the DDH assumption is valid then the CDH assumption is also verified.*

Proof: Let $\mathcal{A}$ be an adversary against the CDH assumption. Then adversary $\mathcal{B}$ against DDH is designed by the following:

Adversary $\mathcal{B}(X, Y, Z)$:
if $Z = \mathcal{A}(X, Y)$ **then return** 1
else return 0

The advantage of adversary $\mathcal{B}$ is given by:

$$\begin{aligned}
\mathbf{Adv}^{DDH}(\mathcal{B}) &= Pr\left[\mathcal{B}(g^x, g^y, g^{xy}) \rightarrow 1 \middle| x, y \stackrel{R}{\leftarrow} [1, q]\right] - Pr\left[\mathcal{B}(g^x, g^y, g^r) \rightarrow 1 \middle| x, y, r \stackrel{R}{\leftarrow} [1, q]\right] \\
&= Pr\left[\mathcal{A}(g^x, g^y) \rightarrow g^{xy} \middle| x, y \stackrel{R}{\leftarrow} [1, q]\right] - Pr\left[\mathcal{A}(g^x, g^y) \rightarrow g^r \middle| x, y, r \stackrel{R}{\leftarrow} [1, q]\right] \\
&= \mathbf{Adv}^{CDH}(\mathcal{A}) - \frac{1}{q}
\end{aligned}$$

The number of elements in $\mathbb{G}$ is supposed large hence $1/q$ is negligible. As the DDH assumption holds, the advantage of $\mathcal{B}$ is negligible. Hence the advantage of $\mathcal{A}$ against CDH is also negligible and the CDH assumption holds. ∎

7.2 Extending the Assumptions

7.2.1 Dynamic Diffie-Hellman

The CDH and DDH assumptions are not general enough to easily apply to dynamic group protocols. Thus, the Group Decisional Diffie-Hellman problem (GDDH) has been studied and proven equivalent to DDH [STW96, BCP02]. In GDDH, there are α variables x_1 to x_α for which values are randomly generated. The adversary has access to $g^{\prod_{i \in I} x_i}$ for any strict subset I of $[1, \alpha]$, then he has to distinguish $g^{\prod_{i \in [1,\alpha]} x_i}$ from a random group element. However this variant is not really convenient if for example one wants to prove that assuming DDH, distributions $(g^x, g^{x.y+y}, g^y)$ and $(g^x, g^{x.y+y}, g^r)$ are computationally equivalent. Thus we introduce a new extension of DDH which we call the *Dynamic Decisional Diffie-Hellman* (3DH) assumption. This new assumption extends GDDH in three ways: the adversary may ask for multiple challenges, not only $g^{\prod_{i \in [1,\alpha]} x_i}$; the target challenges can be different from $g^{\prod_{i \in [1,\alpha]} x_i}$; the adversary may ask for linear combinations of group elements, e.g. $g^{2x_1 - x_2}$. All these extensions are used by the Burmester-Desmedt protocol [BD94]. We also introduce a similar generalization for CDH, this generalisation is called *Dynamic Computational Diffie-Hellman*. We do not use our criterion formalism to describe the new assumptions. Although this would be possible, as we do not intend to use the criterion partition theorems, we rather stick with a more classical formalism.

Let X be a countable set of variables. A *monomial* is a product of *distinct* variables. Hence it can be represented by a finite subset of X. The order of a monomial m is the number of its variables and is denoted by $ord(m)$. We consider *polynomials* that are linear combination of monomials. It is important to notice that x^2 is not allowed. This restriction cannot be withdrawn easily because as stated in [BDZ03], the indistinguishability of $(g^x, g^{x.x})$ and (g^x, g^r) under the DDH assumption is an open problem.

Dynamic Computational Diffie Hellman The DCDH game generalizes CDH: the adversary has access to an oracle that given a polynomial p outputs g^p. At the end of the game, the adversary chooses a polynomial q and has to output g^q.

Formally, polynomials are represented by vectors that give the coefficients of the different monomials. The game is indexed by the different monomials m_1 to m_n which the adversary has access to. First, some values are randomly sampled in $\mathbb{Z}_q$ for all the variables that occur in the

monomials $(m_i)_i$. These variables are denoted by x_1 to x_α. Then the values for the different monomials are computed and stored in vector $\vec{m}$. The adversary is executed, it can access one oracle. This oracle takes as argument a vector $\vec{v}$ representing coefficients of monomials. The computation of the polynomial is given by the scalar product of $\vec{v}$ and $\vec{m}$ and the oracle outputs $g^{\vec{v}.\vec{m}}$. Finally, the adversary has to output a group element d and its challenge polynomial $\vec{w}$. The game is a success if d corresponds to the group element related to $\vec{w}$. Of course, there is a restriction so that the game is not too easy to win: the challenge polynomial $\vec{w}$ has to be linearly independent from the arguments of the oracle.

Game $G_{\mathcal{A}}(m_1, ..., m_n)$:
  for i **from** 1 **to** α
    $x_i \xleftarrow{R} \mathbb{Z}_q$
  $\vec{m} := (m_1[(x_i)_i], ..., m_n[(x_i)_i])$
  $d, \vec{w} := \mathcal{A}/\lambda\vec{v}.g^{\vec{v}.\vec{m}}$,
  return $d = g^{\vec{w}.\vec{m}}$

Then the advantage of an adversary $\mathcal{A}$ is given by:

$$\mathbf{Adv}_{\mathcal{A}}^{G(m_1,...,m_n)} = Pr\big[G_{\mathcal{A}}^1(m_1, ..., m_n) = 1\big]$$

An immediate result concerning DCDH is that if the advantage of any adversary against DCDH is negligible, then the advantage of any adversary against CDH is also negligible.

Proposition 7.3 *For every adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that:*

$$\mathbf{Adv}_{\mathcal{A}}^{CDH} = \mathbf{Adv}_{\mathcal{B}}^{G(x_1,x_2,x_1x_2)}$$

Proof: Adversary $\mathcal{B}$ can easily be built using $\mathcal{A}$:

Adversary $\mathcal{B}/\mathcal{O}$:
  $d := \mathcal{A}\big(\mathcal{O}(1,0,0), \mathcal{O}(0,1,0)\big)$
  return $d, (0,0,1)$

Adversary $\mathcal{B}$ makes requests for g^{x_1} and g^{x_2} using his oracle with arguments $(1,0,0)$ and $(0,1,0)$. He gives the so obtained values to $\mathcal{A}$ and returns the output of $\mathcal{A}$ as the value of $g^{x_1x_2}$. ■

Dynamic Decisional Diffie-Hellman The 3DH assumption considers an adversary that can be faced to two different games. As usual in decisional assumptions, the adversary has to guess against which game he is playing. For this purpose he has access to two oracles. These oracles use randomly sampled values (in $\mathbb{Z}_q$) for a finite subset $(x_1, ..., x_\alpha)$ of X.

The first oracle behaves in the same way in the two different games, it takes as argument a polynomial p over $(x_i)_i$ (represented by a vector of coefficients in $\mathbb{Z}_q$) and returns $g^{p(x_1,...,x_\alpha)}$. In the first game, the second oracle behaves in the exact same way as the first oracle. However, in the second game, this second oracle still takes as argument a polynomial p but it returns g^r for some randomly sampled r in $\mathbb{Z}_q$.

Without restrictions, this game would be easy to win: an adversary can first submit a polynomial to the first oracle then submit it to the second oracle. If the two results are the same, the adversary knows with high probability that he is in the first game. Otherwise, the adversary knows for sure that he is in the second game. Hence we add a restriction over possible queries: the set of polynomials submitted to the oracles has to be linearly independent (over $\mathbb{Z}_q$).

It is easy to see that this new 3DH assumption implies the DDH assumption: an adversary can ask for g^{x_1} and g^{x_2} to his first oracle, then he queries the second oracle with polynomial x_1x_2 and receives either $g^{x_1x_2}$ or g^r. Finally the adversary has to deduce in which situation he is.

Formalization We write $\mathcal{A}/\lambda x.F(x), \lambda x.F'(x)$ to denote an adversary $\mathcal{A}$ that can access two oracles, The first oracle takes as input x and returns $F(x)$ whereas the second one returns $F'(x)$.

Let n be a positive integer polynomially bounded in the security parameter η. Let m_1 to m_n be n different monomials. In game $G(m_1, ..., m_n)$, values in $\mathbb{Z}_q$ are randomly generated for the α different variables occurring in the monomials. Then a vector $\vec{m}$ of size n is computed, this vector contains the integers values for the different monomials. The adversary has to distinguish between two situations: in the first one, he has access to two oracles that behave in the same way. They take as argument a vector $\vec{v}$ of integers of length n and returns $g^{\vec{v}.\vec{m}}$. In the second situation, the first oracle behaves as previously and the second one returns a random group element. However, there is one restriction concerning the requests made by $\mathcal{A}$. Let V_1 be the set of vectors asked to the first oracle and V_2 be the set of vectors submitted to the second oracle, then $V_1 \cup V_2$ has to be a family of linearly independent vectors (over $\mathbb{Z}_q$), i.e. requests are independent one from another. In particular, an adversary cannot submit the same vector to both oracles (as this is a trivial dependency). He cannot even submit the same vector to the same oracle twice. It is interesting to see that if the adversary was able to call the second oracle twice with the same vector, he would obtain the same result in the first game and two different results in the second one (as for each call to the second oracle, a new randomly sampled group element is returned), thus the game would be easy to win.

Game $G^1_{\mathcal{A}}(m_1, ..., m_n)$ and game $G^0_{\mathcal{A}}(m_1, ..., m_n)$ are formally defined by:

Game $G^1_{\mathcal{A}}(m_1, ..., m_n)$:
for i **from** 1 **to** α
$x_i \xleftarrow{R} \mathbb{Z}_q$
$\vec{m} := (m_1[(x_i)_i], ..., m_n[(x_i)_i])$
$d := \mathcal{A}/\lambda\vec{v}.g^{\vec{v}.\vec{m}}, \lambda\vec{v}.g^{\vec{v}.\vec{m}}$
return d

Game $G^0_{\mathcal{A}}(m_1, ..., m_n)$:
for i **from** 1 **to** α
$x_i \xleftarrow{R} \mathbb{Z}_q$
$\vec{m} := (m_1[(x_i)_i], ..., m_n[(x_i)_i])$
$d := \mathcal{A}/\lambda\vec{v}.g^{\vec{v}.\vec{m}}, \lambda\vec{v}.g^{r}$ where $r \xleftarrow{R} \mathbb{Z}_q$
return d

The first oracle is called the *real oracle* whereas the second is called the *RR oracle* The advantage of an adversary $\mathcal{A}$ is given by:

$$\mathbf{Adv}_{\mathcal{A}}^{G(m_1,...,m_n)} = Pr[G^1_{\mathcal{A}}(m_1, ..., m_n) = 1] - Pr[G^0_{\mathcal{A}}(m_1, ..., m_n) = 1]$$

The main result concerning 3DH is that it is equivalent to DDH. In a first proposition, we detail how an advantage against G can be reduced into multiple advantages against DDH.

Proposition 7.4 *Let Γ be a polynomially bounded list of monomials and let α be the number of different variables that are used in Γ. We suppose that if $m_1 m_2$ appears in Γ, then m_1 and m_2 also appear in Γ. If $\mathcal{A}$ is an adversary against $G(\Gamma)$, then there exist $|\Gamma| - \alpha$ adversaries $\mathcal{B}_i$ against DDH such that:*

$$\mathbf{Adv}_{\mathcal{A}}^{G(\Gamma)} = 2 \sum_{i=1}^{|\Gamma|-\alpha} \mathbf{Adv}_{\mathcal{B}_i}^{DDH}$$

Proof: This proof uses four intermediate lemmas. The first lemma tells us that in a finite field (with q elements), the number of solutions for a linear system of n independent equations among α variables is $q^{\alpha-n}$.

Lemma 7.1 *Let q be a prime. Let us consider a linear system of n equations implying α variables in $\mathbb{Z}_q$.*

$$\forall i \in [1, n], \sum_{j=1}^{\alpha} v_{i,j}.x_j = a_i$$

If vectors $\vec{v}_i = (v_{i,1}, ..., v_{i,\alpha})$ are linearly independent over $\mathbb{Z}_q$, then the number of solutions of the system is given by:

$$\left|\left\{\vec{x} \in \mathbb{Z}_q^{\alpha} | \forall i \in [1, n], \sum_{j=1}^{\alpha} v_{i,j}.x_j = a_i\right\}\right| = q^{\alpha-n}$$

Proof: This proof can be done using an induction on the number α of variables.

1. If α equals 1, then linear independence implies that there is at most one equation $v.x = a$ Let us first consider that there are no equations, then the number of solution is q:
$$\left|\left\{x \in \mathbb{Z}_q\right\}\right| = q$$

2. If α equals 1 and there is one equation. As $\mathbb{Z}_q$ is a field, this equation has exactly one solution $x = v^{-1}.a$. Hence,
$$\left|\left\{x \in \mathbb{Z}_q | v.x = a\right\}\right| = 1$$

3. In the case of $\alpha + 1$ variables, let us consider the first equation
$$\sum_{j=1}^{\alpha} v_{1,j}.x_j = a_1$$
Linear independence implies that there exists a coefficient k such that $v_{1,k}$ is different from 0. Then x_k is given by:
$$x_k = v_{1,k}^{-1}.(a - \sum_{j \neq k} v_{1,j}.x_j)$$
This substitution is used to transform the $n-1$ other equations into equations where x_k does not appear any more. The new system of equation is:
$$\forall i \in [2,n], \sum_{j \neq k}(v_{i,j} - v_{1,k}^{-1}.v_{1,j}).x_j = a_i + v_{1,k}^{-1}.a_1$$
Linear independence of the first system implies linear independence for this new system. As this system only involves $\alpha - 1$ equations, the induction hypothesis holds hence:
$$\left|\left\{x_1, ..., x_{k-1}, x_{k+1}, ..., x_\alpha \in \mathbb{Z}_q^{\alpha-1} | \forall i \in [2,n], \sum_{j \neq k}(v_{i,j} - v_{1,k}^{-1}.v_{1,j}).x_j = a_i + v_{1,k}^{-1}.a_1\right\}\right| = q^{(\alpha-1)-(n-1)}$$
Then for each solution of the new system, there is only one possibility for x_k. Hence the number of solutions of both systems are the same thus:
$$\left|\left\{\vec{x} \in \mathbb{Z}_q^{\alpha} | \forall i \in [1,n], \sum_{j=1}^{\alpha} v_{i,j}.x_j = a_i\right\}\right| = q^{\alpha-n}$$

■

Using the first lemma, it is possible to relate the number of solutions of two closely linked systems.

Lemma 7.2 *Let q be a prime. Let us consider n (resp. m) vectors of $\mathbb{Z}_q^{\alpha}$ denoted by $\vec{v}_i$ (resp. $\vec{w}_j$) such that all these vectors are linearly independent. Then*
$$\begin{aligned} & \left|\left\{\vec{x} \in \mathbb{Z}_q^{\alpha} \middle| \begin{array}{c} \forall i \in [1,n],\ \vec{v}_i.\vec{x} = a_i \\ \forall j \in [1,m],\ \vec{w}_j.\vec{x} = b_j \end{array} \right\}\right|.q^m \\ = & \left|\left\{\vec{x} \in \mathbb{Z}_q^{\alpha}, \vec{r} \in \mathbb{Z}_q^{m}, \middle| \begin{array}{c} \forall i \in [1,n],\ \vec{v}_i.\vec{x} = a_i \\ \forall j \in [1,m],\ r_j = b_j \end{array} \right\}\right| \end{aligned}$$

Proof: Linear independence allows us to apply lemma 7.1 twice: First,
$$\left|\left\{\vec{x} \in \mathbb{Z}_q^{\alpha} \middle| \begin{array}{c} \forall i \in [1,n],\ \vec{v}_i.\vec{x} = a_i \\ \forall j \in [1,m],\ \vec{w}_j.\vec{x} = b_j \end{array} \right\}\right| = q^{\alpha-(n+m)}$$

Then on the second set of solutions, we first remark that:

$$\left|\left\{\vec{x}\in\mathbb{Z}_q^\alpha,\vec{r}\in\mathbb{Z}_q^m,\left|\begin{array}{c}\forall i\in[1,n],\ \vec{v}_i.\vec{x}=a_i\\ \forall j\in[1,m],\ r_j=b_j\end{array}\right.\right\}\right| = \left|\left\{\vec{x}\in\mathbb{Z}_q^\alpha,\left|\forall i\in[1,n],\ \vec{v}_i.\vec{x}=a_i\right.\right\}\right| = q^{\alpha-n}$$

The conclusion is immediate from here. ■

An immediate corollary of this is that if $\vec{X}$ denotes a random variable over $\mathbb{Z}_q^\alpha$ (with uniform probability) and $\vec{R}$ denotes a random variable over $\mathbb{Z}_q^m$ (also with uniform probability). Then with the same hypothesis than in the previous proposition,

$$Pr\left[\begin{array}{c}\forall i\in[1,n],\ \vec{v}_i.\vec{X}=a_i\\ \forall j\in[1,m],\ \vec{w}_j.\vec{X}=b_j\end{array}\right] = Pr\left[\begin{array}{c}\forall i\in[1,n],\ \vec{v}_i.\vec{X}=a_i\\ \forall j\in[1,m],\ R_j=b_j\end{array}\right] = \frac{1}{q^{n+m}}$$

Using this result, we now prove our result for a base case where all the monomials are variables.

Lemma 7.3 *Let x_1 to x_α be α different variables. Let $\mathcal{A}$ be an adversary against $G(x_1,...,x_\alpha)$ then*

$$\mathbf{Adv}_{\mathcal{A}}^{G(x_1,...,x_\alpha)} = 0$$

Proof: Let $\mathcal{A}$ be an adversary. Let us consider that $\mathcal{A}$ calls the first (resp. second) oracle n (resp. m) times with arguments $\vec{v}_i$ (resp. $\vec{w}_j$) and receives as output g^{a_i} (resp. g^{b_j}). Then as the families of vectors $\vec{v}$ and $\vec{w}$ are linearly independent, the probabilities of producing such observations with such arguments are the same in the two possible games. Therefore, we can compute the advantage of $\mathcal{A}$:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{G(m_1,...,m_\alpha)} &= Pr[G_{\mathcal{A}}^1(m_1,...,m_\alpha)=1] - Pr[G_{\mathcal{A}}^0(m_1,...,m_\alpha)=1]\\ &= \sum_u Pr[G_{\mathcal{A}}^1(m_1,...,m_\alpha)=1|\mathcal{A}\text{ observed }u].Pr[u]\\ &\quad -\sum_u Pr[G_{\mathcal{A}}^0(m_1,...,m_\alpha)=1|\mathcal{A}\text{ observed }u].Pr[u]\\ &= \sum_u Pr[u].\big(Pr[G_{\mathcal{A}}^1(m_1,...,m_\alpha)=1|\mathcal{A}\text{ observed }u]\\ &\quad -Pr[G_{\mathcal{A}}^0(m_1,...,m_\alpha)=1|\mathcal{A}\text{ observed }u]\big)\\ &= 0\end{aligned}$$

■

The previous lemma is useful to initialize our induction argument. The next lemma allows to perform the induction itself.

Lemma 7.4 *Let m_1 to m_α be α monomials which set of variables is X_m. Let x_1 and x_2 be two different variables from X_m and let $x_{1,2}$ be a variable that is not in X_m. Let m'_1 to m'_α denote monomials such that m'_i is monomial m_i where x_1x_2 is replaced by $x_{1,2}$.*

For any adversary $\mathcal{A}$ against $G(m_1,...,m_\alpha)$, there exists an adversary $\mathcal{B}$ against DDH such that:

$$\mathbf{Adv}_{\mathcal{A}}^{G(m_1,...,m_\alpha)} = 2\mathbf{Adv}_{\mathcal{B}}^{DDH} + \mathbf{Adv}_{\mathcal{A}}^{G(m'_1,...,m'_\alpha)}$$

Proof: Adversary $\mathcal{B}$ is designed to play against DDH by using $\mathcal{A}$. He uses his DDH challenges to answer requests of $\mathcal{A}$. First adversary $\mathcal{B}$ associates a group element g_i to each monomial m_i, this is done by function F that outputs a vector of values. The i^{th} element of the vector corresponds to applying the following function to m_i:

$$\begin{array}{lcll} m & \rightarrow & exp(Z, m\setminus\{x_1,x_2\}) & \text{if } x_1,x_2\in m\\ m & \rightarrow & exp(X, m\setminus\{x_1\}) & \text{else if } x_1\in m\\ m & \rightarrow & exp(Y, m\setminus\{x_2\}) & \text{else if } x_2\in m\\ m & \rightarrow & exp(g, m) & \text{otherwise}\end{array}$$

Where exp is the exponentiation function defined by $exp(u,m) = u^{\prod_{x\in m} x}$. Then adversary $\mathcal{B}$ is given by:

Adversary $\mathcal{B}(X,Y,Z)$:
$\forall i \geq 3,\ x_i \xleftarrow{R} \mathbb{Z}_q$
$b \xleftarrow{R} [0,1]$
$g_1, ..., g_\alpha := F(X,Y,Z,x_3,..,x_n)$
$d := \mathcal{A}/\lambda\vec{v}.\prod_{i=1}^{\alpha} g_i^{\vec{v}[i]}$
$\lambda\vec{v}$.**if** b **then** $\prod_{i=1}^{\alpha} g_i^{\vec{v}[i]}$ **else** g^r
return $b = d$

The execution of $\mathcal{B}$ with arguments $(g^{x_1}, g^{x_2}, g^{x_1x_2})$ is the same as the game that involves $\mathcal{A}$ against $G(m_1,...,m_\alpha)$ whereas the execution of $\mathcal{B}$ with arguments $(g^{x_1}, g^{x_2}, g^{x_{1,2}})$ is the same as the game that involves $\mathcal{A}$ against $G(m'_1,...,m'_\alpha)$. The advantage of $\mathcal{B}$ is given by:

$$\begin{aligned} 2\mathbf{Adv}_{\mathcal{B}}^{DDH} &= 2Pr\left[\mathcal{B}(g^{x_1}, g^{x_2}, g^{x_1x_2}) = 1\right] - 2Pr\left[\mathcal{B}(g^{x_1}, g^{x_2}, g^{x_{1,2}}) = 1\right] \\ &= 2\left(Pr\left[\mathcal{B}(g^{x_1}, g^{x_2}, g^{x_1x_2}) = 1\right] - 1\right) - 2\left(Pr\left[\mathcal{B}(g^{x_1}, g^{x_2}, g^{x_{1,2}}) = 1\right] - 1\right) \\ &= \mathbf{Adv}_{\mathcal{A}}^{G(m_1,...,m_\alpha)} - \mathbf{Adv}_{\mathcal{A}}^{G(m'_1,...,m'_\alpha)} \end{aligned}$$

Moreover as $(m_i)_i$ is a family of linearly independent monomials, $(m'_i)_i$ is also a family of linearly independent monomials (because $x_{1,2}$ is a *fresh* variable). ■

Finally, proposition 7.4 can be proven by iterating this last lemma. For this purpose, we introduce $ord(\Gamma)$ which is defined as the sum of the order of monomials in Γ and proceed using an induction on $ord(\Gamma) - |\Gamma|$.

1. If $ord(\Gamma) - |\Gamma|$ equals 0, then all the monomials have order 1 and thus the number of variables α is equal to Γ. In this situation, lemma 7.3 applies and gives us that for any adversary $\mathcal{A}$,
$$\mathbf{Adv}_{\mathcal{A}}^{G(\Gamma)} = 0$$

2. Else $ord(\Gamma)$ is strictly greater than $|\Gamma|$ thus there exists a monomial m in Γ whose order is strictly greater than 1. Let x_1 and x_2 be two variables from m. Let $x_{1,2}$ be a fresh variable and let m'_1 to m'_α denote monomials m_i where x_1x_2 is replaced by $x_{1,2}$. The list of monomials m'_1 to m'_α is denoted by Γ'. Lemma 7.4 applies and hence,

   If $\mathcal{A}$ is an adversary against $G(m_1,...,m_\alpha)$, there exists an adversary $\mathcal{B}_{|\Gamma|-\alpha}$ against DDH such that:
$$\mathbf{Adv}_{\mathcal{A}}^{G(\Gamma)} = 2\mathbf{Adv}_{\mathcal{B}_{|\Gamma|-\alpha}}^{DDH} + \mathbf{Adv}_{\mathcal{A}}^{G(\Gamma')}$$

   The induction hypothesis applies on Γ' (as the sum of orders have strictly decreased). Hence there exists $|\Gamma'| - (\alpha+1)$ adversaries against DDH denoted by $\mathcal{B}_1$ to $\mathcal{B}_{|\Gamma'|-(\alpha+1)}$ such that:
$$\mathbf{Adv}_{\mathcal{A}}^{G(\Gamma')} = 2\sum_{i=1}^{|\Gamma'|-(\alpha+1)} \mathbf{Adv}_{\mathcal{B}_i}^{DDH}$$

   Moreover Γ' and Γ have the same number of elements hence we finally get that:
$$\mathbf{Adv}_{\mathcal{A}}^{G(\Gamma)} = 2\sum_{i=1}^{|\Gamma|-\alpha} \mathbf{Adv}_{\mathcal{B}_i}^{DDH}$$

■

Let us consider the case of α different variables. Then let us suppose that Γ contains all the monomials using the α variables. The number of such monomials is $2^\alpha - 1$. Hence, it is impossible

to have a polynomial (in η) number of variables, as Γ would contain an exponential number of elements (and so the game would not be polynomial time anymore). Even when DDH hold, proposition 7.4 cannot be used to conclude that 3DH holds. As we do not want to consider an exponential number of monomials but we want to have a polynomial number of distinct variables, a solution is to bound the degree of the monomials independently of η.

Corollary 7.1 *Let P be a polynomial. Let x_1 to $x_{P(\eta)}$ be $P(\eta)$ different variables and m be an integer. Let Γ denote the list of monomials using x_i whose orders are lower than m. If the DDH assumption holds, then the advantage of any adversary $\mathcal{A}$ against $G(\Gamma)$ is negligible.*

In the following, we use a bounded variant of 3DH. We limit ourselves to the case of n distinct variables where n does not depend on the security parameter. Therefore, the number of possible monomials is bounded and 3DH is equivalent to DDH. Let us call $3DH_n$ this game. Then it is immediate using the previous proposition that $3DH_n$ is implied by DDH. Moreover, we call $DCDH_n$ the computational variant of $3DH_n$.

Proposition 7.5 *If the DDH assumption holds, then both $3DH_n$ and $DCDH_n$ hold.*

Proof: $3DH_n \Rightarrow DCDH_n$: Let $\mathcal{A}$ be an adversary against $DCDH_n$. Then $\mathcal{B}$ is the adversary against $3DH_n$ defined by:

Adversary $\mathcal{B}/\mathcal{O}_S, \mathcal{O}_C$
$(d, \vec{w}) := \mathcal{A}/\lambda\vec{v}.\mathcal{O}_S(\vec{v})$
$d' := \mathcal{O}_C(\vec{w})$
return $d = d'$

Adversary $\mathcal{B}$ uses his real oracles to answer queries made by $\mathcal{A}$. Finally, $\mathcal{A}$ claims he has found the value d of g^p for some polynomial p independent from polynomials used in previous queries. Thus $\mathcal{B}$ can use his RR oracle to obtain either g^p or g^r. If this value is equal to the output of $\mathcal{A}$, $\mathcal{B}$ outputs 1, else $\mathcal{B}$ outputs 0. The advantage of $\mathcal{B}$ is given by:

$$\mathbf{Adv}_{\mathcal{B}}^{G(m_1,...,m_n)} = Pr[G^1_{\mathcal{B}}(m_1, ..., m_n) = 1] - Pr[G^0_{\mathcal{B}}(m_1, ..., m_n) = 1]$$

The first probability corresponds to the advantage of $\mathcal{A}$ against $DCDH_n$. The second one is the probability that $\mathcal{A}$ outputs g^r where r is randomly sampled from $[1, q]$ and $\mathcal{A}$ has no other information related to r. As q is large, this probability is negligible. Thus the advantage of $\mathcal{A}$ against $DCDH_n$ is negligible. ∎

Summing up the Results Let n be an integer. Henceforth, 3DH is used instead of $3DH_n$. Figure 7.1 summarizes the relations between the DH-assumptions:

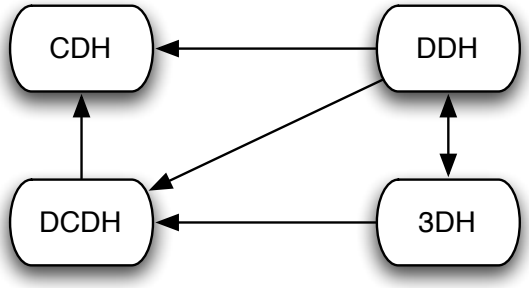


Figure 7.1: Implications between the DH-assumptions

A natural extension would be to allow multi-sets as oracle arguments. However, as noted in [BDZ03], the equivalence between DDH and such extension is a difficult yet unsolved problem.

7.3 Introducing Symbolic Equivalence

In this section, we adapt definitions from chapter 2 We consider protocols in the passive case, hence we extend our notion of message and define symbolic equivalence between messages. This equivalence relation is an extension of the one introduced in [AR00] to the case of modular exponentiation.

Besides, atomic data and concatenated messages, the security protocols we consider may use symmetric encryption and modular exponentiation. Modular exponentiations can be used as keys for symmetric encryption. For this purpose we use a hash function in the computational setting which transforms (randomly sampled) group elements into (randomly sampled) symmetric keys. This hash function is not present in the symbolic model as it is only used when performing symmetric encryption with a group element as key.

7.3.1 Messages and Deductions

We still consider that the set of atomic message $\mathcal{AM}$ is composed of the set $\mathcal{AN}$ of agent names, the set $\mathcal{N}$ of nonces, the set $\mathcal{K}$ of keys. However we add to $\mathcal{AM}$ a new sort $\mathcal{EN}$ of *exp nonces*. Exp nonces are the only type of messages that can be exponentiated. We assume that all these sets are disjoints. The set $\mathcal{K}$ only contains symmetric keys from $SymK$.

Let K, N and p be meta-variables over keys, nonces and respectively polynomials using exp-nonces. We still only consider power-free polynomials: x^2 is forbidden. The set **Msg** of messages is defined by the following grammar:

$$\begin{array}{rcl} msg & ::= & \langle msg, msg\rangle \mid \{msg\}_{key} \mid N \mid key \\ key & ::= & K \mid \exp(p) \end{array}$$

As before, message $\langle m_1, m_2\rangle$ represents pairing of messages m_1 and m_2, $\{m\}_k$ represents (symmetric) encryption of message m using key k and $\exp(p)$ represents modular exponentiation of g to the power of p. For any message m, $poly(m)$ designates the set of polynomials that appear in m.

First we define a deduction relation $E \vdash m$ where E is a finite set of messages and m is a message. The intuitive meaning of $E \vdash m$ is that m can be deduced from E. The deductibility relation is an extension of the classical Dolev-Yao inference system [DY83] as introduced in chapter 2. However we only keep the necessary rules for symmetric encryption.

$$\frac{m\in E}{E\vdash m} \qquad \frac{E\vdash\langle m_1,m_2\rangle}{E\vdash m_1} \qquad \frac{E\vdash\langle m_1,m_2\rangle}{E\vdash m_2}$$

$$\frac{E\vdash m_1 \quad E\vdash m_2}{E\vdash\langle m_1,m_2\rangle} \qquad \frac{E\vdash m \quad E\vdash k}{E\vdash\{m\}_k} \qquad \frac{E\vdash\{m\}_k \quad E\vdash k}{E\vdash m}$$

We add two new deduction rules in order to handle modular exponentiation:

$$\frac{E \vdash \exp(p) \quad E \vdash \exp(q)}{E \vdash \exp(\lambda p + q)}\lambda \in \mathbb{Z} \qquad\qquad \frac{}{E \vdash \exp(1)}$$

These two deductions correspond to capacities of the adversary in the computational setting: the adversary can multiply two group elements g^u and g^v in order to get g^{u+v}. Moreover the adversary can exponentiate a group element g^u and obtain $g^{\lambda.u}$. Thus in the symbolic setting, we allow the first deduction. The second deduction is necessary as we supposed the group generator to be public, hence it is possible for any adversary to output it. We do not consider any explicit equational theory (other than polynomial equality) as in [CKRT03b, MS03]. Indeed, as we are mainly interested in the passive setting, we do not allow exponentiations of exponentiations or products of exponentiations. Hence, we only have to consider equalities between polynomials in normal form (i.e. linear combinations of monomials).

After adding the two new deductions, the deductibility relation is still decidable.

Proposition 7.6 *Let m be a message and E be a finite set of messages. Then deductibility of m from E is decidable.*

Proof:

Let m be a message and E be a finite set of messages. Let $\mathcal{K}$ and $\mathcal{P}$ be respectively the set of keys and the set of polynomials that appear in E and m.

The subset $\mathcal{K}_d$ of $\mathcal{K}$ of deducible keys and the subset $\mathcal{P}_d$ of $\mathcal{P}$ of deducible polynomials [1] can be inductively defined by:

1. Initially, $\mathcal{K}_d$ and $\mathcal{P}_d$ are empty.
2. Then at each step, any key that is reachable using keys from $\mathcal{K}_d$ and exponentiations from $\mathcal{P}_d$ is added to $\mathcal{K}_d$. The same thing is done for reachable polynomials.
3. At the end of each step, if p is a polynomial from $\mathcal{P}$ which is a linear combination of polynomials from $\mathcal{P}_d$, then p is also added to $\mathcal{P}_d$.

Sets $\mathcal{K}$ and $\mathcal{P}$ are finite so this algorithm terminates. Moreover after applying this algorithm, all the deducible keys and polynomials are respectively in $\mathcal{K}_d$ and $\mathcal{P}_d$, let k be a deducible key, then consider the minimal size derivation proving that k is deducible, an iterative argument on this derivation proves that k is in $\mathcal{K}_d$.

In order to test that m is deducible from E, we build the set S of atoms, exponentiations and encryptions (using non-deducible keys or exponentiations) that are reachable from m using $\mathcal{K}_d$ and $\mathcal{P}_d$. Then deductibility holds if all the elements of S are reachable in E using $\mathcal{K}_d$ and $\mathcal{P}_d$. ■

7.3.2 Symbolic Equivalence

Patterns are used to characterize the information that can be extracted from a message. These patterns are close to those introduced in [AR00, MP05] but are extended in order to handle modular exponentiation. Let m be a message then its pattern is inductively defined by:

$$
\begin{aligned}
pattern(\langle m_1, m_2\rangle) &= \langle pattern(m_1), pattern(m_2)\rangle \\
pattern(\{m'\}_{\exp(p)}) &= \{pattern(m')\}_{\exp(p)} && \text{if } m \vdash \exp(p) \\
pattern(\{m'\}_{\exp(p)}) &= \{\square\}_{\exp(p)} && \text{if } m \nvdash \exp(p) \\
pattern(\{m'\}_K) &= \{pattern(m')\}_K && \text{if } m \vdash K \\
pattern(\{m'\}_K) &= \{\square\}_K && \text{if } m \nvdash K \\
pattern(N) &= N \\
pattern(K) &= K \\
pattern(\exp(p)) &= \exp(p)
\end{aligned}
$$

Symbol $\square$ represents a cipher-text that the adversary cannot decrypt. We say that two messages are *equivalent* if they have the same pattern.

$$m \equiv n \text{ if and only if } pattern(m) = pattern(n)$$

We now introduce equivalence up to renaming by allowing renaming of nonces, keys and polynomials. Renaming of keys is necessary: in the computational setting K and K' cannot be distinguished but (K, K) and (K, K') can be. Thus we want K and K' to be equivalent but not (K, K) and (K, K'). Such renamings were already used in [AR00] and in [MP05]. Renaming of nonces works in the same way. However renaming of polynomials is more subtle: let us consider message $(\exp(x), \exp(y), \exp(x+y))$, if this message is simply transformed into $(\exp(x), \exp(y), \exp(z))$,

[1] by abuse of notation, we say that a polynomial p is deducible if $\exp(p)$ is deducible.

then instantiations of these two messages are easy to distinguish in the computational setting. In the first case, the third element is the product of the two first ones whereas in the second case, this is only the case with negligible probability. In order to fix this problem, we only consider *linear relation preserving bijections* or lrp bijections. Such bijections have to preserve linear equations among polynomials which are exponentiated. Let us formalize this. Let σ be a bijection from $poly(n)$ to $poly(m)$. Then σ is said to be lrp if the exact same equations are satisfied after applying it:

$$\forall p_1 ... p_n \in poly(n),\ \forall a_1, ..., a_n, b \in \mathbb{Z},\ \sum_{i=1}^{n} a_i p_i = b \Leftrightarrow \sum_{i=1}^{n} a_i p_i \sigma = b$$

It is important to note that coefficients range over $\mathbb{Z}$ and not $\mathbb{Z}_q$ anymore. This condition guarantees that linearly dependent bound polynomials are adequately renamed. Then two messages are *equivalent up to renaming* if they are equivalent up to some renaming of keys, nonces and polynomial.

$m \cong n$ if and only if
$\exists \sigma_1$ a permutation of **Keys**
$\exists \sigma_2$ a permutation of **Nonces**
$\exists \sigma_3$ a linear relation preserving bijection of polynomials
such that $m \equiv n\sigma_1\sigma_2\sigma_3$

However, linear relation preserving as defined previously is hard to check, thus we give another equivalent way of defining lrp bijections. Let $p_1, ..., p_j$ be a basis of $poly(n)$. Bijection σ is lrp if the two following conditions are satisfied:

1. Polynomials $p_1\sigma, ..., p_j\sigma$ have to be linearly independent.
2. For any polynomial p of $poly(n)$, let λ_1 to λ_j be the integers such that $p = \sum_{i=1}^{j} \lambda_i p_i$. Then σ has to verify the following equality:

$$(\sum_{i=1}^{j} \lambda_i p_i)\sigma = \sum_{i=1}^{j} \lambda_i (p_i \sigma)$$

This new definition allows one to define bijections only on a basis of $poly(n)$. Definitions for other polynomials of $poly(n)$ are a consequence of the values chosen on this basis.

Using this new definition, an interesting result is the decidability of equivalence up to renaming.

Proposition 7.7 *Let m and n be two messages. Equivalence up to renaming of m and n is decidable.*

7.3.3 Examples

In this section, we give some examples that illustrate the choices we made when defining the equivalence. These choices are motivated by the possibilities of adversaries in the computational setting. Unlike [AR00], our symbolic model does not include symbolic constants like 0 or 1. However it is possible to encode these constants using for example exponentiations. Then 1 denotes g^0 (note that this is unit element for multiplication) and 0 denotes g^1 (but this is not the unit element for addition). Symbols 0 and 1 can only be used as constants and cannot be used to represent arithmetic properties.

1. $\{0\}_K \cong \{1\}_K$. This example shows that symmetric encryption perfectly hides its plain-text.

2. $(\{0\}_K, \{0\}_K) \cong (\{0\}_K, \{1\}_K)$. Symmetric encryption also hides equalities among the underlying plain-texts. To achieve this, encryption has to be probabilistic. As modular exponentiation is deterministic, we cannot ask modular exponentiation to hide such relations.

3. $\big(\exp(X_1), \exp(X_2), \exp(X_1 + X_2)\big) \cong \big(\exp(X_1 - X_2), \exp(X_2), \exp(X_1)\big)$. Linear relations between exponents are not hidden by modular exponentiation. In this case, the third element is linked to the two previous ones. Hence the same relation holds on both side: the exponents of the third term is the sum of the exponents of the two first terms.

4. $\big(\exp(X_1), \exp(X_2), \exp(X_1 + X_2)\big) \ncong \big(\exp(X_1 + X_2), \exp(X_2), \exp(X_1)\big)$. If linear equations are not preserved by the renaming, then the two messages are not equivalent up to renaming. In this situation, the third exponents is the sum of the previous ones in the left message, whereas it is the difference in the right message.

5. $\big(\exp(X_1), \exp(X_2), \{0\}_{\exp(X_1X_2)}\big) \cong \big(\exp(X_1), \exp(X_2), \{1\}_{\exp(X_1X_2)}\big)$ This example illustrates a passive adversary which observes a Diffie-Hellman key exchange protocol. Two exponentiations are exchanged that allow the two participants to build a shared secret key. In this example, this key is used to hide a plain-text which can be either 0 or 1. This example can be generalized to represent a three-party Diffie-Hellman protocol:

$$\begin{aligned} & \big(\exp(X_1), \exp(X_2), \exp(X_3), \exp(X_1X_2), \exp(X_1X_3), \exp(X_2X_3), \{0\}_{\exp(X_1X_2X_3)}\big) \\ \cong\ & \big(\exp(X_1), \exp(X_2), \exp(X_3), \exp(X_1X_2), \exp(X_1X_3), \exp(X_2X_3), \{1\}_{\exp(X_1X_2X_3)}\big) \end{aligned}$$

7.4 Soundness of Formal Encryption

In this section, we formalize the mapping between symbolic messages and distributions of bit-strings. The main result is that equivalence up to renaming is sound: if two messages are equivalent up to renaming in the symbolic setting, their concretizations are computationally indistinguishable. To achieve this, the cryptographic primitives used to concretize the messages have to be secure. For modular exponentiation, the security notion is DDH whereas for the symmetric encryption scheme, we use the classical IND-CPA (indistinguishability against chosen plain-text attacks) notion [GM84].

7.4.1 Computational Semantics of Messages

The computational semantics depends on a symmetric encryption scheme $\mathcal{SE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ and on a family of cyclic groups $\mathbb{G}$. We suppose that there exists a hash algorithm *hash* such that the distribution of keys generated by $\mathcal{KG}$ is indistinguishable from the result of hash applied to a random element of $\mathbb{G}$. We associate with each symbolic message m a distribution of bit-string $[\![m]\!]_{\mathcal{SE},\mathbb{G}}$ that depends on η. This distribution is defined by the following random algorithm:

1. For each key K (resp. each nonce N, each exp-nonce X) from m, a value $\widehat{K}$ (resp. $\widehat{N}$, $\widehat{X}$) is randomly sampled using $\mathcal{KG}$ (resp. is randomly sampled in $[0,1]^\eta$, is randomly sampled in $\mathbb{Z}_q$).

2. The computational value of a composed message m denoted by $\widehat{m}$ is recursively computed:

$$\begin{aligned} \widehat{(m,n)} &= \widehat{m}.\widehat{n} && \text{concatenation} \\ \widehat{\{m\}_K} &= \mathcal{E}(\widehat{m}, \widehat{K}) && \text{encryption with key} \\ \widehat{\{m\}_{\exp(p)}} &= \mathcal{E}\big(\widehat{m}, \mathrm{hash}(\widehat{\exp(p)})\big) && \text{encryption with exponentiation} \\ \widehat{\exp(\sum_{i=1}^{n} \lambda_i X_1^i ... X_{j_i}^i)} &= g^{\sum_{i=1}^n \lambda_i \widehat{X_1^i} ... \widehat{X_{j_i}^i}} && \text{modular exponentiation} \end{aligned}$$

3. The resulting bit-string is $\widehat{m}$.

This algorithm is close to the *concr* algorithm used in previous chapters. We did not want to reuse this *concr* algorithm as here we first have to randomly sample values for the different atoms whereas *concr* is given a computational substitution.

7.4.2 Soundness Result

In this section, we present the main soundness result of this chapter. This result is an extension of the main theorem presented in [AR00] to the case of messages that involve modular exponentiation. As in previous work, this result needs an acyclicity hypothesis: messages considered here must not contain any encryption cycle. If one wants to consider encryption cycles, the IND-CPA hypothesis is too weak. However, key dependent security [BRS01] should be a strong enough hypothesis to ensure soundness in this case.

First let us properly define our acyclicity requirement: a message m is said to be *acyclic* if the following conditions are satisfied:

1. Let $\mathcal{P}(m)$ be the set of polynomials p such that $\exp(p)$ appears as a key in m. Then polynomial p cannot be involved in any linear combination in m, i.e. there does not exist p_1,...p_j polynomials from m (used as keys or as plain-texts) different from p such that $p = \sum_{i=1}^{j} \lambda_i.p_i$

2. There exists an ordering $\prec$ between keys and polynomials from $\mathcal{P}(m)$: if u appears encrypted using v or $\exp(v)$ then $u \prec v$. This ordering must not have any cycles.

This requirement is quite rough but generally holds on key exchange protocols. Its main purpose is to forbid key cycles of the form $\{\exp(x), \exp(x+y)\}_{\exp(y)}$ which can lead to real attacks.

We also have to define indistinguishable distributions. Intuitively, two distributions D_1 and D_2 are computationally indistinguishable if for any adversary $\mathcal{A}$, the probability for $\mathcal{A}$ to make the difference between a randomly sampled element of D_1 or of D_2 is negligible.

Definition 7.1 *Let D_1 and D_2 be two distributions (that depend on η). The advantage of an adversary $\mathcal{A}$ in distinguishing D_1 and D_2 is defined by:*

$$\mathbf{Adv}_{\mathcal{A}}^{D_1,D_2} = Pr[x := D_1(\eta)\ ;\ \mathcal{A}(x) = 1] - Pr[x := D_2(\eta)\ ;\ \mathcal{A}(x) = 1]$$

Distributions D_1 and D_2 are computationally indistinguishable*, $D_1 \approx D_2$, if the advantage for any adversary $\mathcal{A}$ in distinguishing D_1 and D_2 is negligible.*

Then our main result states that distributions related to equivalent messages are computationally indistinguishable.

Proposition 7.8 *Let m and n be two acyclic messages, such that $m \cong n$. Let $\mathcal{SE}$ be a symmetric encryption scheme that is IND-CPA secure and $\mathbb{G}$ be a group such that the DDH assumption holds, then $[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![n]\!]_{\mathcal{SE},\mathbb{G}}$.*

Proof: This proof is done by considering two intermediate lemmas. The first one handles renamings, the second one, using DDH and IND-CPA, relates the computational distribution of m to the computational distribution of its pattern. Hence we introduce a computational evaluation for $\Box$ which is: $\widehat{\Box} = 0$.

Lemma 7.5 *Let σ_1 be a permutation of* **Keys***, σ_2 be a permutation of* **Nonces** *and σ_3 be a lrp bijection of polynomials. If the DDH assumption holds on $\mathbb{G}$ then for any message m we have that:*

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![m\sigma_1\sigma_2\sigma_3]\!]_{\mathcal{SE},\mathbb{G}}$$

Proof: The cases of σ_1 and σ_2 are trivial. The difficulty is to prove that for any message m,

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![m\sigma_3]\!]_{\mathcal{SE},\mathbb{G}}$$

Let x_1 to x_α be the exp-nonces used by m and let Γ be the set of monomials that use these variables. Let p_1 to p_j be a basis of $poly(m)$. Other polynomials from $poly(m)$ are written $p = \sum_{i=1}^{j} \lambda_i.p_i$. Then we consider renaming σ'_3 defined by $p_i\sigma'_3 = y_i$ for $i \in [1, j]$ where each y_i is a fresh distinct variable. As DDH holds, 3DH also holds. Using 3DH, it is easy to prove that

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![m\sigma'_3]\!]_{\mathcal{SE},\mathbb{G}}$$

For this purpose, let us consider an adversary $\mathcal{A}$ that tries to distinguish these two distributions. Then we build an adversary $\mathcal{A}'$ against 3DH that has the exact same advantage:

Adversary $\mathcal{A}'/\mathcal{O}_R, \mathcal{O}_{RR}$:
  for i **from** 1 **to** j
    $y_i := \mathcal{O}_{RR}(p_i)$
  randomly generate keys and nonces from m
  recursively compute $\widehat{m}$
  in order to compute $\exp(\widehat{\sum_{i=1}^{j} \lambda_i.p_i})$ use $\prod_{i=1}^{j} y_i^{\lambda_i}$

When the challenge bit of $\mathcal{A}'$ equals 1, $\mathcal{A}'$ is faced to $[\![m]\!]_{\mathcal{SE},\mathbb{G}}$. Otherwise it is faced to $[\![m\sigma'_3]\!]_{\mathcal{SE},\mathbb{G}}$. Hence, the advantage of $\mathcal{A}'$ is given by:

$$\begin{aligned}
\mathbf{Adv}^{G(\Gamma)}_{\mathcal{A}'} &= Pr[G^1_{\mathcal{A}}(\Gamma) = 1] - Pr[G^0_{\mathcal{A}}(\Gamma) = 1] \\
&= Pr[\mathcal{A}(bs) = 1 | bs := [\![m]\!]_{\mathcal{SE},\mathbb{G}}] - Pr[\mathcal{A}(bs) = 1 | bs := [\![m\sigma'_3]\!]_{\mathcal{SE},\mathbb{G}}] \\
&= \mathbf{Adv}^{[\![m]\!]_{\mathcal{SE},\mathbb{G}}, [\![m\sigma'_3]\!]_{\mathcal{SE},\mathbb{G}}}_{\mathcal{A}}
\end{aligned}$$

As 3DH holds, the advantage of $\mathcal{A}'$ is negligible and so the advantage of $\mathcal{A}$ is also negligible. Therefore, these two distributions are computationally indistinguishable.

Moreover, let us consider renaming σ''_3 defined by $(p_i\sigma_3)\sigma''_3 = x_i$ for $i \in [1, j]$. Using 3DH we also have that:

$$[\![m\sigma_3]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![m\sigma_3\sigma''_3]\!]_{\mathcal{SE},\mathbb{G}}$$

Messages $m\sigma'_3$ and $m\sigma_3\sigma''_3$ are identical therefore we finally get that:

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![m\sigma_3]\!]_{\mathcal{SE},\mathbb{G}}$$

■

Lemma 7.6 *Let m be an acyclic message. If the DDH assumption holds on $\mathbb{G}$ and $\mathcal{SE}$ is secure against IND-CPA then:*

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![pattern(m)]\!]_{\mathcal{SE},\mathbb{G}}$$

Proof: We only give a sketch of this proof as this is mainly an adaptation of the hybrid argument described in [AR00].

This proof is done using a recursion alon $\prec$. At each step the number of non-deducible keys (or exponentiations) that are used to encrypt messages different from $\Box$ decreases. Let $\mathcal{K}$ be the set of such keys and exponentiations.

1. If $\mathcal{K}$ is empty, then m is equal to its pattern and the property is trivial.
2. Else if the maximum of $\mathcal{K}$ for $\prec$ is a key k. Key k is not deducible hence acyclicity of m implies that k only occurs at key positions in m. Let m' be message m where encryptions using k are replaced by $\{\Box\}_k$. Then using IND-CPA (see [AR00] for details), messages m and m' are computationally indistinguishable.

3. Else if the maximum of $\mathcal{K}$ for $\prec$ is an exponentiation $\exp(p)$. Acyclicity implies that p is linearly independent from other polynomials of m and that p only appears at key positions in m. Let m' be message m where encryptions using $\exp(p)$ are replaced by $\{\Box\}_{\exp(p)}$. Let m_p be message m where $\exp(p)$ is replaced by some fresh key k_p and m'_p be m_p where encryptions using k_p are replaced by $\{\Box\}_{k_p}$. Then using 3DH (which holds as DDH holds), we get that m and m_p are computationally indistinguishable. Using ind-cpa, we get that m_p and m'_p are computationally indistinguishable and finally by using once more 3DH, we get that m'_p and m_p are indistinguishable. Finally, we get that:

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![m']\!]_{\mathcal{SE},\mathbb{G}}$$

■

Let m and n be two acyclic messages such that $m \cong n$. By definition we know that there exist σ_1 be a permutation of **Keys**, σ_2 be a permutation of **Nonces** and σ_3 be a lrp bijection of polynomials such that $m \equiv n\sigma_1\sigma_2\sigma_3$. Using lemma 7.6, we get that:

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![pattern(m)]\!]_{\mathcal{SE},\mathbb{G}}$$

Moreover, as n is acyclic, $n\sigma_1\sigma_2\sigma_3$ is also acyclic, hence we obtain:

$$[\![n\sigma_1\sigma_2\sigma_3]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![pattern(n\sigma_1\sigma_2\sigma_3)]\!]_{\mathcal{SE},\mathbb{G}}$$

By definition of $m \equiv n\sigma_1\sigma_2\sigma_3$, the two patterns are equal and we get:

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![n\sigma_1\sigma_2\sigma_3]\!]_{\mathcal{SE},\mathbb{G}}$$

Finally we apply lemma 7.5 to obtain the expected result:

$$[\![m]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![n]\!]_{\mathcal{SE},\mathbb{G}}$$

■

This result states soundness of formal equivalence in the computational world. However, the converse (i.e. completeness) is in general false. There are two main problems that prevent completeness. First, the symmetric encryption scheme may allow decryption with the wrong key and output a random bit-string in this case. Then messages $(\{N\}_K, K)$ and $(\{N\}_K, K')$ can be computationally indistinguishable. This can be solved by adding a confusion freeness hypothesis for the symmetric encryption scheme [MW04b, AJ01]. The second problem is that the symmetric encryption scheme can satisfy key concealing (this is ensured by type 0 security in [AR00]). Then messages $(\{0\}_K, \{0\}_{K'})$ and $(\{0\}_K, \{0\}_K)$ are computationally indistinguishable but are not equivalent even with renaming. To solve this, one can either ask the encryption scheme to be key revealing or modify the pattern definition in order to hide the key name (but the encryption scheme has to be key concealing in order to prove soundness).

7.4.3 Application to the Burmester-Desmedt Protocol

As an example, we show how to apply our results to the group Key Exchange protocol of Burmester-Desmedt presented at Eurocrypt'94 [BD94]. The aim of this protocol is to establish a secret key shared among the members of the group. It is scalable as it requires only two rounds and a constant number of modular exponentiation per user. No proof of this protocol has been published until recently [KY03, BD05], where a proof of key secrecy in the presence of a passive adversary is presented. The protocol is not secure in the presence of an active adversary as it does not provide any form of authentication.

Protocol description Consider a network in which members of a group can broadcast messages to all others. Let η be a security parameter and let $A_1, A_2, \cdots, A_n$, for $n \in \mathbb{N}$, be members of a group. In this section, all subscripts are taken modulo n, for example A_{n+1} designates A_1 and A_0 designates A_n.

1. **Initialization:** The parameters of the protocol are: a security parameter η, a finite cyclic group $\mathbb{G} = \langle g \rangle$ of prime-order q, a large number. The parameters $\mathbb{G}$, g and q are published.
2. **Round 1:** Each participant A_i samples a random $x_i \in \mathbb{Z}_q$, and broadcasts $Z_i := g^{x_i}$.
3. **Round 2:** Each participant A_i broadcasts $X_i = (Z_{i+1}/Z_{i-1})^{x_i}$. Hence, Z_i is equal to $g^{x_i x_{i+1} - x_{i-1} x_i}$.
4. **Key computation:** Each party computes the key [2]:

$$K_i := Z_{i-1}^{n x_i} X_i^{n-1} X_{i+1}^{n-2} \dots X_{i+n-2}$$

Let us compute a simplified value for the key and verify that this value does not depend on i.

$$\begin{aligned}
X_{i+n-2} &= g^{x_{i-2}x_{i-1} - x_{i-3}x_{i-2}} \\
X_{i+n-3}^2 X_{i+n-2} &= g^{2(x_{i-3}x_{i-2} - x_{i-4}x_{i-3})} g^{x_{i-2}x_{i-1} - x_{i-3}x_{i-2}} \\
&= g^{x_{i-2}x_{i-1} + x_{i-3}x_{i-2} - 2x_{i-4}x_{i-3}} \\
\dots &= \dots \\
X_i^{n-1} X_{i+1}^{n-2} \dots X_{i+n-2} &= g^{x_{i-1}x_{i-2} + x_{i-3}x_{i-4} + \dots + x_{i-n+1}x_{i-n} - (n-1)x_{i-n}x_{i-n-1}} \\
K_i &= g^{n x_i x_{i-1}} X_i^{n-1} X_{i+1}^{n-2} \\
&= g^{\sum_{i=1}^n x_i x_{i+1}}
\end{aligned}$$

We call this protocol BDGKE, the Burmester-Desmedt Group Key Exchange protocol.

Security properties We prove that BDGKE enjoys the key secrecy property in presence of a passive adversary that may read and record the messages sent during protocol execution. This is the main result of [BD05], where a complexity-theoretic proof is presented. Key secrecy was introduced by Bellare and Rogaway [BR93] and later extended for multi-party applications by Bresson et al. [BCPQ01]. It states that the key computed using the protocol is indistinguishable from a randomly generated key. This implies in particular that either this key is not used to perform any encryption or the symmetric encryption scheme has to be key concealing as defined in section 6.3.2.

Protocol verification A transcript of the execution of the protocol is given by the expression M:

$$M = \left\langle \exp(x_1), \cdots, \exp(x_n), \exp(x_2x_1 - x_nx_1), \cdots, \exp(x_1x_n - x_{n-1}x_n), \exp(\sum_{i=1}^n x_i x_{i+1}) \right\rangle$$

Let N be the expression obtained from M by replacing the key $\exp(\sum_{i=1}^n x_i x_{i+1})$ by $\exp(r)$.

$$N = \left\langle \exp(x_1), \cdots, \exp(x_n), \exp(x_2x_1 - x_nx_1), \cdots, \exp(x_1x_n - x_{n-1}x_n), \exp(r) \right\rangle$$

Then, by proposition 7.8, if $M \cong N$ then $[\![M]\!]_{\mathcal{SE},\mathbb{G}} \approx [\![N]\!]_{\mathcal{SE},\mathbb{G}}$, provided DDH is satisfied. Therefore, to prove that BDGKE satisfies the key secrecy property, under the hypothesis that DDH holds, it is sufficient to show $M \cong N$.

[2] Strictly speaking, K_i is not the key but the shared secret from which a shared key is derived by, for instance, applying a hash function.

Now, since M and N do not contain any encryption as sub-expression, we have $pattern(M) = M$ and $pattern(N) = N$. Let $\mathcal{P} = \{x_i, x_{i+1}x_i - x_{i-1}x_i \mid i = 1, \cdots, n\}$. Then, $poly(M) = \mathcal{P} \cup \{\sum_{i=1}^{n} x_i x_{i+1}\}$ and $poly(N) = \mathcal{P} \cup \{r\}$.

Consider now the substitution $\sigma : poly(N) \to poly(M)$ that coincides with the identity except that $\sigma(r) = \sum_{i=1}^{n} x_i x_{i+1}$. Obviously σ is bijective. Hence, all that remains to be verified for concluding $M \cong N$, is that σ is a lrp substitution. The only non-trivial equality among polynomials from $poly(M)$ or $poly(N)$ is:

$$x_1 x_n - x_{n-1} x_n = \sum_{i=1}^{n-1} -(x_{i+1}x_i - x_{i-1}x_i)$$

Since, σ is defined as the identity on the polynomials implied in the previous equality, we have

$$(x_1 x_n - x_{n-1} x_n)\sigma = \sum_{i=1}^{n-1} -(x_{i+1}x_i - x_{i-1}x_i)\sigma$$

As σ preserves this equation and all the other equations are either derived from this one or trivial, σ preserves linear relations. Thus we get that σ is a lrp and so $M \cong N$. Finally, using proposition 7.8, we get that the two distributions are indistinguishable. This proves that for any adversary $\mathcal{A}$, if $\mathcal{A}$ is given g^{x_i} and $g^{x_i x_{i+1} - x_{i-1}x_i}$ for any i, then $\mathcal{A}$ cannot make the difference between the shared key and a random group element with non negligible probability.

Although this example illustrates a possible use of exponentiation in our symbolic equivalence notion, it does not use symmetric encryption. Hence we extend this example in order to use this cryptographic primitive. We consider a Burmester-Desmedt key agreement and after that a participant sends a secret nonce S using the shared key. We want to verify that this secret nonce is indistinguishable from another randomly sampled secret nonce. The adversary can observe the execution of the protocol. After that he is given either the nonce value or another randomly sampled values and he has to decide which is the case. Thus we consider two possible transcripts for the protocol. M represents the correct execution:

$$M = \langle \exp(x_1), \cdots, \exp(x_n), \exp(x_2x_1 - x_nx_1), \cdots, \exp(x_1x_n - x_{n-1}x_n), \{S\}_{\exp(\sum_{i=1}^{n} x_i x_{i+1})}, S \rangle$$

And N is the execution where the adversary is finally a random nonce S' which is not the secret nonce used in the encryption.

$$N = \langle \exp(x_1), \cdots, \exp(x_n), \exp(x_2x_1 - x_nx_1), \cdots, \exp(x_1x_n - x_{n-1}x_n), \{S\}_{\exp(\sum_{i=1}^{n} x_i x_{i+1})}, S' \rangle$$

We first have to verify that our acyclicity requirement is met. This is the case as $\exp(\sum_{i=1}^{n} x_i x_{i+1})$ is linearly independent from any other polynomials from M and N. There is no necessity for renaming polynomials. M and N have the same patterns up to renaming of S' to S. Therefore $M \cong N$ and applying proposition 7.8, we can conclude that the two distributions are indistinguishable and that the SecNonce property is verified for S. This result can also be seen as a consequence of the first result: after seeing the key agreement part, the key cannot be distinguished from a random key. Hence encryptions using this key are safe as the encryption scheme is IND-CPA.

Chapter 8

Unbounded Number of Sessions

Contents

In section 6.2, we linked the symbolic and computational aspects of security protocols. However, to achieve this, we had to make several hypotheses. The most restrictive one is that we consider protocols that are only made of a single instance of a single role. As noted in section 2.3.3, this restriction is fair if we consider a bounded number of sessions of a protocol. In this case, it is possible to consider a single role without loss of generality. This restriction is real when considering an unbounded number of sessions. Then it is impossible to make such a simplification without losing generality.

In this chapter, we investigate the case of an unbounded number of sessions. Note that a truly unbounded number of sessions does not make any sense. Adversaries have an execution time that is polynomially bounded. Thus, the number of possible sessions is also polynomially bounded. The main difficulty with respect to the proof presented in section 6.2 is that reduction properties such that property 5.4 only work for a bounded number of challenges. Let us consider more closely this property.

Proposition 5.4 Let N be an integer. If an asymmetric encryption scheme $\mathcal{AE}$ is IND-CCA, then $\mathcal{AE}$ is N-PAT-IND-CCA.

The integer N is fixed and does not depend on the security parameter η. We are interested in extending this proposition to the case where a polynomial number of keys is available to the adversary.

Proposition Poly Let P be a polynomial. If an asymmetric encryption scheme $\mathcal{AE}$ is IND-CCA, then $\mathcal{AE}$ is P-PAT-IND-CCA.

In criterion P-PAT-IND-CCA, the adversary has access to $P(\eta)$ different challenge keys. The ultimate goal of this chapter is to define a truly dynamic and unbounded criterion ∞-PAT-IND-CCA. In this criterion, the adversary can ask for creation of new challenge keys (this is done by accessing an oracle). Then he can access oracles related to the newly generated key and to previously generated ones.

Proposition ∞ If an asymmetric encryption scheme $\mathcal{AE}$ is IND-CCA, then $\mathcal{AE}$ is ∞-PAT-IND-CCA.

Moreover this kind of proposition can usually be proved using an hybrid argument. This is the case in [BBM00] where N-IND-CCA is proven equivalent to IND-CCA even if N is polynomial in the security parameter. The main challenge when considering polynomial criteria is indeed *adaptive corruption*, if some keys can be revoked then the equivalence result is far more complicated to obtain. The adaptive corruption problem raises issues close to the selective decommitment for which no solution is known today [CFGN96, DNRS99].

It is possible to extend the computational semantics of protocols using the same idea. Using an oracle, the adversary can ask for the creation of a new session. He can specify the agents name and parameters in this session. The adversary also has access to oracles in order to send and receive messages. These oracles take as input the session and the agent that the adversary is referring to.

The first step is to introduce criteria where a polynomial number of challenges are generated. The advantage of this new criterion formalism is that we separate challenges that are generated a polynomial number of times (i.e. keys in IND-CCA) from challenges that are generated only once (i.e. bit b in IND-CCA).

This chapter is organised as follows. In section 1, we introduce polynomial criteria and give an adapted simplified partition theorem. This theorem is exemplified on an extension of IND-CPA that deals with adaptive corruptions. In section 2 we formulate the main partition theorem in the polynomial case, we exemplify this on a variant of SYM-CPA. Fully dynamic criteria are defined in section 3 where adapted partition theorems are also given. Finally section 4 shows how these results can be used to prove computational soundness of formal methods.

8.1 Polynomial Number of Challenges

In this section, we first introduce *iterated criteria*. Then we adapt our partition theorem to the case of such criteria. However, we do not consider the extended version of the theorem. The case of the "complete" theorem is not really more complicated and is detailed in section 8.2. To illustrate the interest of our theorem, we exemplify its application to an extension of IND-CPA where a polynomial number of challenge keys are considered in an adaptive corruption setting. Using the following theorem, polynomial IND-CPA can be proven equivalent to IND-CPA.

8.1.1 Iterated Criteria

Let P be a polynomial. We do not want to generate a negative or null number of challenges. Hence in the rest of this chapter, we only consider positive polynomials, i.e. polynomials Q such that for any positive integer x, $Q(x)$ is a positive non-null integer. This is not a true restriction as for any polynomial P, there exists a *positive* polynomial Q such that for any x, $P(x) \leq Q(x)$.

An iterated criterion γ is a tuple $(\Theta_0; \Theta; F_0; F; V_0)$ where:

- Θ_0 is the challenge generator that is only called once.
- Θ is the *single challenge generator*. During the experiment related to this criterion, this generator is called $P(\eta)$ times (where η is the security parameter). Hence $P(\eta)$ challenges are generated. They are usually denoted by θ_1 to $\theta_{P(\eta)}$).
- F_0 is the oracle related to Θ_0, it does not use any other challenges.
- F is the oracle related to challenges generated by Θ. It is composed of an oracle for each generated challenge θ_i for i between 1 and $P(\eta)$.

- Finally verifier V_0 checks that the output of the adversary is correct. To simplify things, V_0 only depends on θ_0 (i.e. the challenge generated by Θ_0) in this section.

Oracle F_0 only uses challenge θ_0. The part of oracle F related to the i^{th} generated challenge uses challenge θ_i and can also use challenges θ_j for j lower than i. In particular, oracle F can use the information stored in θ_0. Hence oracle F can have a variable number of parameters. The oracle related to θ_1 is $F(s, \theta_0, \theta_1)$ whereas the oracle related to θ_2 is $F(s, \theta_0, \theta_1, \theta_2)$. In the following, we use the notation F_i to refer to oracle F used in the case of θ_i.

Let $\mathcal{A}$ be an adversary against γ. Then the experiment involving $\mathcal{A}$ depends on the positive polynomial P giving the number of challenges. Notation γ_P is used to denote criterion γ where the challenge generator Θ is iterated $P(\eta)$ times. The experiment proceeds as follows: challenge θ_0 is generated using Θ_0. Challenges θ_1 to $\theta_{P(\eta)}$ are generated using Θ. Adversary $\mathcal{A}$ is executed and has access to oracle F_0 and to oracles F_i where i ranges between 1 and $P(\eta)$. Finally, the adversary outputs his result d and the verifier is called to check that the output is correct. The game can be detailed by the following procedure:

Game $\mathbf{G}_{\mathcal{A}}^{\gamma_P}(\eta)$:
θ_0:=$\Theta_0(\eta)$
for i **from** 1 **to** $P(\eta)$
θ_i:=$\Theta(\eta)$
d:=$\mathcal{A}(\eta)/\lambda s.F_0(s, \theta_0)$,
$\lambda s.F(s, \theta_0, \theta_1)$
...
$\lambda s.F(s, \theta_0, \theta_1, \ldots, \theta_{P(\eta)})$
return $V_0(d, \theta_0)$

Note that as before, generators Θ_0 and Θ may generate multiple challenges. Oracle F_0 and each oracle F may represent multiple oracles. The definition of advantage is unchanged compared to classical criteria.

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_P}(\eta) = 2\big(Pr[\mathbf{G}_{\mathcal{A}}^{\gamma_P}(\eta) = true] - PrRand^{\gamma_P}(\eta)\big)$$

As verifier V_0 only depends on θ_0, $PrRand$ does not change when P is modified. Moreover, $PrRand$ is defined by using an experiment without any oracle, hence we use $PrRand^{\gamma_0}$ to compute $PrRand$. Thus, the advantage is given by:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_P}(\eta) = 2\big(Pr[\mathbf{G}_{\mathcal{A}}^{\gamma_P}(\eta) = true] - PrRand^{\gamma_0}(\eta)\big)$$

Let us illustrate this extension of criteria. For this purpose, we consider an extension of N-PAT-IND-CPA where N can be a polynomial.

Example 8.1 (P-PAT-IND-CPA) *Let $\mathcal{AE}$ be an asymmetric encryption scheme $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$. Criterion P-PAT-IND-CPA is defined by $(\Theta_0; \Theta; F_0; F; V_0)$ where:*

- *Θ_0 generates a random bit b.*
- *Θ generates a key pair pk, sk using $\mathcal{KG}$.*
- *Oracle F_0 is empty, it always returns the empty bit-string.*
- *Oracle F is composed of two oracles. The first one outputs the public key from Θ_i. The second one takes as argument a pair of patterns pat_0, pat_1. When related to challenge Θ_i, oracle F selects pattern pat_b (where bit b comes from θ_0), applies concr using θ_j for j between 1 and $i-1$ and encrypts the result using the public key stored in θ_i. Hence,*

$$F^{LR}(\langle pat_0, pat_1\rangle, \theta_0, \ldots, \theta_i) = \mathcal{E}\big(concr(pat_{b\theta_0}, \theta_1, \ldots \theta_{i-1}), pk\theta_i\big)$$

The public key oracle is given by:

$$F^{pk}(s, \theta_0, \ldots, \theta_i) = pk\theta_i$$

- *Verifier V_0 verifies that the adversary correctly output the value of bit b.*

This criterion allows an adversary to try to break IND-CPA by using a polynomial number of challenge keys. The acyclicity requirement related to PAT is automatically met as oracle F_i can only fill patterns using keys in θ_j for j strictly lower than i.

8.1.2 Partition Theorem

The partition theorem introduced in chapter 5 can be adapted to the case of iterated criteria. We only consider the simplified partition theorem here. Its main restriction towards the general case is that the final verifier can only depend on a fixed part of the criterion (denoted by θ_0 in the theorem). In this case, the theorem is iterated $P(\eta)$ times. Thus the advantage of any adversary $\mathcal{A}$ against γ_P can be related to the advantage of an adversary against criterion γ_0 (also denoted criterion δ) and to the advantage of another adversary against an indistinguishability criterion β. Therefore, if one wants to check that the advantage of any adversary against γ_P is negligible, he just has to check that the advantages of any adversary against γ_0 and β are negligible and to apply the partition theorem.

Theorem 8.1 (Simplified Partition Theorem, Polynomial Case) *Let P be a positive polynomial. Let γ be the iterated criterion*

$$\gamma = (\Theta_0, \Theta, F_0, F, V_0)$$

We assume that there exist some functions f, f' and g that can be represented by PRTMs such that for any i greater than or equal to 1 such that the oracle F_i consists in both $\lambda s \cdot f(g(s, [\theta_0, \ldots, \theta_{i-1}]), \theta_i)$ and $\lambda s \cdot f'(s, \theta_i)$ when applied to parameters s and θ_0 to θ_i.

Then, for any adversary $\mathcal{A}$ against criterion γ_P, there exist two adversaries $\mathcal{B}$ and $\mathcal{C}$, such that :

$$\forall \eta,\ \boldsymbol{Adv}_{\mathcal{A}}^{\gamma_P}(\eta) = 2P(\eta)\boldsymbol{Adv}_{\mathcal{B}}^{\beta}(\eta) + \boldsymbol{Adv}_{\mathcal{C}}^{\delta}(\eta)$$

where $\beta = (\Theta, b; f \circ LR^b, f'; v_b)$ is indeed an indistinguishability criterion and $\delta = (\Theta_0; F_0; V_0)$ is our "main" criterion.

Proof: This proof can be done by iterating theorem 5.1. The idea is from $\mathcal{A}$ to obtain $\mathcal{B}$ and $\mathcal{C}$ using the partition theorem, then to decompose $\mathcal{C}$ and get $\mathcal{B}'$ and $\mathcal{C}'$ and so on on $\mathcal{C}'$. The advantage of $\mathcal{A}$ has the form:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_P}(\eta) = 2\sum_{i=1}^{P(\eta)} \mathbf{Adv}_{\mathcal{B}^i}^{\beta}(\eta) + \mathbf{Adv}_{\mathcal{C}^{P(\eta)}}^{\delta}(\eta)$$

We then consider an adversary $\mathcal{D}$ against β that randomly chooses an integer i in $[1, P(\eta)]$ and has the same behavior as $\mathcal{B}^i$. Hence the advantage of $\mathcal{D}$ is the average of the advantage of the $\mathcal{B}^i$'s. Thus by renaming $\mathcal{D}$ as $\mathcal{B}$, we get the expected result.

Let us make precise the technical part of the proof by formally describing machines $\mathcal{B}^i$ and $\mathcal{C}^i$. The form of these machines comes from the proof of theorem 5.1. It is necessary to describe these machines as we want to check that there exists a machine $\mathcal{D}$ whose advantage is the average advantage of the $\mathcal{B}^i$'s. For this reason, given an integer i between 1 and $P(\eta)$, it must be possible to generate the code of the $\mathcal{B}^i$ in polynomial time.

Adversary $\mathcal{C}^i$ plays against criterion δ_i which is equivalent to criterion γ iterated $P(\eta) - i$ times, this criterion is equivalent to criterion γ_{P-i}.

Adversary $\mathcal{C}^i(\eta)/\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_{P(\eta)-i}$:
  for j **in** $[1, i]$:
    θ_0^j:=$\Theta_0(\eta)$
    θ_1^j:=$\Theta(\eta)$
    ...

$\theta^j_{P(\eta)+1-j}$:=$\Theta(\eta)$
s:=$\mathcal{A}(\eta)/\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_{P(\eta)-i}$,
$\lambda s.F_{P(\eta)+1-i}(s, \theta^i_0, \theta^i_1, ..., \theta^i_{P(\eta)-1+i})$,
...
$\lambda s.F_{P(\eta)}(s, \theta^1_0, \theta^1_1, ..., \theta^1_{P(\eta)})$
return s

The adversary $\mathcal{C}^i$ can be generated in polynomial time. Adversary $\mathcal{B}^i$ uses adversary $\mathcal{C}^{i-1}$ as a sub-routine. His oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ are respectively linked to $f \circ LR^b$ and f'.

Adversary $\mathcal{B}^i(\eta)/\mathcal{O}_1, \mathcal{O}_2$
$\theta_0 := \Theta_0(\eta)$
for j **in** $[1, P(\eta) - i]$:
$\theta'_j := \Theta(\eta)$
$\theta_j := \Theta(\eta)$
s:=$\mathcal{C}^{i-1}(\eta)/\lambda s.F_0(s, \theta_0)$,
...
$\lambda s.F_{P(\eta)+1-k}(s, \theta_0, ..., \theta_{P(\eta)+1-k})$,
$\lambda s.\mathcal{O}_1\big\langle g(s, [\theta'_1, ..., \theta'_{P(\eta)+1-k}, \theta_{P(\eta)-k}]), g(s, [\theta_1, ..., \theta_{P(\eta)+1-k}, \theta_{P(\eta)-k}])\big\rangle$,
$\lambda s.\mathcal{O}_2(s, \theta_{P(\eta)-k})$
if $V_0(s, \theta_0)$ **return** 1
else return 0

Generating the adversary $\mathcal{B}^i$ can also be done in a polynomial time. To compute the relative advantage of the different adversaries $\mathcal{B}^i$ and $\mathcal{C}^i$, we use the same reasoning as in the proof of theorem 5.1. For the sake of simplicity, we do not describe formally the different games. This could be done in a similar way as in the classical proof. When the challenge bit b equals 1, the game involving $\mathcal{B}^{i+1}$ against β is equivalent to the game involving $\mathcal{C}^i$ against δ_i. Thus, the probabilities of success are the same:

$$Pr[\mathbf{G}^{\beta}_{\mathcal{B}^i}(\eta) = true | b = 1] = Pr[\mathbf{G}^{\delta_i}_{\mathcal{C}^i}(\eta) = true]$$

When the challenge bit b equals 0, the game involving $\mathcal{B}^{i+1}$ against β is equivalent to the game involving $\mathcal{C}^{i+1}$ against δ_{i+1}

$$Pr[\mathbf{G}^{\beta}_{\mathcal{B}^i}(\eta) = true | b = 0] = Pr[\mathbf{G}^{\delta_{i+1}}_{\mathcal{C}^{i+1}}(\eta) = true]$$

By subtracting the second equation from the first one, we get the advantage of $\mathcal{B}^i$:

$$\mathbf{Adv}^{\beta}_{\mathcal{B}^i} = Pr[\mathbf{G}^{\delta_i}_{\mathcal{C}^i}(\eta) = true] - Pr[\mathbf{G}^{\delta_{i+1}}_{\mathcal{C}^{i+1}}(\eta) = true]$$

Moreover, criterion δ_i and δ_{i+1} have the same verifier V_0 that only uses a common challenge generated by Θ_0. Thus, the $PrRand$ for these two criteria is the same. For any i and j, we have that:

$$PrRand^{\delta_i} = PrRand^{\delta_j} = PrRand^{\delta_0}$$

This can be used in the previous equation:

$$\mathbf{Adv}^{\beta}_{\mathcal{B}^i} = \big(Pr[\mathbf{G}^{\delta_i}_{\mathcal{C}^i}(\eta) = true] - PrRand^{\delta_i}\big) - \big(Pr[\mathbf{G}^{\delta_{i+1}}_{\mathcal{C}^{i+1}}(\eta) = true] - PrRand^{\delta_{i+1}}\big)$$

By using the definition of advantage, we get the following equality:

$$\forall i \in [0, P(\eta) - 1],\ \mathbf{Adv}^{\delta_i}_{\mathcal{C}^i}(\eta) = 2.\mathbf{Adv}^{\beta}_{\mathcal{B}^{i+1}}(\eta) + \mathbf{Adv}^{\delta_{i+1}}_{\mathcal{C}^{i+1}}(\eta)$$

Criterion δ_0 is criterion γ. Criterion $\delta_{P(\eta)}$ is criterion δ. Moreover, adversary $\mathcal{C}^0$ is the same as adversary $\mathcal{A}$. Hence by summing these lines, we get:

$$\mathbf{Adv}^{\gamma_P}_{\mathcal{A}}(\eta) = 2\sum_{i=1}^{P(\eta)} \mathbf{Adv}^{\beta}_{\mathcal{B}^i}(\eta) + \mathbf{Adv}^{\delta}_{\mathcal{C}^{P(\eta)}}(\eta)$$

Adversary $\mathcal{D}$ is easy to deduce from adversary $\mathcal{B}^i$. This adversary plays against β. He randomly chooses an integer i in $[1, P(\eta)]$ and executes adversary $\mathcal{B}^i$.

Adversary $\mathcal{D}/\mathcal{O}_1, \mathcal{O}_2$
  $i \xleftarrow{R} [1, P(\eta)]$
  return $\mathcal{B}^i/\mathcal{O}_1, \mathcal{O}_2$

The advantage of $\mathcal{D}$ is given by:

$$\begin{aligned}
\mathbf{Adv}^{\beta}_{\mathcal{D}}(\eta) &= Pr[\mathbf{G}^{\beta}_{\mathcal{D}}(\eta) = true] - PrRand^{\beta}(\eta)\big) \\
&= \frac{1}{P(\eta)}\sum_{j=1}^{P(\eta)} \big(Pr[\mathbf{G}^{\beta}_{\mathcal{D}}(\eta) = true | i = j] - PrRand^{\beta}(\eta)\big) \\
&= \frac{1}{P(\eta)}\sum_{j=1}^{P(\eta)} \mathbf{Adv}^{\beta}_{\mathcal{B}^i}(\eta)
\end{aligned}$$

We finally get that:

$$\forall \eta,\ \mathbf{Adv}^{\gamma_P}_{\mathcal{A}}(\eta) = 2P(\eta)\mathbf{Adv}^{\beta}_{\mathcal{D}}(\eta) + \mathbf{Adv}^{\delta}_{\mathcal{C}^{P(\eta)}}(\eta)$$

■

Using the previous theorem, we reduce some criteria that involve a polynomial number of challenges to equivalent "atomic" criteria. We first consider the case of P-AC-IND-CPA, an extension of IND-CPA that handles a certain form of key decommitment.

8.1.3 The P-AC-IND-CPA Criterion

The P-AC-IND-CPA criterion is another intuitive extension of IND-CPA. A polynomial number of key pairs are generated ($P(\eta)$) along with a challenge bit b. Then the adversary can ask for the public part of each key, he also has access to a left-right oracle for each key pair (pk_i, sk_i): the adversary submits a pair of bit-strings of the same length $\langle bs_0, bs_1\rangle$ and receives the encryption of bs_b using pk_i, $\mathcal{E}(bs_b, pk_i)$. The main change with respect to IND-CPA is that this criterion handles a kind of decommitment: the adversary can revoke any key (this can be viewed as corrupting an honest agent). For this reason, the adversary may ask for the secret part of any key. In this case, he must not have used the left-right oracle related to the key (otherwise it would be easy to win) and will not have access to this oracle any more. This is called corruption of a key and AC stands for Adaptive Corruption as the adversary can choose which keys to corrupt one after another.

Let $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme and P be a positive polynomial. We build a new criterion based on IND-CPA with the following features: the adversary has access to the classical left-right oracle and the public key oracle for a polynomial number of keys; he may also ask for revelation of the secret part of some of the keys with some restrictions. Let P-AC-IND-CPA be the criterion $(\Theta_0; \Theta; F_0; F; V_0)$ where:

- Θ_0 generates a random bit b;
- Θ generates a pair of matching public and secret keys (pk, sk), together with two booleans, can_reveal and $is\_revealed$, that are respectively initialized to $true$ and $false$ (these booleans are used to implement restrictions on secret key revelation);

- F_0 is the empty oracle;
- F is the family of the oracles F_i, each one containing three oracles. The first oracle, rel_pk_i, outputs the public key pk_i, the second one is the left-right encryption oracle denoted by enc_pk_i, and the third one, rel_sk_i, that can release the secret key sk_i. Implementations for the oracles are detailed below.
- $V_0(s, \theta_0)$ returns $true$ iff $s = b$.

The encryption oracle takes as argument a pair of bit-strings $\langle bs_0, bs_1 \rangle$, checks that the key has not been revealed and returns the encryption of m_b.

Oracle $enc\_pk_i(\langle bs_0, bs_1 \rangle)$:
  if $is\_revealed$ **then**
    return $\perp$
  else
    $can\_reveal := false$
    return $\mathcal{E}(bs_b, pk_i)$

The second oracle reveals the secret part of the key. It verifies that the key has not been used previously with the left-right oracle.

Oracle rel_sk_i :
  if can_reveal **then**
    $is\_revealed := true$
    return sk_i
  else
    return $\perp$

Finally, the last oracle gives access to the public part of the key.

Oracle rel_pk_i :
  return pk_i

The objective is now to prove the equivalence between this new criterion and IND-CPA. First we discuss a simplified version of this equivalence. Instead of considering this criterion with a polynomial number of keys, we take the case where the number of keys is bounded by a constant N. This criterion is denoted by N-AC-IND-CPA (γ_N). We first prove the equivalence between N-AC-IND-CPA and N-IND-CPA. The equivalence between N-IND-CPA and 1-IND-CPA is a well-known result.

Let $\mathcal{A}$ be an adversary against γ_N. Then there exists an adversary $\mathcal{B}$ against N-IND-CPA whose advantage is "similar" to the advantage of $\mathcal{A}$. The adversary B randomly chooses a sub-set R of $[1, N]$. With probability $1/2^N$, these will be the keys whose inverse is asked by $\mathcal{A}$. Therefore $\mathcal{B}$ simulates these keys (by generating them using $\mathcal{KG}$). If $\mathcal{A}$ asks a left-right encryption using one of these keys, $\mathcal{B}$ randomly returns 0 or 1. Else $\mathcal{B}$ returns the result of $\mathcal{A}$.

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_N}(\eta) \leq 2^N \mathbf{Adv}_{\mathcal{B}}^{N-IND-CPA}(\eta)$$

This allows us to reduce N-AC-IND-CPA to IND-CPA and to obtain the following proposition.

Proposition 8.1 *For any integer $N > 0$, an asymmetric encryption scheme is secure against N-AC-IND-CPA iff it is secure against IND-CPA.*

Proof: We proved that security against IND-CPA implies security against N-AC-IND-CPA. The converse is immediate. Let $\mathcal{A}$ be an adversary against IND-CPA, $\mathcal{A}$ can be used against N-AC-IND-CPA and the advantages are the same in both cases. Thus if we have security against

N-AC-IND-CPA, for any adversary $\mathcal{A}$ against IND-CPA, the advantage of $\mathcal{A}$ against N-AC-IND-CPA is negligible. The advantage of $\mathcal{A}$ against IND-CPA is the same and so is also negligible. For this reason, security against N-AC-IND-CPA implies security against IND-CPA. ■

However, this method cannot be generalized to the case of P-AC-IND-CPA as $2^{P(\eta)}$ makes that the right part of the inequality is not negligible anymore. Therefore, we use our new partition theorem to prove this result. For this purpose, let us verify the assumptions of our theorem.

- V_0 only depends on b which is generated by Θ_0, F_0 is the empty oracle so it only depends on θ_0.
- Oracle F_i is composed of a part f' containing the public key oracle and the secret key revelation oracle. It also contains a part $f(g(s, [\theta_0, ..., \theta_{i-1}]), \theta_i)$ where $g(\langle m_0, m_1\rangle, [\theta_0, ..., \theta_{i-1}])$ returns m_b (b is the challenge bit from θ_0 and f makes the encryption using key pk_i (from θ_i) and the related checks.

Then if we apply the partition theorem, $\beta = (\Theta, b; f \circ LR^b, f'; v_b)$ is criterion 1-AC-IND-CPA and $\delta = (\Theta_0; F_0; V_0)$ is a criterion where the adversary has to guess the value of a bit b without any oracle (as F_0 is the empty oracle). Let $\mathcal{A}$ be an adversary against P-AC-IND-CPA, there exist two adversaries $\mathcal{B}$ and $\mathcal{C}$, such that :

$$\forall \eta,\ \mathbf{Adv}^{\gamma_P}_{\mathcal{A}}(\eta) = 2P(\eta)\mathbf{Adv}^{\beta}_{\mathcal{B}}(\eta) + \mathbf{Adv}^{\delta}_{\mathcal{C}}(\eta)$$

The advantage of $\mathcal{C}$ has to be zero, hence P-AC-IND-CPA is equivalent to 1-AC-IND-CPA. As there is only one key left, the technique presented above applies and 1-AC-IND-CPA is equivalent to IND-CPA. It is now easy to conclude:

Proposition 8.2 *For any positive polynomial P, an encryption scheme is secure against P-AC-IND-CPA iff it is secure against IND-CPA.*

Note that we only prove that security against IND-CPA implies security against P-AC-IND-CPA. The other implication is obvious.

8.1.4 Adaptive Corruption

The adaptive corruption problem can be described as follows: $P(\eta)$ keys are randomly sampled and the adversary is given the encryption of bit-string 0 for each of these keys. Then the adversary chooses half of the keys and receives their bit-string values. Then the question is: are the other keys still secure ? The intuitive answer is yes and our last result allows us to prove it.

Let us first give a formal definition to the adaptive corruption problem. An adversary is composed of two stages $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ receives some cipher-texts and has to output a subset U of $[1, P(\eta)]$ which contains exactly half of the elements (let V be the other elements of $[1, P(\eta)]$). $\mathcal{A}_2$ receives the values of the related keys and is faced to the classical IND-CPA game on the other keys.

Game $\mathbf{G}^{AC}_{\mathcal{A}_1,\mathcal{A}_2}(\eta)$:
for i **from** 1 **to** $P(\eta)$
$pk_i, sk_i := KG(\eta)$
$U, mem := \mathcal{A}_1(\mathcal{E}(0, pk_1), ..., \mathcal{E}(0, pk_{P(\eta)}))$
$b \xleftarrow{R} \{0, 1\}$
$d := \mathcal{A}_2(pk_{U[1]}, ..., pk_{U[P(\eta)/2]}, mem)/\lambda\langle bs_0, bs_1\rangle.\mathcal{E}(bs_b, pk_{V[1]})$
...
$\lambda\langle bs_0, bs_1\rangle.\mathcal{E}(bs_b, pk_{V[P(\eta)/2]})$
return $d = b$

The advantage of an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ against AC is given by:

$$\mathbf{Adv}^{AC}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 2.Pr[\mathbf{G}^{AC}_{\mathcal{A}_1,\mathcal{A}_2}(\eta) = 1] - 1$$

Then using the previous proposition it is easy to prove that IND-CPA implies adaptive corruption.

Proposition 8.3 *Let $\mathcal{SE}$ be an IND-CPA symmetric encryption scheme, then for any adversary $(\mathcal{A}_1, \mathcal{A}_2)$, the advantage of $(\mathcal{A}_1, \mathcal{A}_2)$ against SD is negligible.*

Proof: Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against SD. Then it is easy to build an adversary $\mathcal{A}$ against P-AC-IND-CPA whose advantage is the same as the advantage of $(\mathcal{A}_1, \mathcal{A}_2)$. Adversary $\mathcal{A}$ uses $\mathcal{A}_1$ and $\mathcal{A}_2$ in the following way: $\mathcal{A}$ uses his public-key oracles to compute $\mathcal{E}(0, pk_i)$ for each challenge key pk_i. $\mathcal{A}_1$ outputs a subset U of $[1, P(\eta)]$. Then $\mathcal{A}$ uses his revelation oracles to get the keys related to U. These keys are given to $\mathcal{A}_2$. When $\mathcal{A}_2$ issues a query, this query is given to the corresponding left-right oracle of $\mathcal{A}$. As queries of $\mathcal{A}_2$ concern keys that are not in U, $\mathcal{A}$ did not ask for revelation of these keys and the left-right oracles can still be used. Finally, $\mathcal{A}$ outputs the same result as $\mathcal{A}_2$. The games involving $\mathcal{A}$ and $(\mathcal{A}_1, \mathcal{A}_2)$ are the same thus their respective advantages are similar.

Moreover using proposition 8.1, the advantage of $\mathcal{A}$ is negligible. Therefore the advantage of $(\mathcal{A}_1, \mathcal{A}_2)$ is also negligible. ■

8.2 Generalization of the Complete Partition Theorem

In the previous section, we only treated the case of theorem 5.1. In this section, we extend this result to the case of theorem 5.2. The main difficulty is that the verifier does not only depend on θ_0 anymore. It can also depend on the variously generated θ_i. We first have to define iterated criteria with multiple verifiers. After that, we extend our partition theorem to the case of such criteria. An example of iterated criterion with multiple verifiers is P-SYM-CPA where P is a polynomial. An adversary against this criterion can win his challenge by producing a valid encryption for any of the $P(\eta)$ challenge keys.

8.2.1 Iterated Criterion with Multiple Verifiers

Let P be a positive polynomial. In order to introduce multiple verifiers, we add to the definition of iterated criteria a new verifier V. This verifier takes as arguments the output of the adversary and a θ_i produced by Θ. Hence this verifier can be instantiated for any i between 1 and $P(\eta)$. To win his game, an adversary as either to satisfy V_0 or one of the V_i.

An iterated criterion with multiple verifiers γ is a tuple $(\Theta_0; \Theta; F_0; F; V_0; V)$ where:

- Θ_0 is the challenge generator that is only called once.
- Θ is the *single challenge generator*.
- F_0 is the oracle related to Θ_0.
- F is the oracle related to a challenge generated by Θ.
- Verifier V_0 checks that the output of the adversary is correct with respect to θ_0 produced by Θ_0.
- Verifier V checks that the output of the adversary is correct with respect to a θ_i produced by Θ.

As before, oracle F_0 only uses challenge θ_0. Oracle F uses challenges θ_i and can also use challenges θ_j for j lower than i. Hence oracle F can have a variable number of parameters. Verifier V_0 only depends on θ_0 and verifier V only depends on a single θ_i generated by Θ. The verifier V that uses θ_i is also denoted by V_i.

Let $\mathcal{A}$ be an adversary against γ. Then the experiment involving $\mathcal{A}$ is close to the experiment in the case of a single verifier. It depends on the positive polynomial P giving the number of

challenges. Notation γ_P is still used to denote criterion γ where the challenge generator Θ is iterated $P(\eta)$ times. We define a different experiment for each verifier (thus there are $P(\eta)+1$ different experiments). The experiments proceed as in the previous section: challenge θ_0 is generated using Θ_0. Challenges θ_1 to $\theta_{P(\eta)}$ are generated using Θ. Adversary $\mathcal{A}$ is executed and has access to oracle F_0 and to oracles F_i where i ranges between 1 and $P(\eta)$. Finally, the adversary outputs his result d and the verifier (V_0 or one of the V_i) is called to check that the output is correct. The games can be detailed by the following procedure. Note that the game depends on an integer j specifying which verifier is used.

Game $\mathbf{G}_{\mathcal{A}}^{\gamma_P,j}(\eta)$:
θ_0:=$\Theta_0(\eta)$
for i **from** 1 **to** $P(\eta)$
θ_i:=$\Theta(\eta)$
d:=$\mathcal{A}(\eta)/\lambda s.F_0(s,\theta_0),$
$\lambda s.F(s,\theta_0,\theta_1)$
$\ldots$
$\lambda s.F(s,\theta_0,\theta_1,\ldots,\theta_{P(\eta)})$
return $V_j(d,\theta_j)$

We define a different advantage for each of the different values of j:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_P,j}(\eta) = 2.\big(Pr[\mathbf{G}_{\mathcal{A}}^{\gamma_P,j}(\eta)=true] - PrRand^{\gamma_P,j}(\eta)\big)$$

As verifier V_0 only depends on θ_0, $PrRand^{\gamma_0,0}$ is equal to $PrRand$ in the case of criterion $(\Theta_0;F_0;V_0)$. The advantage in this case is given by:

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_P,0}(\eta) = 2.\big(Pr[\mathbf{G}_{\mathcal{A}}^{\gamma_P,0}(\eta)=true] - PrRand^{(\Theta_0;F_0;V_0)}(\eta)\big)$$

For the other verifiers V, $PrRand^{\gamma_P,j}$ is equal to $PrRand$ in the case of criterion $(\Theta;\epsilon;V)$. Hence the advantage in this other case is given by the following for j between 1 and $P(\eta)$.

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_P,j}(\eta) = 2.\big(Pr[\mathbf{G}_{\mathcal{A}}^{\gamma_P,j}(\eta)=true] - PrRand^{(\Theta;\epsilon;V)}(\eta)\big)$$

Finally, the advantage is defined as the maximum advantage against any of the possible verifiers (this is similar to what is done for multiple verifiers in section 4.2).

$$\mathbf{Adv}_{\mathcal{A}}^{\gamma_P}(\eta) = \max_{0\leq i\leq P(\eta)}\big(\mathbf{Adv}_{\mathcal{A}}^{\gamma_P,i}(\eta)\big)$$

We also say that a cryptographic scheme is *safe* for iterated criterion γ and polynomial P if for any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ against γ_P is negligible (the cryptographic scheme is used to implement the criterion). Hence, using max in the definition, we say that a cryptographic scheme is safe for a criterion iff it is safe for any criteria that use only one of the verifiers.

Using this new class of criteria, it is possible to generalize theorem 5.2 so that it can handle a polynomial number of challenges.

8.2.2 Partition Theorem in the General Case

The partition theorem in its generalized form can be adapted to the case of iterated criteria with multiple verifiers. The result is that the advantage of any adversary against an iterated criterion $(\Theta_0;\Theta;F_0;F;V_0;V)$ can be linked to the advantage of an adversary against $(\Theta_0;F_0;V_0)$, to the advantage of an adversary against an indistinguishability criterion and to the advantage of an adversary against a criterion using verifier V. The main use of this theorem is to prove safety of a scheme for γ_P. For this purpose, it is sufficient to prove safety of the scheme for the three "small" criteria.

Theorem 8.2 (Partition Theorem, Polynomial Case) *Let P be a positive polynomial. Let γ_P be the criterion*

$$\gamma_P = (\Theta_0; \Theta; F_0; F; V_0; V)$$

We assume the following hypothesis: there exist three functions f, f' and g (that can be represented by PRTMs) such that for any i oracle F_i consists in both:

$$\lambda s \cdot f(g(s, [\theta_0, \ldots, \theta_{i-1}]), \theta_i) \text{ and } \lambda s \cdot f'(s, \theta_i)$$

Then, for any adversary $\mathcal{A}$ against criterion γ_P, there exist three adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ verifying the following equality:

$$\forall \eta, \ \boldsymbol{Adv}_{\mathcal{A}}^{\gamma_P}(\eta) = 2P(\eta)^2 \boldsymbol{Adv}_{\mathcal{B}}^{\beta}(\eta) + \boldsymbol{Adv}_{\mathcal{C}}^{\delta}(\eta) + P(\eta) \boldsymbol{Adv}_{\mathcal{D}}^{\sigma}(\eta)$$

where $\beta = (\Theta, b; f \circ LR^b, f'; v_b)$ is indeed an indistinguishability criterion, $\delta = (\Theta_0; F_0; V_0)$ is our "main" criterion and $\sigma = (\Theta; f, f'; V)$ is the criterion related to verifier V.

Proof: This proof is very close to the proof of theorem 8.1. The main difference is that instead of iterating the simplified partition theorem, the full partition theorem is iterated here. Once more, the idea is to decompose the advantage of $\mathcal{A}$ into the advantage of three machines $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, then to iterate the reasoning on the advantage of $\mathcal{C}$. The key of this proof is to verify that the machines generated by the partition theorem have a common form which allows for the automatic generation of these machines.

The technical part of the proof consists in formally describing machines $\mathcal{B}^i$, $\mathcal{C}^i$ and $\mathcal{D}^i$. The form of these machines comes from the proof of theorem 5.2. To ensure equalities instead of inequalities, adversaries $\mathcal{B}^i$, $\mathcal{C}^i$ and $\mathcal{D}^i$ can either be of the following form or be adversaries that have a null advantage.

Adversary $\mathcal{C}^i$ plays against criterion δ_i which is equivalent to criterion γ iterated $P(\eta) - i$ times, this criterion is equivalent to criterion γ_{P-i}.

Adversary $\mathcal{C}^i(\eta)/\mathcal{O}_1, ..., \mathcal{O}_{P(\eta)-i}$:
  for j **in** $[1, i]$:
    θ_0^j:=$\Theta_0(\eta)$
    θ_1^j:=$\Theta(\eta)$
    ...
    $\theta_{P(\eta)+1-j}^j$:=$\Theta(\eta)$
  s:=$\mathcal{A}(\eta)/\mathcal{O}_1, ..., \mathcal{O}_{P(\eta)-i}$,
      $\lambda s.F_{P(\eta)+1-i}(s, \theta_0^i, \theta_1^i, ..., \theta_{P(\eta)-1+i}^i)$,
      ...
      $\lambda s.F_{P(\eta)}(s, \theta_0^1, \theta_1^1, ..., \theta_{P(\eta)}^1)$
  return s

The adversary $\mathcal{C}^i$ can be generated in polynomial time. Adversary $\mathcal{B}_j^i$ uses adversary $\mathcal{C}^{i-1}$ as a subroutine. His oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ are respectively linked to $f \circ LR^b$ and f'. Index j denotes the verifier that the adversary is using.

Adversary $\mathcal{B}_j^i(\eta)/\mathcal{O}_1, \mathcal{O}_2$
  $\theta_0 := \Theta_0(\eta)$
  for j **in** $[1, P(\eta) - i]$:
    $\theta'_j := \Theta(\eta)$
    $\theta_j := \Theta(\eta)$
  s:=$\mathcal{C}^{i-1}(\eta)/\lambda s.F_0(s, \theta_0)$,
      ...
      $\lambda s.F_{P(\eta)+1-k}(s, \theta_0, ..., \theta_{P(\eta)+1-k})$,
      $\lambda s.\mathcal{O}_1\big\langle g(s, [\theta'_1, ..., \theta'_{P(\eta)+1-k}, \theta_{P(\eta)-k}]), g(s, [\theta_1, ..., \theta_{P(\eta)+1-k}, \theta_{P(\eta)-k}])\big\rangle$,
      $\lambda s.\mathcal{O}_2(s, \theta_{P(\eta)-k})$

if $V_j(s,\theta_0)$ **return** 1
else return 0

Generating the adversary $\mathcal{B}^i$ can also be done in a polynomial time.

Finally, adversary $\mathcal{D}^i$ uses adversary $\mathcal{C}^{i-1}$ in order to play against criterion σ defined by the triple $(\Theta; f, f'; V)$.

Adversary $\mathcal{D}^i(\eta)/\mathcal{O}_1, \mathcal{O}_2$
$b \xleftarrow{R} [0,1]$
$\theta_0 := \Theta_0(\eta)$
for j **in** $[1, P(\eta) - i]$:
$\theta'_j := \Theta(\eta)$
$\theta_j := \Theta(\eta)$
$s{:=}\mathcal{C}^{i-1}(\eta)/\lambda s.F_0(s,\theta_0),$
$\quad ...$
$\quad \lambda s.F_{P(\eta)+1-k}(s,\theta_0,...,\theta_{P(\eta)+1-k}),$
$\quad \lambda s.\mathcal{O}_1 \circ LR^b\langle g(s,[\theta'_1,...,\theta'_{P(\eta)+1-k},\theta_{P(\eta)-k}]), g(s,[\theta_1,...,\theta_{P(\eta)+1-k},\theta_{P(\eta)-k}])\rangle,$
$\quad \lambda s.\mathcal{O}_2(s,\theta_{P(\eta)-k})$
return s

Generating these last adversaries $\mathcal{D}^i$ can also be done in a polynomial time under the same hypothesis as $\mathcal{C}_i$.

To compute the relative advantages of the different adversaries $\mathcal{B}^i$, $\mathcal{C}^i$ and $\mathcal{D}^i$ we use the same reasoning as in the proof of theorem 5.2. For the sake of simplicity, we do not describe formally the different games. This could be done in a similar way as in the classical proof.

Criterion δ_i^j denotes criterion δ_i where V_j is the only verifier considered. When the challenge bit b equals 1, the game involving $\mathcal{B}_j^i$ against β is equivalent to the game involving $\mathcal{C}^i$ against δ_i Thus, the probabilities of success are the same:

$$Pr[\mathbf{G}^{\beta}_{\mathcal{B}^i_j}(\eta) = true | b = 1] = Pr[\mathbf{G}^{\delta_i^j}_{\mathcal{C}^i}(\eta) = true]$$

When the challenge bit b equals 0, the game involving $\mathcal{B}_j^i$ against β is equivalent to the game involving $\mathcal{C}^{i+1}$ against δ_{i+1}

$$Pr[\mathbf{G}^{\beta}_{\mathcal{B}^i_j}(\eta) = true | b = 0] = Pr[\mathbf{G}^{\delta_{i+1}^j}_{\mathcal{C}^{i+1}}(\eta) = true]$$

Using these two equations, we get the advantage of $\mathcal{B}^i$:

$$\mathbf{Adv}^{\beta}_{\mathcal{B}^i_j} = Pr[\mathbf{G}^{\delta_i^j}_{\mathcal{C}^i}(\eta) = true] - Pr[\mathbf{G}^{\delta_{i+1}^j}_{\mathcal{C}^{i+1}}(\eta) = true]$$

Moreover, criteria δ_i^j and δ_{i+1}^j have the same verifier V_j that only uses a common challenge generated by Θ_j. Thus, the $PrRand$ for these two criteria is the same. For any i and i', we have that:

$$PrRand^{\delta_i^j} = PrRand^{\delta_{i'}^j} = PrRand^{\sigma}$$

This can be used in the previous equation:

$$\mathbf{Adv}^{\beta}_{\mathcal{B}^i_j} = \left(Pr[\mathbf{G}^{\delta_i^j}_{\mathcal{C}^i}(\eta) = true] - PrRand^{\delta_i^j}\right) - \left(Pr[\mathbf{G}^{\delta_{i+1}^j}_{\mathcal{C}^{i+1}}(\eta) = true] - PrRand^{\delta_{i+1}^j}\right)$$

By using the definition of advantage, we get the following equality:

$$\forall i \in [0, P(\eta) - 1],\ \mathbf{Adv}^{\delta_i^j}_{\mathcal{C}^i}(\eta) = 2\mathbf{Adv}^{\beta}_{\mathcal{B}^i_j}(\eta) + \mathbf{Adv}^{\delta_{i+1}^j}_{\mathcal{C}^{i+1}}(\eta)$$

Criterion δ_0^0 is criterion γ.

$$\forall i \in [0, P(\eta) - 1],\ \mathbf{Adv}^{\delta_i^i}_{\mathcal{C}^i}(\eta) = \mathbf{Adv}^{\sigma}_{\mathcal{D}^i}(\eta)$$

Hence by using the definition of advantage with multiple verifiers, we get that for any i between 0 and $P(\eta) - 1$,

$$\mathbf{Adv}^{\delta_i^i}_{\mathcal{C}^i}(\eta) = \left(\sum_j 2\mathbf{Adv}^{\beta}_{\mathcal{B}_j^i}(\eta) \right) + \mathbf{Adv}^{\delta_{i+1}^i}_{\mathcal{C}^{i+1}}(\eta) + \mathbf{Adv}^{\sigma}_{\mathcal{D}^i}(\eta)$$

Finally, adversary $\mathcal{D}$ chooses a random integer i between 1 and $P(\eta)$ and executes $\mathcal{D}^i$. Adversary $\mathcal{B}$ generates two integers i and j and executes $\mathcal{B}_j^i$. Thus, we finally get the theorem's main result:

$$\forall \eta,\ \mathbf{Adv}^{\gamma_P}_{\mathcal{A}}(\eta) = 2P(\eta)^2\mathbf{Adv}^{\beta}_{\mathcal{B}}(\eta) + \mathbf{Adv}^{\delta}_{\mathcal{C}}(\eta) + P(\eta)\mathbf{Adv}^{\sigma}_{\mathcal{D}}(\eta)$$

■

Using the previous theorems, we reduce some criteria that involve a polynomial number of challenges to equivalent "atomic" criteria. We first consider the case of AC-PAT-SYM-CPA, an extension of SYM-CPA that handles key corruption as in AC-IND-CPA. We use symmetric encryption in this example in order to have multiple verifiers: to win his challenge, an adversary can either guess the random bit b or for any challenge key he can forge an encryption using this key that has not been produced by the encryption oracle (the forged encryption has to be *fresh*).

8.2.3 The P-AC-PAT-SYM-CPA Criterion

The P-AC-PAT-SYM-CPA criterion is an extension of the SYM-CPA criterion for symmetric encryption schemes. Let $\mathcal{SE}$ be a symmetric encryption scheme composed of the key generation algorithm $\mathcal{KG}$, the encryption algorithm $\mathcal{E}$ and the decryption algorithm $\mathcal{D}$. In this new criterion, $P(\eta)$ different symmetric keys are generated using $\mathcal{KG}$. A random challenge bit b is also sampled. The adversary can query three oracles for each key k:

1. The first oracle takes as argument a pair of patterns $\langle pat_0, pat_1 \rangle$ and returns the encryption using k of the valuation of pattern pat_b using the *concr* algorithm. There is still an acyclicity restriction that prevents a key k to occur encrypted by itself. This oracle is called *left-right oracle* related to key k.

2. The second oracle takes as argument a bit-string bs and returns the encryption of bs using k. This is useful to represent the classical key corruption problem. The adversary can ask for corruption of a key after seeing cipher-texts using that key, however these cipher-texts should not contain useful information for the adversary (hence this is not a left-right encryption oracle but just an encryption oracle). Corruption of the key is still possible after using this oracle.

3. The third oracle is related to corruption. The adversary can ask to see the value of a challenge key k. In this situation, the adversary must not have queried the left-right oracle related to k before and it will not have access to this oracle after the revelation of k.

There are multiple ways to win in this criterion. An adversary can either find the value of bit b or forge a "fresh" encryption using one of the challenge keys.

Formally, the P-AC-PAT-SYM-CPA criterion is represented by an iterated criterion using multiple verifiers. This criterion is defined by the tuple $(\Theta_0; \Theta; F_0; F; V_0; V)$ where each component can be precisely described:

- Θ_0 generates the challenge bit b;

- Θ generates a symmetric key using the key generation algorithm $\mathcal{KG}$. It also generates two booleans, *can_reveal* and *is_revealed*, that are respectively initialized to *true* and $false$ (these booleans are used to implement restrictions on secret key revelation). It finally generates a shared memory *mem* that is used to store previous outputs of the left-right oracle;
- F_0 is the empty oracle;
- F is composed of three oracles: the first one is the classical left-right oracle related to patterns, the second one is the encryption oracle, the third one is the corruption oracle which allows the adversary to get the value of the key. The formal definitions for these oracles are given thereafter. The encryption oracle takes as argument a pair of patterns $\langle pat_0, pat_1\rangle$, checks that the key has not been revealed and returns the encryption of the concretization of pat_b. The result of the encryption is also stored in *mem* for verifier V (as V has to check that the output of $\mathcal{A}$ is a *fresh* message using the symmetric key).

  Oracle $enc\_k(\langle pat_0, pat_1\rangle, \theta_0, ..., \theta_i)$:
  if $is\_revealed\theta_i$ **then**
  return $\perp$
  else
  $can\_reveal\theta_i := false$
  $out := \mathcal{E}(concr(pat_{b\theta_0}, \theta_1 ... \theta_{i-1}), k\theta_i)$
  $mem\theta_i := out :: mem\theta_i$
  return *out*

  The second oracle takes as input a bit-string and simply returns its encryption. The result also has to be stored for verifier V.

  Oracle $enc'\_k(bs, \theta_0, ..., \theta_i)$:
  $out := \mathcal{E}(bs, k\theta_i)$
  $mem\theta_i := out :: mem\theta_i$
  return *out*

  The third oracle reveals the key. It verifies that the key has not been used previously with the left-right oracle.

  Oracle $rel\_k(\theta_0, ..., \theta_i)$:
  if $can\_reveal\theta_i$ **then**
  $is\_revealed\theta_i := true$
  return k_i
  else
  return $\perp$

- V_0 tests that the adversary correctly guessed the challenge bit b from θ_0.
- V tests that the adversary has produced a valid encryption for key k from θ. This encryption has to be different from all the bit-strings that were stored in the *mem* mutable field from θ.

The objective is now to prove the equivalence between this new criterion and SYM-CPA. As in the case of P-AC-IND-CPA, proving the equivalence is easy if P is a constant polynomial but this is not trivial in any other cases. For this reason, we use this generalized partition theorem to prove the result.

Proposition 8.4 *For any positive polynomial P, a symmetric encryption scheme is secure against P-AC-PAT-SYM-CPA iff it is secure against SYM-CPA.*

Proof: First, it is trivial to show that if an encryption scheme is secure against P-AC-PAT-SYM-CPA, then it is secure against SYM-CPA.

Theorem 8.2 can be used to prove the converse. Let γ_P denote criterion P-AC-PAT-SYM-CPA, γ is defined as an iterated criterion. In order to apply this theorem, oracle F can be decomposed in two parts: the f' part that contains the revelation oracle rel_k and the encryption oracle enc'_k which only depend on θ_i and a second part that contains the left-right encryption oracle enc_k. This second part can itself be decomposed in two layers:

$$enc\_k(s, \theta_0, ..., \theta_i) = f(g(s, [\theta_0, \ldots, \theta_{i-1}]), \theta_i)$$

Where $g(\langle pat_0, pat_1\rangle, [\theta_0, \ldots \theta_{i-1}])$ returns the concretization of pat_b where bit b is taken from θ_0. Formally,

$$g(\langle pat_0, pat_1\rangle, [\theta_0, \ldots \theta_{i-1}]) = concr(pat_{b\theta_0}, \theta_1 \ldots \theta_{i-1})$$

The f layers contains all the operations related to θ_i, it is defined by the following algorithm. Because of acyclicity, the argument of this algorithm is a bit-string and not a pattern.

Algorithm $f(bs, \theta_i)$:
  if $is\_revealed\theta_i$ **then**
    return $\perp$
  else
    $can\_reveal\theta_i := false$
    $out := \mathcal{E}(bs, k\theta_i)$
    $output\theta_i := out :: output\theta_i$
    return out

The hypotheses of the polynomial partition theorem hold therefore for any adversary $\mathcal{A}$ there exist three adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ such that:

$$\forall \eta,\ \mathbf{Adv}^{\gamma_P}_{\mathcal{A}}(\eta) = 2P(\eta)^2\mathbf{Adv}^{\beta}_{\mathcal{B}}(\eta) + \mathbf{Adv}^{\delta}_{\mathcal{C}}(\eta) + P(\eta)\mathbf{Adv}^{\sigma}_{\mathcal{D}}(\eta)$$

where the different criteria are given by:

- $\beta = (\Theta, b; f \circ LR^b, f'; v_b)$ is an indistinguishability criterion. A challenge bit b and a symmetric key k are randomly generated. Arguments of the $f \circ LR^b$ oracle can only be bit-strings, this is the classical left-right encryption oracle. The verifier checks that the adversary correctly guessed the value of b. This criterion is close to SYM-CPA except that if the key has not been used, it can be revealed using the f' oracle.

- $\delta = (\Theta_0; F_0; V_0)$ is the main criterion. As oracle F_0 is empty, adversaries have to guess the value of bit b generated by Θ_0 without any oracle. Hence the advantage of any adversary against δ is 0.

- $\sigma = (\Theta; f, f'; V)$ is the criterion related to verifier V. A symmetric key is generated and can be revealed. To win his challenge, an adversary has to forge a fresh encryption using the challenge key without asking for its revelation. As the f' part cannot be called by the adversary without losing, this criterion is equivalent to SYM-CPA/UNF. Therefore, the advantage of any adversary against σ is negligible.

The final step of this proof is to prove that the advantage of any adversary against β is negligible. Then the advantage of adversary $\mathcal{A}$ is negligible and it is possible to conclude that the encryption scheme $\mathcal{SE}$ is safe for P-AC-PAT-SYM-CPA.

Criterion β is criterion SYM-CPA/IND except that the key can be revealed. Let $\mathcal{B}$ be an adversary against β, adversary $\mathcal{B}'$ uses $\mathcal{B}$ to play against SYM-CPA/IND. This adversary executes $\mathcal{B}$ with his oracle for the left-right encryption part. In case $\mathcal{B}$ calls his second oracle, $\mathcal{B}'$ returns 1 for his challenge (returning a random bit would also work).

Adversary $\mathcal{B}'(\eta)/\mathcal{O}$:
$d := \mathcal{B}(\eta)/\mathcal{O}, \lambda q.$ **return** 1
return d

Then let E be the event where $\mathcal{B}$ asks for revelation of the key. The idea there is that when E occurs, $\mathcal{B}$ cannot make any request to his first oracle. Hence his has to guess bit b without any information on it, thus his advantage in this condition is 0. S denotes criterion SYM-CPA/IND.

$$\begin{aligned}
\mathbf{Adv}^S_{\mathcal{B}'}(\eta) &= 2Pr[G^S_{\mathcal{B}'}(\eta) = true] - 1 \\
&= 2Pr[G^S_{\mathcal{B}'}(\eta) = true|E]Pr[E] + 2Pr[G^S_{\mathcal{B}'}(\eta) = true|\neg E]Pr[\neg E] - 1 \\
&= Pr[E] + 2Pr[G^S_{\mathcal{B}}(\eta) = true|\neg E]Pr[\neg E] - 1 \\
&= 2Pr[G^S_{\mathcal{B}}(\eta) = true|E]Pr[E] + +2Pr[G^S_{\mathcal{B}}(\eta) = true|\neg E]Pr[\neg E] - 1 \\
&= \mathbf{Adv}^\beta_{\mathcal{B}}(\eta)
\end{aligned}$$

As adversary $\mathcal{B}'$ plays against SYM-CPA/IND, his advantage is negligible. Hence the advantage of $\mathcal{B}$ is also negligible. The encryption scheme is safe for β and for P-AC-PAT-SYM-CPA. ■

8.3 One Step Further: Dynamic Criteria

The first part of this chapter described the case of criteria where the adversary has access to a polynomial number of challenges (and so to a polynomial number of oracles). A natural idea comes from the fact that adversaries are polynomially bounded. Consequently, even if an adversary has access to a truly unbounded number of challenges and oracles, he is only able to use a polynomial number of them. Thus this section introduces dynamic criteria: let us exemplify this on an extension of IND-CPA to an unbounded number of challenges, this extension is denoted by ∞-IND-CPA.

Let $\mathcal{AE}$ be an asymmetric encryption scheme composed of a key generation algorithm $\mathcal{KG}$, an encryption algorithm $\mathcal{E}$ and a decryption algorithm $\mathcal{D}$. In the case of ∞-IND-CPA, the adversary can ask for dynamic creation of new keys. For this reason, he has access to a *new challenge* oracle ν. This oracle does not use any argument. It generates a new key-pair using $\mathcal{KG}$ and returns an index i. This index is used to specify which key has to be used when querying the oracles. A random bit b is also generated at the start of the game. Then the adversary has access to an oracle F. This oracle takes as argument an index i and a pair of bit-strings composed of bs_0 and bs_1. The oracle returns the encryption of bit-string bs_b using the public key whose index is i: $\mathcal{E}(bs_b, pk_i)$. The adversary has to guess the value of bit b in order to win his challenge.

8.3.1 Definition

A criterion is said to be dynamic when the adversary can ask for the generation of a polynomial number of challenges. Obviously, each new challenge is related to a new oracle, that we only allow to depend on the challenges of which the adversary has asked for the generation previously. Hence, cycles are avoided in the construction of the criterion itself.

A dynamic criterion γ is composed of five elements. These elements are the same as the ones used for iterated criteria. The difference lies in the way these elements are used. These five elements are:

1. An initial challenge generator Θ_0;
2. A single challenge generator Θ;
3. An initial oracle F_0;
4. A single oracle F;
5. And a verifier V_0.

In this section, we suppose that the verifier V_0 only depends on the challenge generated by Θ_0.

Let $\mathcal{A}$ be an adversary. The semantics of the game involving $\mathcal{A}$ against dynamic criterion γ is simple. The challenge generator Θ_0 is used to compute the initial challenge θ_0. The adversary is executed, he has access to three oracles:

- The initial oracle F_0 that uses θ_0 which is the initial challenge generated by Θ_0 (for most criteria, this oracle is empty).
- The new challenge oracle ν: this oracle generates a new challenge θ using Θ. It does not take any argument but it returns the index i used to store θ. Let i denote a global index that has been initialized to 0. The pseudo-code for oracle ν is:

  Oracle $\nu()$:
  $i := i + 1$
  $\theta_i := \Theta(\eta)$
  return i

- A meta-oracle μ. This oracle takes two arguments: an index j and the request itself denoted by q. This oracle checks that challenges have been generated up to index j. If this is not the case, $\perp$ is returned. Else it uses the single oracle F on q in order to answer. The corresponding pseudo-code is:

  Oracle $\mu(j, q)$:
  if $i < j$ **then return** $\perp$
  return $F(q, \theta_0, ..., \theta_j)$

  The single oracle F has access to all the previously generated challenges including θ_0. This allows us to model criteria using patterns while avoiding cycle problems.

The experiment involving the adversary $\mathcal{A}$ against dynamic criterion γ can be represented by the following game:

Game $\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta)$:
$\theta_0 := \Theta_0(\eta)$
$d := \mathcal{A}/\lambda q.F_0(q, \theta_0)$
$\lambda().\nu()$
$\lambda(j, q).\mu(j, q)$
return $V_0(d, \theta_0)$

The notions of advantage and $PrRand$ are unchanged: $PrRand$ is the best probability to win that an adversary $\mathcal{A}$ can get without using the oracles. Hence for a criterion γ, $PrRand$ only depends on the initial generator Θ_0 and the verifier V_0. The advantage of an adversary $\mathcal{A}$ against dynamic criterion γ is still defined by:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 2\big(Pr[\mathbf{G}^{\gamma}_{\mathcal{A}}(\eta) = true] - PrRand^{\gamma}(\eta)\big)$$

Dynamic criteria are illustrated on a simple example in section 8.3.3. Beforehand we extend the partition theorem in order to deal with dynamic criteria.

8.3.2 Partition Theorem for Dynamic Criteria

The polynomial partition theorem can easily be adapted to the dynamic case. The main argument is that the adversary has a polynomial bound P on his execution time. Hence the adversary is not able to create more than $P(\eta)$ challenges. Playing against the dynamic criterion is just like playing against the polynomial criterion where $P(\eta)$ challenges are allowed.

Theorem 8.3 (Simplified Partition Theorem, Dynamic Case) *Let P be a positive polynomial. Let γ be the dynamic criterion $(\Theta_0; \Theta; F_0; F; V_0)$.*

We assume the following hypotheses :

1. *V_0 only depends on the challenge generated by Θ_0, denoted by θ_0. This hypothesis is always met as it occurs in the definition of dynamic criteria.*
2. *There exist some functions f, f' and g such that for any i the oracle F consists in both :*

$$\lambda s \cdot f(g(s, [\theta_0, \ldots, \theta_{i-1}]), \theta_i) \text{ and } \lambda s \cdot f'(s, \theta_i)$$

Then, for any adversary $\mathcal{A}$ (whose execution time is bounded by polynomial P) against criterion δ, there exist two adversaries $\mathcal{B}$ and $\mathcal{C}$, such that:

$$\forall \eta, \boldsymbol{Adv}^{\delta}_{\mathcal{A}}(\eta) = 2P(\eta)\boldsymbol{Adv}^{\beta}_{\mathcal{B}}(\eta) + \boldsymbol{Adv}^{\delta}_{\mathcal{C}}(\eta)$$

where $\beta = (\Theta, b; f \circ LR^b; v_b)$ is indeed an indistinguishability criterion and $\delta = (\Theta_0; F_0; V_0)$ is our "main" criterion.

Proof: The proof is immediate as a consequence of theorem 8.1. Indeed $\mathcal{A}$ can only access $P(\eta)$ different challenges. Hence the advantage of an adversary $\mathcal{A}$ against γ can easily be transformed into the advantage of an adversary $\mathcal{A}'$ (derived from $\mathcal{A}$) against γ' (which is the same criterion as γ with P challenges). ■

8.3.3 Applications

Dynamic criteria have immediate applications in extending previous criteria such as P-AC-IND-CPA. This defines a criterion ∞-AC-IND-CPA where the number of challenge keys is adaptively chosen by the adversary. This criterion is formally defined as $\delta = (\Theta; \kappa; F; F_{\_}; V)$ where: Θ randomly generates a bit b, κ generates a pair of keys using $\mathcal{KG}$, F is the empty oracle, F_k contains all the previous oracles related to the kth key, $V(s, \theta)$ returns true iff s is equal to b.

We consider the case of an asymmetric encryption scheme $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$. For each challenge key (pk, sk), the adversary has access to three oracles:

1. The public key oracle which returns pk.
2. The left-right encryption oracle which is given a pair of patterns (pat_0, pat_1) and outputs the encryption of the concretization of pat_b using key pk.
3. The revelation oracle which returns sk. This oracle can only be called if the left-right oracle has not been called before. After that, the left-right oracle cannot be accessed anymore.

Proposition 8.5 *An encryption scheme is secure against ∞-AC-IND-CPA iff it is secure against IND-CPA.*

Proof: This proof can be done by adapting the proof of proposition 8.1. ■

8.4 Computational Soundness of Formal Methods

In this section, we illustrate how polynomial and dynamic criteria can be used to prove computational soundness of symbolic analysis. In a first example, we consider adaptive security and extend previously existing results [MP05]. The adaptive security case is an extension of Abadi-Rogaway's logic where there adversary can have an adaptive behavior. The second example focuses on protocols in the active case as presented earlier in this document. These two results are not comparable as the adaptive setting is not a sub-case of the active adversary model.

8.4.1 Adaptive Security of Formal Encryption

We have stated before that P-AC-PAT-SYM-CPA and SYM-CPA are equivalent. However as we do not consider the active setting but the adaptive one, the unforgeability part of SYM-CPA is not necessary to prove computational soundness. Thus we introduce a new criterion P-AC-PAT-IND-CPA for symmetric encryption scheme. The challenge generator and the oracle are the same as in P-AC-PAT-SYM-CPA however the only way to win is to find the value of the challenge bit b, hence there is only a single verifier that depends on b. A simple application of theorem 8.1 gives the following result.

Proposition 8.6 *Let P be a polynomial. A symmetric encryption scheme $\mathcal{SE}$ is secure against P-AC-PAT-IND-CPA iff it is secure against IND-CPA.*

Proof: This proof can be done by adapting the proof of proposition 8.1. ∎

Using this we prove an extended version of the soundness theorem given in [MP05] which is itself an extension of the well-known result by Abadi and Rogaway [AR00]. In this section, symmetric encryption is the only considered cryptographic primitive.

Symbolic Equivalence We use the same notion of equivalence as the one given in section 7.3.2 which is borrowed from [AR00] and [MP05]. However we remove the part concerning modular exponentiation, thus the pattern of a message m (which characterizes the information that is accessible by an adversary from m) is defined by:

$$\begin{aligned}
pattern(\langle m_1, m_2\rangle) &= \langle pattern(m_1), pattern(m_2)\rangle \\
pattern(\{m'\}_K) &= \{pattern(m')\}_K && \text{if } m \vdash K \\
pattern(\{m'\}_K) &= \{\square\}_K && \text{if } m \nvdash K \\
pattern(N) &= N \\
pattern(K) &= K
\end{aligned}$$

Where symbol $\square$ still represents a cipher-text that the adversary cannot decrypt. We say that two messages are *equivalent* if they have the same pattern.

$$m \equiv n \text{ if and only if } pattern(m) = pattern(n)$$

Then two messages m and n are *equivalent up to renaming*, $m \cong n$, if there exists σ_1 a permutation of **Keys** and σ_2 a permutation of **Nonces** such that m and $n\sigma_1\sigma_2$ are equivalent.

A consequence of proposition 7.8 is computational soundness of equivalence up to renaming: if two acyclic messages are equivalent up to renaming, then their computational implementations are indistinguishable (no polynomial time adversary can distinguish them with non-negligible probability).

Proposition 8.7 *Let m and n be two acyclic messages, such that $m \cong n$. Let $\mathcal{SE}$ be a symmetric encryption scheme that is IND-CPA secure, then $[\![m]\!]_{\mathcal{SE}} \approx [\![n]\!]_{\mathcal{SE}}$.*

Adaptive security is an extension of this soundness result to the case where the adversary adaptively choses messages m and n. The main difficulty is that the message chosen by the adversary may use a polynomial (in η) number of keys.

The Adaptive Security Problem The security game described here is an extension of the one proposed in [AR00] to the case of an adaptive attacker: a challenge bit b is generated and the adversary tries to guess the value of bit b. For this purpose, he has access to an oracle called the *left-right concretization oracle*. Arguments of this oracle are pairs of equivalent (symbolic) messages m_0, m_1. The oracle returns a possible concretization of m_b.

However, besides acyclicity, there are restrictions on the messages that the adversary is allowed to call the oracle with. Let us consider an adversary that first submits the pair $\{N_0\}_k, \{N_1\}_k$ to his left-right concretization oracle. Then the adversary receives a concretization of $\{N_b\}_k$. Then if the adversary submits pair k, k and pair N_0, N_0 to the left-right concretization oracle, he receives the computational values of k, N_0. Using the computational value of k, the adversary can decrypt the implementation of $\{N_b\}_k$ and compare the result with the value of N_0. The adversary is able to guess the value of bit b with high probability. Pairs of messages submitted by the adversary to his oracle are equivalent: $\{N_0\}_k \equiv \{N_1\}_k$, $k \equiv k$ and $N_0 \equiv N_0$, but the concatenations of these messages are note equivalent:

$$\langle \{N_0\}_k, k, N_0 \rangle \not\equiv \langle \{N_1\}_k, k, N_0 \rangle$$

Thus if the adversary submits the pair $\{N_0\}_k, \{N_1\}_k$ to his oracle, he is not allowed to ask for key k after that because knowing key k, messages $\{N_0\}_k$ and $\{N_1\}_k$ are not equivalent anymore. The restriction is that the concatenations of messages asked by the adversary to his left-right concretization oracle have to be equivalent.

It is important to remark that arguments of the left-right concretization oracle are *symbolic* messages. The adversary does not have to know the computational values of these messages to ask them, he just submits symbolic terms, for example the adversary can get the computational value of key k by querying his left-right concretization oracle with pair k, k. Whatever the value of the challenge bit is, the adversary receives the value of k. A similar game was described in [MP05] where equivalence up to renaming is used. We stick to equivalence without renaming in order to preserve simplicity.

Let $\mathcal{SE}$ denote a symmetric encryption scheme. Formally, the game that involves adversary $\mathcal{A}$ is described by the following algorithm. The integer j is used to store the number of pairs of messages that were asked during previous queries (the queries themselves are stored in two arrays of symbolic messages M_0 and M_1). The computational substitution θ contains the values for the different keys and nonces that appears in messages submitted to the left-right oracle. The *update* function takes as argument two messages and updates θ by randomly sampling nonces and keys that were not previously in θ. Note that the update function depends on the nonce generation algorithm and on the key generation algorithm from $\mathcal{SE}$ and that the *concr* function depends on the encryption algorithm from $\mathcal{SE}$.

Game $\mathrm{Adpt}_{\mathcal{A}}(\eta)$:
$b \xleftarrow{R} [0,1]$
$j := 0$
$\theta := []$
$d := \mathcal{A}(\eta)/\lambda\langle m_0, m_1\rangle.M_0[j] := m_0$
$M_1[j] := m_1$
if not $\langle M_0[0], ..., M_0[j]\rangle \equiv \langle M_1[0], ..., M_1[j]\rangle$ **then return** $\perp$
else $j := j + 1$; $\theta := update(\theta, m_0, m_1)$
return $concr(m_b, \theta)$
return $b = d$

It is possible to describe this game in terms of security criteria. However this is not done here as keys and nonces are dynamically generated (and there can be a polynomial quantity of such creations).

Definition 8.1 *A symmetric encryption scheme $\mathcal{SE}$ is said to be secure against* adaptive symbolic message attacks *if the advantage of any adversary against game Adpt is negligible.*

The advantage of an adversary $\mathcal{A}$ against game Adpt is defined by:

$$\mathbf{Adv}_{\mathcal{A}}^{Adpt}(\eta) = 2Pr[Adpt_{\mathcal{A}}(\eta) = true] - 1$$

Soundness of Adaptive Security As usual, key cycles are problematic. Hence we add the hypothesis that message submitted by the adversary cannot contain encryption cycles. Another hypothesis is made in [MP05] in order to prove soundness. This requirement states that keys cannot be sent after being used. The sequence of messages $\{0\}_k$ followed by $\{k\}_{k'}$ is invalid because key k is used in the first message (to encrypt 0) and is sent in the second message (protected by k'). Although this is fair, it is possible to weaken this requirement: a key can be sent after being used if it was not used to protect another key. Thus the sequence of messages $\{0\}_k$ followed by $\{k\}_{k'}$ is valid but $\{k''\}_k$ followed by $\{k\}_{k'}$ is not valid as key k is used to encrypt another key.

Let us formalize these two requirements through the following definition. First we suppose without loss of generality that symmetric keys are denoted by k_i where i is an integer.

Definition 8.2 *A message m is said to be* cycle free *if for any sub-message $\{m'\}_{k_j}$ of m only keys k_i with $i > j$ can appear in m'.*

A sequence of message m_i is said to be valid *if each m_i is cycle-free and if key k_u is used (at a key position) in m_i to encrypt another key then k_u cannot be sent in messages m_j for $j \geq i$.*

It is now possible to state our soundness theorem. This theorem proves that if an encryption scheme is secure (for an IND-CPA like notion) then any adversary has a negligible advantage in the Adpt game: safety in the symbolic model (represented by the symbolic equivalence relation) implies safety in the computational model (represented by an indistinguishability game). We suppose that the encryption scheme used here hides the length of plain-texts.

This theorem constitutes the main result of [MP05], however we allow some form of adaptive corruption as keys can be used then sent. Although we do not tackle fully adaptive corruptions, our increment is linked to the selective decommitment problem and proposes an approach for solving this open problem [DNRS99].

Theorem 8.4 *Let $\mathcal{SE}$ be a symmetric encryption scheme that is secure for IND-CPA, then $\mathcal{SE}$ is secure against adaptive symbolic message attacks.*

Proof: Let $\mathcal{SE}$ be a symmetric encryption scheme that is secure for IND-CPA, Let $\mathcal{A}$ be an adversary against adaptive symbolic message attacks. Let P be the polynomial bound on the execution of $\mathcal{A}$, hence $\mathcal{A}$ can use at most $P(\eta)$ different symmetric keys. Using $\mathcal{A}$, it is possible to build an adversary $\mathcal{B}$ against P-AC-PAT-IND-CPA which has a comparable advantage. Adversary $\mathcal{B}$ uses his challenge keys to answer queries that $\mathcal{A}$ makes to his oracle. For this purpose $\mathcal{B}$ randomly generates the necessary nonces. The left-right oracle related to key i is denoted by $\mathcal{O}^i_{LR}$, the unary encryption oracle is denoted $\mathcal{O}^i_E$ and the associated revelation oracle is denoted by $\mathcal{O}^i_r$.

Adversary $\mathcal{B}(\eta)/\mathcal{O}^1_{LR}, \mathcal{O}^1_E, \mathcal{O}^1_r ..., \mathcal{O}^{P(\eta)}_{LR}, \mathcal{O}^{P(\eta)}_E, \mathcal{O}^{P(\eta)}_r$:
$j := 0$
$\theta := []$
$d := \mathcal{A}(\eta)/\lambda\langle m_0, m_1\rangle.M_0[j] := m_0$
$M_1[j] := m_1$
if not $M_0[0], ..., M_0[j] \equiv M_1[0], ..., M_1[j]$ **then return** $\perp$
else $j := j + 1$; $\theta := update'(\theta, m_0, m_1)$
return $concr'(m_0, m_1, \theta)$
return d

This adversary uses two new functions $update'$ and $concr'$. The computational substitution θ is only used to store values for nonces. Key values are not stored as they are part of $\mathcal{B}$'s challenge. Hence the $update'$ function only generates the necessary values for new atoms from m_0 and m_1. The $concr'$ function also has to be adapted in order to use the different left-right oracles in order to build a possible concretization of m_b. Note that the $concr'$ algorithm also has to store the keys for which it had to call the revelation oracle $\mathcal{O}_r$. We distinguish two versions of the $concr'$ algorithm. The first one detailed below works on pairs of messages. This version is used by adversary $\mathcal{B}$. If the adversary asks for a key, the revelation oracle is used to get the computational value of the related challenge key. If the adversary asks for a nonce, its value can be found in θ.

Algorithm $concr'(m_0, m_1, \theta)$:
  match m_0, m_1 **with**
  | k_i, k_i → **return** $\mathcal{O}_r(i)$
  | N, N → **return** $a\theta$
  | $\langle n_0, n_0' \rangle, \langle n_1, n_1' \rangle$ → **return** $concr'(n_0, n_1, \theta).concr'(n_0', n_1', \theta)$
  | $\{n_0\}_{k_i}, \{n_1\}_{k_i}$ → **if** k_i has been revealed **then return** $\mathcal{E}\big(concr'(n_0, n_1, \theta), \mathcal{O}_r(i)\big)$
    else $pat_0 := concr'(n_0, \theta)$ Partial Concretization
    $pat_1 := concr'(n_1, \theta)$ Partial Concretization
    return $\mathcal{O}^i_{LR}(pat_0, pat_1)$

This algorithm uses another version of the $concr'$ function that only uses a single message and returns a pattern. This version generates a pattern instead of a bit-string. Symmetric keys are replaced by requests for such keys and nonces are replaced using their value from θ.

We now want to prove that the game involving $\mathcal{A}$ against adaptive symbolic message is similar to the game involving $\mathcal{B}$ against P-AC-PAT-IND-CPA. For this purpose, we relate the behavior of $concr'$ to the behavior of $concr$. First let us consider the cases where $concr'$ fails:

1. The revelation oracle $\mathcal{O}_r(i)$ can refuse to reveal the key value. This means that oracle $\mathcal{O}^i_{LR}$ has already been called, hence key k_i is sent but it has been used as a key before either to encrypt another key or on two different patterns. Our hypotheses prevent this situation from happening.

2. The left-right encryption oracle can return $\perp$. This situation corresponds to encryption cycles and cannot occur due to our hypotheses.

3. The pattern matching of the first version of $concr'$ is not exhaustive. However, the $concr'$ function is only used if the following condition holds.

$$M_0[0], ..., M_0[j] \equiv M_1[0], ..., M_1[j]$$

   Revealed keys correspond exactly to keys that are deducible by the adversary. A quick induction on the structure of the proof of $M_0[j] \equiv M_1[j]$ allows us to deduce that the pattern matching presented here cannot fail.

Now if we consider that $concr'$ outputs a bit-string, then it is easy to see that the distribution created by applying $concr'$ in the case where the challenge bit b equals 1 is the same as the distribution created by $concr$ to message m_0. Formally, if the challenge bit b equals 1 and the challenge keys corresponds to the keys from θ,

$$concr'(m_0, m_1, \theta) = concr(m_0, \theta)$$

When considering message m_1 in the case where b equals 1, we have the same equality:

$$concr'(m_0, m_1, \theta) = concr(m_1, \theta)$$

The two functions $concr'$ and $concr$ behave in similar ways. Thus the game involving $\mathcal{A}$ against adaptive symbolic message is similar to the game involving $\mathcal{B}$ against P-AC-PAT-IND-CPA.

We finally get that the advantages of $\mathcal{B}$ and $\mathcal{A}$ are equal. Symmetric encryption scheme $\mathcal{SE}$ is secure against IND-CPA. Therefore by using proposition 8.6, we get that $\mathcal{SE}$ is secure against P-AC-PAT-IND-CPA. Hence the advantage of $\mathcal{B}$ is negligible and the advantage of $\mathcal{A}$ is also negligible. Therefore, $\mathcal{SE}$ is secure against adaptive symbolic message attacks. ■

This theorem proves that if a symmetric encryption scheme is secure, then the equivalence relation correctly abstracts indistinguishability even in the case of an adaptive adversary. This result cannot be easily compared with results of soundness for protocols. Here, the adversary cannot specify some of the symmetric keys or some nonces values whereas this is possible when considering protocols. However the adversary can choose which messages are going to be concretized

whereas in linear protocols, the adversary has no choice. These two approaches are different. The main interest of this approach is that proofs are simpler because the queries of the adversary do not have to be parsed. Moreover examples of usage for the adaptive model include analysis of multicast protocols are presented in [MP05].

A Limitation: Dynamic Encryption Cycles In this document, cycles have always been defined in a static way: when considering pattern criteria like N-PAT-IND-CPA, the order between keys is $k_1 < k_2 < ... < k_N$. Hence k_1 can be used to encrypt keys k_2 to k_N, and each key k_i can be used to encrypt keys k_{i+1} to k_N. An interesting extension would be to check cycles in a dynamic way: the adversary can submit whichever patterns he wants to each of the oracle, the only limitation is that he should not create encryption cycles. For example, the adversary can first ask for $\mathcal{E}(k_1, k_2)$, then for $\mathcal{E}(k_3, k_2)$. At this point, he cannot ask for $\mathcal{E}(k_2, k_3)$ as this would create a cycle.

Let us formalize this through a new criterion N-dPAT-IND-CPA. This criterion γ_N is defined as the triple $(\Theta; F; V)$. We consider an asymmetric encryption scheme $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$.

- Θ generates N key pairs (pk_1, sk_1) to (pk_N, sk_N) using $\mathcal{KG}$. It also generates the challenge bit b.
- F gives access to two oracles for each key pk_i: the public key oracle which returns pk_i and the left-right encryption oracle which given a pair of patterns $\langle pat_0, pat_1 \rangle$ concretizes pat_b using the necessary keys and outputs the encryption of the result using pk_i.

  F also stores the dependence created by the adversary: if sk_j is encrypted by pk_i, F stores that $i < j$. If a request can create a cycle for $<$, then $\mathcal{A}$ outputs an error $\perp$.
- Finally, the verifier checks that the adversary correctly guessed the value of the challenge bit b.

This new criterion is an extension of N-dPAT-IND-CPA. It is clear that if $\mathcal{AE}$ is secure against N-dPAT-IND-CPA, then it is also secure against IND-CPA but the converse is more complicated. In order to study the converse, we separate two different cases, in the first one, N is independent of η, in the second one, with a slight abuse of notation, N is a polynomial in η.

If N is independent of η, then it is possible to prove that security against IND-CPA implies security against N-dPAT-IND-CPA. For this purpose, we know that security against IND-CPA implies security against N-PAT-IND-CPA. Then let $\mathcal{A}$ be an adversary against N-dPAT-IND-CPA, we build an adversary $\mathcal{B}$ against N-PAT-IND-CPA which use $\mathcal{A}$ as a subroutine. The idea is that $\mathcal{B}$ tries to guess the order that $\mathcal{A}$ will use between keys. As the number of possible orders, $N!$ is independent of η, this strategy succeeds with non-negligible probability. Formally, $\mathcal{B}$ generates a permutation σ of $[1, N]$. Each time $\mathcal{A}$ makes a request, references to key pair number i are replaced by references to key pair $i\sigma$. Then with probability greater than $1/N!$, $\mathcal{B}$ perfectly simulates $\mathcal{A}$, in the other case (i.e. if $\mathcal{B}$ is confronted to a cycle), $\mathcal{B}$ always output 1. Hence we get that the advantage of $\mathcal{B}$ satisfies the relation:

$$\mathbf{Adv}_{\mathcal{B}}^{N-PAT}(\eta) \geq \frac{1}{N!}.\mathbf{Adv}_{\mathcal{A}}^{N-dPAT}(\eta)$$

As security against N-PAT-IND-CPA hold, the left member is negligible, so the advantage of $\mathcal{A}$ is also negligible and we get that the encryption scheme is secure against N-dPAT-IND-CPA.

If N is polynomial in η, the former technique cannot apply: the problem is that $N!$ is exponential in η therefore we cannot conclude from that relation. Thus, when the number of challenges is unbounded, this problem seems worse to solve than adaptive corruption. We did not manage to prove any result linking N-dPAT-IND-CPA to IND-CPA in this setting hence we leave this as an open question.

8.4.2 Computational Soundness of Security Protocols

The final objective when introducing polynomial and dynamic criteria was to be able to extend the link between the symbolic and computational models of protocols to the case of an unbounded number of sessions. Even if the secrecy problem is undecidable in the latter case, there exists a wide variety of protocol checkers that work on unbounded sessions (usually by using abstract interpretation). Such verifiers include Hermes [BLP03b], ProVerif [BAF05], Securify [Cor03] or Casrul [Rus03].

In order to consider protocols with an unbounded number of session, we first have to introduce definitions related to such protocol, then we have to give symbolic and computational semantics to these protocols. As the $CMExec$ procedure from chapter 6 takes as argument a bounded number of actions, this procedure has to be rewritten from scratch. The adversary should be able to create new sessions whenever he wants. He should also be able to make precise the interleaving of the different sessions by naming the agent and the session he is referring too. For these reasons, computational semantics in this case become hard to understand. Therefore we do not give here any formal result. Instead, we informally describe how our new results can be used to extend results from 6.2.

Unbounded Number of Session Proofs detailed in section 6.2 cannot be directly extended to consider an unbounded number of sessions. A major problem is that the adversary $\mathcal{B}$ that is built in these proofs starts by randomly selecting a subset of keys CK. This set is equal to the set of keys that are not leaked to the adversary with probability 2^{-N} where N is the total number of keys. In the bounded model, this probability is constant hence it is not negligible. However in the case of an unbounded number of sessions, the probability is $2^{-P(\eta)}$ where P is a polynomial. This probability is negligible and the usual proof scheme cannot apply.

There are different easy ways to handle this problem by adding some restrictions. For example, [MW04c] and [CW05] do not consider emission of secret keys. The corruption model of [CW05] allows the adversary to statically corrupt participants: corruption is done at the beginning of the protocol and it is not possible to later corrupt some more participants. Therefore keys that are secret at the beginning of the protocol stay secret through the whole protocol execution. In [JLM05a] there is no corruption model but keys can be sent. In order to prevent keys from being known by the adversaries, a restrictive hypothesis is done: (secret) keys that are secret at the beginning of the protocol cannot be disclosed to the adversary later.

Using the ∞-AC-PAT-SYM-CPA criterion and other criteria, it is possible to bypass such restrictions. The proof can be achieved with an adaptive corruption model, however we have to keep restrictions from the previous section: the order between keys is fixed at the beginning of the protocol, moreover a key can only be corrupt if it was not used before to encrypt other keys.

Chapter 9

Opacity

Contents

In the symbolic world, the notion of secrecy has been formulated in various ways in the computer security literature. However, two different views of secrecy have been developed over the years by two separate communities. The first one starts from the notion of information flow, describing the knowledge an adversary could gain in terms of properties such as non-deducibility or non-interference. The second view was initiated in Dolev and Yao's work and focused initially on security properties. The idea here is to describe properly the capability of the adversary. Some variants of secrecy appeared, such as strong secrecy, giving more expressivity than the security property but still lacking the expressivity of information flow concepts. Moreover in the computational setting, indistinguishability is used to represent a wide variety of properties like strong secrecy or anonymity. Thus our objective is this chapter is to find an equivalent of indistinguishability for the symbolic world.

Opacity was introduced in order to describe complex security properties in the symbolic setting. It has first been formulated for security protocols [Maz04b]. Then this work was extended in terms of Petri nets [BKR04b, BKR04a, BKMR05]. In this chapter, we first introduce a general notion of opacity for the framework of labelled transition systems. When using opacity we have fine-grained control over the observation capabilities of the players, and we show one way that these capabilities may be encoded. The essential idea is that a predicate is opaque if an observer of the system will never be able to establish the trueness or falsity of this predicate. Moreover, opacity has a natural transcription in the computational setting. An opaque predicate can be directly transformed into

an indistinguishability property. A property is opaque in the computational world if it is not possible for any adversary to distinguish cases where the property is verified from other cases with non-negligible probability. Then it is possible to link the symbolic and computational version of indistinguishability.

This chapter is structured as follows. In the first section, after recalling some basic definitions, we present a generalisation of opacity, and show how this specialises into the three previously defined variants: initial opacity, final opacity and total opacity. We also show how opacity is related to previous work in security. We consider how opacity may describe anonymity and non-interference, and quickly discuss it in the context of security protocols. In section 9.2, we consider the question of opacity checking, and state a general undecidability result for opacity. After restricting ourselves to Petri nets, we give some decidability and undecidability properties. As opacity is undecidable as soon as we consider systems with an infinite number of states, we present an approximation technique which may provide a way of model checking even in such cases. This section also includes two examples. The first, drawn from the commercial world, illustrates how anonymity may be expressed using opacity. The second considers a voting scheme, and shows how the approximation technique might be used. Then section 9.3 relates opacity in the symbolic setting to indistinguishability in the computational setting. These results are applied to obtain the first proof of security for the Chaum voting scheme in the computational world.

9.1 Introducing Symbolic Opacity

9.1.1 Basic Definitions

The set of finite sequences over a set A is denoted by A^*, and the empty sequence by ϵ. The length of a finite sequence λ is denoted by $len(\lambda)$, and its projection onto a set $B \subseteq A$ by $\lambda|_B$.

Definition 9.1 *A labelled transition system (LTS) is a tuple $\Pi = (S, L, \Delta, S_0)$, where S is the (potentially infinite) set of states, L is the (potentially infinite) set of labels, $\Delta \subseteq S \times L \times S$ is the transition relation, and S_0 is the nonempty (finite) set of initial states.*
A run of Π is a pair (s_0, λ), where $s_0 \in S_0$ and $\lambda = l_1 \dots l_n$ is a finite sequence of labels such that there are states $s_1, \dots, s_n$ satisfying $(s_{i-1}, l_i, s_i) \in \Delta$, for $i = 1, \dots, n$. We also denote the state s_n by $s_0 \oplus \lambda$, and call it reachable from s.
The set of all runs is denoted by $run(\Pi)$, and the language generated by Π is defined as $\mathcal{L}(\Pi) = \{\lambda \mid \exists s_0 \in S_0 : (s_0, \lambda) \in run(\Pi)\}$.

A LTS $\Pi = (S, L, \Delta, S_0)$ is *deterministic* if for any transitions $(s, l, s'), (s, l, s'') \in \Delta$, it is the case that $s' = s''$. Otherwise Π is said to be *non-deterministic*. A classical property states that it is possible to *determinize* a non-deterministic LTS:

Proposition 9.1 *Let Π be a non-deterministic LTS, then there exists a deterministic LTS Π' such that $run(\Pi) = run(\Pi')$.*

Let $\Pi = (S, L, \Delta, S_0)$ be an LTS fixed for the rest of this section, and Θ be a set of elements called *observables*. We now aim at modelling the different capabilities for observing the system modelled by Π. First, we introduce a general observation function and then, specialize it to reflect limited information about runs available to an observer. An observation function outputs a word, i.e. a list of observables (which can be seen as letters).

Definition 9.2 *Any function $obs : run(\Pi) \to \Theta^*$ is an observation function. It is called label-based and: static / dynamic / orwellian / m-orwellian ($m \geq 1$) if respectively the following hold (below $\lambda = l_1 \dots l_n$):*

- *static: there is a mapping $obs' : L \to \Theta \cup \{\epsilon\}$ such that for every run (s, λ) of Π, $obs(s, \lambda) = obs'(l_1) \dots obs'(l_n)$.*

- *dynamic: there is a mapping* $obs' : L \times L^* \to \Theta \cup \{\epsilon\}$ *such that for every run* (s, λ) *of* Π, $obs(s, \lambda) = obs'(l_1, \epsilon)obs'(l_2, l_1) \dots obs'(l_n, l_1 \dots l_{n-1})$.
- *orwellian: there is a mapping* $obs' : L \times L^* \to \Theta \cup \{\epsilon\}$ *such that for every run* (s, λ) *of* Π, $obs(s, \lambda) = obs'(l_1, \lambda) \dots obs'(l_n, \lambda)$.
- *m-orwellian: there is a mapping* $obs' : L \times L^* \to \Theta \cup \{\epsilon\}$ *such that for every run* (s, λ) *of* Π, $obs(s, \lambda) = obs'(l_1, \kappa_1) \dots obs'(l_n, \kappa_n)$, *where for* $i = 1, \dots, n$, $\kappa_i = l_{max\{1,i-m+1\}} l_{\max\{1,i-m+1\}+1} \cdots l_{\min\{n,i+m-1\}}$.

In each of the above four cases, we often use $obs(\lambda)$ *to denote* $obs(s, \lambda)$.

Note that allowing *obs′* to return ϵ allows one to model invisible actions. The different kinds of observable functions reflect different computational power of the observers. Static functions correspond to an observer which always interprets the same executed label in the same way. Dynamic functions correspond to an observer which has potentially infinite memory to store labels, but can only use knowledge of previous labels to interpret a label. Orwellian functions correspond to an observer which has potentially infinite memory to store labels, and can use knowledge (either subsequent or previous) of other labels to (re-)interpret a label. m-orwellian functions are a restricted version of the last class where the observer can store only a bounded number of labels. Static functions are nothing but 1-orwellian ones; static functions are also a special case of dynamic functions; and both dynamic and m-orwellian are a special case of orwellian functions.

It is possible to define state-based observation functions. For example, a state-based static observation function *obs* is one for which there is $obs' : S \to \Theta \cup \{\epsilon\}$ such that for every run $(s, l_1 \dots l_1)$, we have $obs(s, l_1 \dots l_n) = obs'(s)obs'(s \oplus l_1) \dots obs'(s \oplus l_1 \dots l_n)$.

Let us consider an observation function *obs*. We are interested in whether an observer can establish a property ϕ (a predicate over system states and traces) for some run having only access to the result of the observation function. We identify ϕ with its characteristic set: the set of runs for which it holds and assume that ϕ is decidable.

Now, given an observed execution of the system, we would want to find out whether the fact that the underlying run belongs to ϕ can be deduced by the observer (note that we are not interested in establishing whether the underlying run does not belong to ϕ; to do this, we would rather consider the property $\overline{\phi} = run(\Pi) \setminus \phi$).

What it means to deduce a property can mean different things depending on what is relevant or important from the point of view of real application. Below, we give a general formalisation of opacity and then specialise it in three different ways.

Definition 9.3 *A predicate* ϕ *over* $run(\Pi)$ *is opaque w.r.t. the observation function obs if, for every run* $(s, \lambda) \in \phi$, *there is a run* $(s', \lambda') \notin \phi$ *such that* $obs(s, \lambda) = obs(s', \lambda')$. *Moreover,* ϕ *is called: initial-opaque / final-opaque / total-opaque if respectively the following hold:*

- *there is a predicate* ϕ' *over* S_0 *such that for every run* (s, λ) *of* Π, *we have* $\phi(s, \lambda) = \phi'(s)$.
- *there is a predicate* ϕ' *over* S *such that for every run* (s, λ) *of* Π, *we have* $\phi(s, \lambda) = \phi'(s \oplus \lambda)$.
- *there is a predicate* ϕ' *over* S^* *such that for every run* $(s, l_1 \dots l_n)$ *of* Π, *we have* $\phi(s, l_1 \dots l_n) = \phi'(s, s \oplus l_1, \dots, s \oplus l_1 \dots l_n)$.

In the first of above three cases, we often write $s \in \phi$ *whenever* $(s, \lambda) \in \phi$.

Initial-opacity has been illustrated by the dining cryptographers example (in [BKR04b] with two cryptographers and [BKR04a] with three). It would appear that it is suited to modelling situations in which initialization information such as crypto keys, etc., needs to be kept secret. More generally, situations in which confidential information can be modelled in terms of initially resolved non-determinism can be captured in this way. Final-opacity models situations where the final result of a computation needs to be secret. Total-opacity is a generalisation of the other two properties asking not only the result of the computation and its parameters to be secret but also the states visited during computation.

Proposition 9.2 *Let ϕ and ϕ' be two predicates over run(Π). If ϕ is opaque w.r.t. an observation function obs and $\phi' \Rightarrow \phi$, then ϕ' is opaque w.r.t. obs.*

Proof: Follows directly from definitions. ∎

The goal of this section is to show how our notion of opacity relates to other concepts commonly used in the formal security community. We compare opacity to forms of anonymity and non-interference.

9.1.2 Anonymity

Anonymity is concerned with the preservation of secrecy of identity through the obscuring of the actions of this identity. It is a function of the behaviour of the underlying (anonymizing) system, as well as being dependent on capabilities of the observer.

The static, dynamic and orwellian forms of observation function presented in definition 9.2 model three different strengths of observer. We now introduce two observation functions needed to render anonymity in terms of suitable opacity properties.

Let $\Pi = (S, L, \Delta, S_0)$ be an LTS fixed for the rest of this section, and $A = \{a_1, \ldots, a_n\} \subseteq L$ be a set of labels over which anonymity is being considered. Moreover, let $\alpha, \alpha_1, \ldots, \alpha_n \notin L$ be fresh labels.

The first observation function, obs^s_A, is static and defined so that $obs^s_A(\lambda)$ is obtained from λ by replacing each occurrence of a_i by α. The second observation function, obs^d_A, is dynamic and defined thus: let $a_{i_1}, \ldots, a_{i_q}$ $(q \geq 0)$ be all the distinct labels of A appearing within λ listed in the (unique) order in which they appeared for the first time in λ; then $obs(\lambda)$ is obtained from λ by replacing each occurrence of a_{i_j} by α_j. For example,

$$obs^s_{\{a,b\}}(acdba) = \alpha cd\alpha\alpha \quad \text{and} \quad obs^d_{\{a,b\}}(acdba) = \alpha_1 cd\alpha_2\alpha_1.$$

Strong anonymity

In [SS96], a definition of strong anonymity is presented for the process algebra CSP. In our (LTS) context, this definition translates as follows.

Definition 9.4 *Π is strongly anonymous w.r.t. A if $\mathcal{L}(\Pi) = \mathcal{L}(\Pi')$, where Π' is obtained from Π by replacing each transition (s, a_i, s') with n transitions: $(s, a_1, s'), \ldots, (s, a_n, s')$.*

In our framework, we have that

Definition 9.5 *Π is O-anonymous w.r.t. A if, for every sequence $\mu \in A^*$, the predicate ϕ_μ over the runs of Π defined by*

$$\phi_\mu(s, \lambda) = \big(len(\lambda|_A) = len(\mu) \ \wedge \ \lambda|_A \neq \mu\big)$$

is opaque w.r.t. obs^s_A.

We want to ensure that every possible sequence μ (with appropriate length restrictions) of anonymized actions is a possible sequence within the LTS. In definition 9.5 above, the opacity of the predicate ϕ_μ ensures that the sequence μ is a possible history of anonymized actions, because it is the only sequence for which the predicate ϕ_μ is false, and so ϕ_μ can only be opaque if μ is a possible sequence.

Proposition 9.3 *Π is O-anonymous w.r.t. A* iff *it is strongly anonymous w.r.t. A.*

Proof: We first observe that the strong anonymity w.r.t. A is equivalent to

$$\{\lambda' \in L^* \mid \exists \lambda \in \mathcal{L}(\Pi) : obs^s_A(\lambda') = obs^s_A(\lambda)\} \ \subseteq \ \mathcal{L}(\Pi)\ . \tag{9.1}$$

We show that Π is O-anonymous w.r.t. A iff (9.1) holds.

($\Longrightarrow$) Suppose that $\lambda \in \mathcal{L}(\Pi)$ and $\lambda' \in L^*$ are such that $obs^s_A(\lambda') = obs^s_A(\lambda)$. Clearly, if $\lambda'|_A = \lambda|_A$ then $\lambda' = \lambda \in \mathcal{L}(\Pi)$, so we assume that $\lambda'|_A \neq \lambda|_A$. Then, for some $s \in S_0$, we have that $\phi_\mu(s, \lambda)$ holds, where $\mu = \lambda'|_A$. Hence, by the opacity of ϕ_μ w.r.t. obs^s_A, there is $(s', \lambda'') \in run(\Pi)$ such that $obs^s_A(\lambda'') = obs^s_A(\lambda)$ and $\phi_\mu(s', \lambda'')$ does not hold. Thus $\lambda' = \lambda'' \in \mathcal{L}(\Pi)$. As a result, (9.1) holds.

($\Longleftarrow$) Suppose that $\mu \in A^*$ and $\phi_\mu(s, \lambda)$ holds. Then $len(\lambda|_A) = len(\mu)$ and $\lambda|_A \neq \mu$. Let $\lambda' \in L^*$ be the unique sequence such that $\lambda'|_A = \mu$ and $obs^s_A(\lambda') = obs^s_A(\lambda)$. By (9.1), there is s' such that $(s', \lambda') \in run(\Pi)$. Clearly, $obs^s_A(\lambda') = obs^s_A(\lambda)$ and $\phi_\mu(s', \lambda')$ does not hold. As a result, ϕ_μ is opaque w.r.t. obs^s_A, and so Π is O-anonymous w.r.t. A. ■

Weak anonymity

A natural extension of strong anonymity is *weak anonymity*[1]. This models easily the notion of *pseudo-anonymity*: actions performed by the same party can be correlated, but the identity of the party cannot be determined.

Definition 9.6 *Π is weakly anonymous w.r.t. A if $\pi(\mathcal{L}(\Pi)) \subseteq \mathcal{L}(\Pi)$, for every permutation π over the set A.*

In our framework, we have that

Definition 9.7 *Π is weak-O-anonymous if, for every sequence $\mu \in A^*$, the predicate ϕ_μ over the runs of Π introduced in definition 9.5 is opaque w.r.t. obs^d_A.*

Then these two notions of weak anonymity can be related through the following proposition.

Proposition 9.4 *Π is weak-O-anonymous w.r.t. A* iff *it is weak-anonymous w.r.t. A.*

Proof: We first observe that $obs^d_A(\lambda) = obs^d_A(\lambda')$ *iff* there is a permutation π over the set A such that $\pi(\lambda) = \lambda'$, and so showing weak anonymity w.r.t. A is equivalent to showing that

$$\{\lambda' \in L^* \mid \exists \lambda \in \mathcal{L}(\Pi) : obs^d_A(\lambda') = obs^d_A(\lambda)\} \ \subseteq \ \mathcal{L}(\Pi) \ .$$

The proof then follows similar lines to that of Theorem 9.3, with obs^d_A playing the role of obs^s_A. ■

Other observation functions

Dynamic observation functions can model for example the *downgrading* of a channel. Before the downgrading nothing can be seen, after the downgrading the observer is allowed to see all transmissions on that channel. A suitable formulation would be as follows.

Suppose that A represents the set of all possible messages on a confidential channel, and $\delta \in L \setminus A$ represents an action of downgrading that channel. Then $obs(\lambda)$ is obtained from λ by deleting each occurrence of a_i which precedes (directly or indirectly) an occurrence of δ. In other words, if the downgrading action appears earlier in the run, then the messages on the channel are observed in the clear, otherwise nothing is observed.

Orwellian observation functions can model conditional or escrowed anonymity, where someone can be anonymous when they initially interact with the system, but some time in the future their identity can be revealed, as outlined below.

Suppose that there are n identities Id_i, each identity being capable of performing actions represented by $a_i \in A$. Moreover, $\alpha \notin L$ represents the encrypted observation of any of these actions, and $\rho_i \in L \setminus A$ represents the action of identity Id_i being revealed. Then $obs(\lambda)$ is obtained from λ by replacing each occurrence of a_i by α, provided that ρ_i never occurs within λ.

[1] We believe that this formulation of weak anonymity was originally due to Ryan and Schneider.

9.1.3 Non-Interference

Opacity can be linked to a particular formulation of non-interference. A discussion of non-interference can be found in [FG94b] and [Rya01]. The basic idea is that labels are split into two sets, $High$ and Low. Low labels are visible by anyone, whereas $High$ labels are private. Then, a system is non-interfering if it is not possible for an outside observer to gain any knowledge about the presence of $High$ labels in the original run (the observer only sees Low labels). This notion is in fact a restriction of standard non-interference. It was originally called non-inference in [O'H90], and is called strong non-deterministic non-interference in [FG01].

Definition 9.8 Π *satisfies non-inference if* $\mathcal{L}(\Pi)|_{Low} \subseteq \mathcal{L}(\Pi)$.

In other words, for any run (s, λ) of Π, there exists a run (s', λ') such that λ' is λ with all the labels in $High$ removed.

The notion of non-interference (and in particular non-inference) is close to opacity as stated by the following two properties. First, we show that it is possible to transform certain initial opacity properties into non-inference properties.

Proposition 9.5 *Any initial opacity problem involving a static observation function can be reduced to a non-inference problem.*

Proof: Let $\Pi = (S, L, \Delta, S_0)$ be an LTS, obs defined through obs' (see definition 9.2) be a static observation function, and ϕ defined through ϕ' (see definition 9.3) be an initial opacity predicate. We construct a new LTS $\Pi' = (S', L', \Delta', S'_0)$ such that:

- $S' = S \cup \{s' \mid s \in S_0\}$ where each s' is a fresh state.
- $L' = obs'(L) \cup \{h\}$ where $h \notin obs'(L)$ is a fresh label.
- Δ' is obtained from Δ by replacing each $(s, l, r) \in \Delta$ by $(s, obs'(l), r)$, and adding, for each $s \in S_0$, a new transition (s', h, s).
- $S'_0 = \{s \mid s \in S_0 \setminus \phi\} \cup \{s' \mid s \in S_0 \cap \phi\}$.

We then consider a non-inference problem for Π' with $Low = obs'(L)$ and $High = \{h\}$, and below we show that Π' satisfies non-inference *iff* for Π the opacity property ϕ w.r.t. obs holds. We assume that Π' is deterministic; otherwise we replace it by its deterministic version.

($\Longrightarrow$) Suppose that $(s, \lambda) \in run(\Pi) \cap \phi$. Then $(s', h\ obs(\lambda)) \in run(\Pi')$. Thus, by non-inference of Π', there is $(r, \kappa) \in run(\Pi')$ such that $obs(\lambda) = \kappa$, so $r \in S_0 \setminus \phi$. Hence there is $(r, \mu) \in run(\Pi)$ such that $obs(\mu) = \kappa = obs(\lambda)$. Consequently, the opacity of ϕ w.r.t. obs holds.

($\Longleftarrow$) Suppose that $(r, \kappa) \in run(\Pi')$ and $\kappa|_{Low} \neq \kappa$. Then $\kappa = h\rho$ and $r = s'$, for some ρ and $s \in S_0 \cap \phi$. Hence there is $(s, \lambda) \in run(\Pi)$ such that $obs(\lambda) = \rho$ and, by the opacity of ϕ holding for obs, there is $(r, \psi) \in run(\Pi)$ such that $r \notin \phi$ and $obs(\psi) = obs(\lambda)$. In turn, this means that $(r, obs(\psi)) \in run(\Pi')$. We finally have $\kappa|_{Low} = \rho = obs(\lambda) = obs(\psi)$, and so Π' satisfies non-inference. ∎

A kind of converse result also holds, in the sense that one can transform any non-inference property to a general opacity property.

Proposition 9.6 *Any non-inference problem can be reduced to an opacity problem.*

Proof: Let $(S, High \cup Low, \Delta, S_0)$ be an LTS. We define ϕ as a predicate over $run(\Pi)$ so that $\phi(s, \lambda)$ holds *iff* $\lambda|_{Low} \neq \lambda$. Moreover, obs is defined as a static observation function such that $obs(\lambda) = \lambda|_{Low}$. Below we show that Π satisfies non-inference *iff* for Π the opacity property ϕ w.r.t. obs holds.

($\Longrightarrow$) Suppose that $(s, \lambda) \in run(\Pi) \cap \phi$. Then $\lambda|_{Low} \neq \lambda$ and so, by the non-inference of Π, there is s' such that $(s', \lambda|_{Low}) \in run(\Pi)$. Clearly, $(s', \lambda|_{Low}) \notin \phi$ and $obs(s', \lambda|_{Low}) = obs(s, \lambda)$, since $(\lambda|_{Low})|_{Low} = \lambda|_{Low}$. As a result, the opacity of ϕ w.r.t. obs holds.

($\Longleftarrow$) Suppose that $(s,\lambda) \in run(\Pi)$ and $\lambda{\restriction_{Low}} \neq \lambda$. Then $(s,\lambda) \in run(\Pi) \cap \phi$ and so, by the the opacity property ϕ w.r.t. *obs*, there is $(s',\lambda') \in run(\Pi) \setminus \phi$ such that $obs(\lambda) = obs(\lambda')$. Thus $\lambda'{\restriction_{Low}} = \lambda'$ and $\lambda'{\restriction_{Low}} = \lambda{\restriction_{Low}}$. Hence $\lambda{\restriction_{Low}} \in \mathcal{L}(\Pi)$. As a result, Π satisfies non-inference. ■

Non-interference in general makes a distinction between public (Low) and private ($High$) messages, and any revelation of a high message breaks the non-interference property. We believe that the ability to fine-tune the *obs* function may make opacity better suited to tackling the problem of *partial information flow*, where a message could provide some partial knowledge and it may take a collection of such leakages to move the system into a compromised state.

9.2 Opacity Checking

Opacity is a very general concept and many instantiations of it are undecidable. This is even true when LTSs are finite. We formulate such a property as proposition 9.8 (part 4), but first we state a general non-decidability result.

Proposition 9.7 *Opacity is undecidable.*

Proof: We show that the reachability problem for Turing machines is reducible to (final) opacity. Let TM be a Turing machine and s be one of its (non-initial) states. We construct an instance of the final opacity as follows: Π is given by the operational semantics of TM, the observation function *obs* is constant, and ϕ returns true *iff* the final state of a run is different from s. Since s is reachable in TM *iff* ϕ is final opaque w.r.t. *obs*, opacity is undecidable. ■

It follows from the above proposition that the undecidability of the reachability problem for a class of machines generating LTSs makes opacity undecidable. We therefore restrict ourselves to Petri nets, a rich model of computation in which the reachability problem is still decidable [Rei98]. Furthermore, Petri nets are well-studied structures and there is a wide range of tools and algorithms for their verification.

9.2.1 Petri Nets

We use Petri nets with weighted arcs [Rei98], and give their operational semantics in terms of *transition sequences*.[2] Note that this varies slightly from the one used in [BKR04b] where the *step sequence* semantics allowed multiple transitions to occur simultaneously. Here, transitions are clearly separated.

A (weighted) *net* is a triple $N = (P,T,W)$ such that P and T are disjoint finite sets, and $W : (T \times P) \cup (P \times T) \to \mathbb{N}$. The elements of P and T are respectively the *places* and *transitions*, and W is the *weight function* of N. In diagrams, places are drawn as circles, and transitions as rectangles. If $W(x,y) \geq 1$ for some $(x,y) \in (T \times P) \cup (P \times T)$, then (x,y) is an *arc* leading from x to y. As usual, arcs are annotated with their weight if this is 2 or more. The *pre-* and *post-multiset* of a transition $t \in T$ are multisets of places, $\textsc{pre}_N(t)$ and $\textsc{post}_N(t)$, respectively given by

$$\textsc{pre}_N(t)(p) = W(p,t) \quad \text{and} \quad \textsc{post}_N(t)(p) = W(t,p),$$

for all $p \in P$. A *marking* of a net N is a multiset of places. Following the standard terminology, given a marking M of N and a place $p \in P$, we say that p is marked if $M(p) \geq 1$ and that $M(p)$ is the number of tokens in p. In diagrams, M is represented by drawing in each place p exactly $M(p)$ tokens (black dots). Transitions represent actions which may occur at a given marking and then lead to a new marking. A transition t is *enabled* at a marking M if $M \geq \textsc{pre}_N(t)$. Thus, in order for t to be enabled at M, for each place p, the number of tokens in p under M should be greater than or equal to the total number of tokens that are needed as an input to t, respecting the weights of the input arcs. If t is enabled at M, then it can be *executed* leading to the marking $M' = M - \textsc{pre}_N(t) + \textsc{post}_N(t)$. This means that the execution of t 'consumes' from each place p

[2] It should be stressed that the transitions in the Petri net context correspond to the labels rather than arcs in the LTS framework.

exactly $W(p,t)$ tokens and 'produces' in each place p exactly $W(t,p)$ tokens. If the execution of t leads from M to M' we write $M[t\rangle M'$ and call M' *reachable* in one step from M. A *marked Petri net* $\Sigma = (N, S_0)$ comprises a net $N = (P, T, W)$ and a finite set of initial markings S_0. It generates the LTS $\Pi_\Sigma = (S, T, \Delta, S_0)$ where S is the set of all the markings reachable (in any number of steps) from the markings in S_0, T is the set of labels, and Δ is defined by $(M, t, M') \in \Delta$ if $M[t\rangle M'$. The language of Σ is that of Π_Σ.

In the case of Petri nets, there are still some undecidable opacity problems.

Proposition 9.8 *The following problems are undecidable for Petri nets:*

1. *Initial opacity when considering a static observation function.*
2. *Initial opacity when considering a state-based static observation function.*
3. *Initial opacity when considering an orwellian observation function even in the case of finite LTSs generated by marked nets.*
4. *Opacity when considering a constant observable function even in the case of finite LTSs generated by marked nets.*

Proof: Below we reduce three undecidable problems to suitable variants of opacity, in each case defining a marked Petri net Σ as well as observation function *obs* and opacity predicate ϕ for the runs of Π_Σ. The first two are related to Petri nets: the language inclusion problem [JE96], and the reachable markings inclusion problem [Pet81]. The third one is the Post Correspondence Problem (PCP): we consider a fixed alphabet with more than two symbols. Then given a finite set of pair of words $(a_i, b_i)_i$ over this alphabet it is impossible for an algorithm to answer the following question: is there a finite sequence of integer i_1 to i_n such that the concatenation of words a_{i_1} to a_{i_n} is equal to b_{i_1} to b_{i_n}.

$$a_{i_1}...a_{i_n} = b_{i_1}...b_{i_n}$$

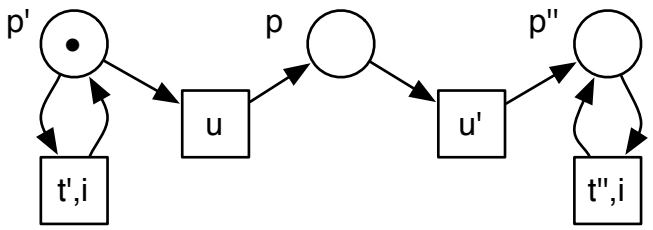
(1) Let $\Sigma_i = (N_i, \{M_i\})$, for $i = 1, 2$, be two marked Petri nets. We first construct their isomorphic copies $\Sigma'_i = (N'_i, \{M'_i\})$, for $i = 1, 2$, in such a way that each transition or place x in Σ_i is renamed to (x, i), and $M'_i = \{(s_1, i), \ldots, (s_{k_i}, i)\}$ where $M_i = \{s_1, \ldots, s_{k_i}\}$. Then:

- $\Sigma = (N, \{M'_1, M'_2\})$ is a marked net such that N is the union of N'_1 an N'_2.
- *obs* is static and given by $obs'(t, i) = t$, for each transition (t, i) in Σ.
- ϕ is true *iff* the first marking of a run is M'_1.

Since the language of Σ_1 is included in that of Σ_2 *iff* ϕ is initial opaque w.r.t. *obs*, part (1) holds.

(2) Let $\Sigma_i = (N_i, \{M_i\})$, for $i = 1, 2$, and Σ be the three marked Petri nets as in the proof of part (1). Then:

- We modify Σ in such a way that each transition (t, i) is replaced by two identically connected copies, (t', i) and (t'', i). We then add to Σ three fresh places, p', p and p'', and two fresh transitions, u and u', in such a way that their arcs are as follows: $W(p', u) = W(u, p) = W(p, u') = W(u', p'') = 1$, $W(p', (t', i)) = W((t', i), p') = 1$ and $W(p'', (t'', i)) = W((t'', i), p'') = 1$, for each transition (t, i) of Σ. Next, to obtain the initial markings, we add one copy of p' to both M'_1 and M'_2.



- *obs* is state-oriented and given by $obs'(M) = \{s_1, \ldots, s_m\}$ for any marking $M = \{p, (s_1, i_1), \ldots, (s_m, i_m)\}$, and $obs'(M) = \epsilon$ otherwise (note that such an observation function allows one to inspect at most one state of a given run).
- ϕ is true *iff* the first marking of a run is $M'_1 + \{p'\}$.

Property ϕ is initial opaque if for any run starting from $M'_1 + \{p'\}$, there is a run starting from $M'_2 + \{p'\}$ that has the same observable trace. There is only a single non empty observation for each run and this observation gives an accessible markings (from Σ_1 if the initial marking was $M'_1 + \{p'\}$ and from Σ'_2 if it was $M'_2 + \{p'\}$). Since the set of reachable markings of Σ_1 is included in that of Σ_2 *iff* ϕ is initial opaque w.r.t. *obs*, part (2) holds.

(3) Let us consider an instance of PCP with (a_i, b_i), for $i = 1, \ldots, n$. Then:

- Σ consists of a net $(\{s, s'\}, \{(a_1, 1), (b_1, 1), \ldots, (a_n, n), (b_n, n)\}, W)$ and the initial markings $S_0 = \{\{s\}, \{s'\}\}$, with the arcs given by

$$W(s, (a_i, i)) = W((a_i, i), s) = W(s', (b_i, i)) = W((b_i, i), s') = 1$$

  for $i = 1, \ldots, n$. Clearly, Π_Σ is finite.
- *obs* is orwellian and depends only on the sequence $\lambda = (x_1, i_1) \ldots (x_m, i_m)$ returning $x_1 \ldots x_m i_1 \ldots i_m$ (note that $x_1 \ldots x_m$ is the concatenation of labels which are words).
- ϕ is true *iff* the first marking of a run is $\{s\}$.

Since the instance of PCP has a solution *iff* ϕ is initial opaque w.r.t. *obs*, part (3) holds.

(4) Let us consider an instance of PCP with (a_i, b_i), for $i = 1, \ldots, n$. Then:

- Σ consists of a net $(\{s\}, \{1, \ldots, n\}, W)$ and the initial marking $S_0 = \{\{s\}\}$, where the arcs are given by $W(s, i) = W(i, s) = 1$ for $i = 1, \ldots, n$. Clearly, Π_Σ is finite.
- *obs* always returns ϵ.
- $\phi(\{s\}, i_1 \ldots i_m)$ is true *iff* $m \geq 1 \Rightarrow a_{i_1} \ldots a_{i_m} \neq b_{i_1} \ldots b_{i_m}$.

Since the instance of PCP has a solution *iff* ϕ is opaque w.r.t. *obs*, part (4) holds. ∎

An analysis of the proof of the last result identifies two sources for the complexity of the opacity problem. The first one is the complexity of the studied property, captured through the definition of ϕ. In particular, the latter may be used to encode undecidable problems and so in practice one should presumably restrict the interest to relatively straightforward versions of opacity, such as the initial opacity. The second source is the complexity of the observation function, and it is presumably reasonable to restrict the interest to some simple classes of observation functions, such as the static observation functions. This should not, however, be considered as a real drawback since the initial opacity combined with an n-orwellian observation function yields an opacity notion which is powerful enough to deal, for example, with bounded security protocols.

What now follows is a crucial result stating that initial opacity with an n-orwellian observation function is decidable provided that the LTS generated by a marked Petri net is finite[3]. In fact, this result could be generalised to any finite LTS.

Proposition 9.9 *In the case of a finite LTS, initial opacity w.r.t. an n-orwellian observation function is decidable.*

Proof: The result was shown in [BKR04b] using regular language inclusion for $n = 1$. Here, we will re-use this result after reducing the case of $n = 2$ to that of $n = 1$ (the proposed reduction can easily be extended to any $n > 2$).

Let $\Pi = (S, L, \Delta, S_0)$ be a finite LTS, for which a 2-orwellian observation function *obs*, and initial opacity predicate ϕ, are given. We define an LTS $\Pi' = (S', L', \Delta', S'_0)$ together with a static observation function obs' and initial opacity predicate ϕ' for the runs of Π', as follows.

[3] Note that the finiteness of LTS is decidable, and can be checked using the standard coverability tree construction [Rei98].

- S' consists of all triples (α, s, β) such that $s \in S$ and one of the following holds:
  - α is the label of an arc incoming to s and β is the label of an arc outgoing from s.
  - α is the label of an arc incoming to s and $\beta = \epsilon$.
  - $s \in S_0$, $\alpha = \epsilon$ and β is the label of an arc outgoing from s.

  Moreover, the triples from the third case form S'_0.
- Δ' consists of all $\big((\alpha, s_1, \beta), l, (\beta, s_2, \gamma)\big)$ such that $(s_1, \beta, s_2) \in \Delta$ and one of the following holds (below we also give the value of $obs'(l)$):
  - $\alpha = \epsilon = \gamma$, $l = \beta$ and $obs'(l) = obs(\beta)$.
  - $\alpha = \epsilon \neq \gamma$, $l = \beta\gamma$ and $obs'(l) = obs(\beta\gamma)$.
  - $\alpha \neq \epsilon = \gamma$, $l = \alpha\beta$ and $obs'(l) = obs(\alpha\beta)$.
  - $\alpha \neq \epsilon \neq \gamma$, $l = \alpha\beta\gamma$ and $obs'(l) = obs(\alpha\beta\gamma)$.
- ϕ' is true for $(\alpha, s, \beta) \in S'_0$ *iff* ϕ was true for s.

We then observe that the opacity problem for ϕ w.r.t. *obs* is equivalent to the opacity problem for ϕ' w.r.t. *obs'*. Hence, since the new LTS is finite, the former is decidable. ∎

The last result is an extension of the main result given in [BKR04b] which stated the same property for $n = 1$ (as well as for two other kinds of opacity mentioned earlier on).

9.2.2 Approximation of Opacity

As initial opacity is, in general, undecidable when LTSs are allowed to be infinite, we propose in this section a technique which might allow to verify it, at least in some cases, using a technique close to *abstract interpretation* [CC77, CC92]. It uses an abstraction of opacity called under/over-opacity.

Definition 9.9 *For $i = 1, 2, 3$, let Π_i be an LTS. Moreover, let obs_i be an observation function and ϕ_i a predicate for the runs of Π_i such that the following hold:*

$$\begin{array}{lll} (\forall \xi \in run(\Pi_1) \cap \phi_1) & (\exists \xi' \in run(\Pi_2) \cap \phi_2) & obs_1(\xi) = obs_2(\xi') \\ (\forall \xi \in run(\Pi_3) \setminus \phi_3) & (\exists \xi' \in run(\Pi_1) \setminus \phi_1) & obs_3(\xi) = obs_1(\xi') \ . \end{array}$$

Then ϕ_1 is under/over-opaque (or simply uo-opaque) w.r.t. obs_1 if for every $\xi \in run(\Pi_2) \cap \phi_2$ there is $\xi' \in run(\Pi_3) \setminus \phi_3$ such that $obs_2(\xi) = obs_3(\xi')$.

Intuitively, Π_2 provides an over-approximation of the runs satisfying ϕ_1, while Π_3 provides an under-approximation of those runs that do not satisfy ϕ_1.

Proposition 9.10 *Uo-opacity w.r.t. obs_1 implies opacity w.r.t. obs_1.*

Proof: Follows directly from definitions. ∎

Given Π_1, obs_1 and ϕ_1, the idea then is to be able to construct an over-approximation and under-approximation to satisfy the last definition. A possible way of doing this in the case of marked Petri nets is described next.

Uo-opacity for Petri nets Suppose that $\Sigma = (N, S_0)$ is a marked Petri net, $\Pi_1 = \Pi_\Sigma$, obs_1 is a static observation function for Π_1 and $\phi_1 \subseteq S_0$ is an initial opacity predicate for Π_1.

Deriving over-approximation Coverability graphs (see [Fin93] for details) are commonly used to analyze unbounded Petri nets (i.e. Petri nets that have an unbounded number of reachable markings). These graphs are used to approximate a Petri net with a finite automata. Instead of using markings for the nodes of the execution graph of the net, nodes from a coverability graph use ω-markings: a place can be assigned the value ω to indicate an unbounded number of tokens. Consuming or producing tokens from/to ω leads to ω. The integer inequality is extended over $\mathbb{N} \cup \{\omega\}$ by $\forall n \in \mathbb{N},\ \omega > n$.

Classical properties of the coverability graph are that it is a finite graph and for any reachable marking of the original Petri net, a greater (for every place) marking can be reached using the same transitions in the coverability graph.

The over-approximation is obtained by generating the coverability graph Π_2 of Σ starting from the initial nodes in $S_0 \cap \phi_1$. The only modification of the original algorithm needed is that in our setup there may be several starting nodes $S_0 \cap \phi_1$ rather than just one. However, this is a small technical detail. The observation function obs_2 is static and defined in the same way as obs_1. The predicate ϕ_2 is true for all the initial nodes $S_0 \cap \phi_1$. Crucially, Π_2 is always a finite LTS.

Proposition 9.11 $(\forall \xi \in run(\Pi_1) \cap \phi_1)\ (\exists \xi' \in run(\Pi_2) \cap \phi_2)\ \ obs_1(\xi) = obs_2(\xi')$.

Proof: It suffices to prove $\{\lambda \in L^* | \exists s_0 \in S_0 : (s_0, \lambda) \in \phi\} \subseteq \mathcal{L}(\Pi_2)$. Suppose that $(s_0, \lambda) \in run(\Pi_1) \cap \phi_1$. We show, by induction on the length of λ, that there is $(s'_0, \lambda) \in run(\Pi_2)$ such that $s'_0 = s_0$ and $s_0 \oplus \lambda$ is covered by $s'_0 \oplus \lambda$ (i.e., for every place p the value assigned by $s_0 \oplus \lambda$ is not greater than that assigned by $s'_0 \oplus \lambda$).

Since the base case trivially holds, assume that the property is true for $(s_0, \lambda) \in run(\Pi_1) \cap \phi_1$ and that $(s_0, \lambda t) \in run(\Pi_1)$. Then, by the induction hypothesis, there is $(s'_0, \lambda) \in run(\Pi_2)$ such that $s'_0 = s_0$ and $s_0 \oplus \lambda$ is covered by $s'_0 \oplus \lambda$. Since t is enabled in Σ at marking $s_0 \oplus \lambda$, and $s'_0 \oplus \lambda$ covers the latter, t labels an arc outgoing from $s'_0 \oplus \lambda$, leading to some state s. Clearly, s covers $s_0 \oplus (\lambda t)$ since no ω entry in $s_0 \oplus \lambda$ can be replaced by a finite value in s. ■

Deriving under-approximation A straightforward way of finding an under-approximation is to impose a maximal finite capacity max for the places of Σ (for example, by using the complement place construction), and then deriving the LTS Π_3 assuming that the initial markings are those in $S_0 \setminus \phi_1$. The observation function obs_3 is static and defined in the same way as obs_1. The predicate ϕ_3 is false for all the initial nodes $S_0 \setminus \phi_1$.

Clearly, Π_3 is always a finite LTS. However, for some Petri nets with infinite reachability graph (as shown later on by the second example), this under-approximation may be too restrictive, even if one takes an arbitrarily large bound max. Then, in addition to using instance specific techniques, one may attempt to derive more generous under-approximations in the following way.

We assume that there are some (invisible) transitions in Σ mapped by obs_1 to ϵ transitions, and propagate the information that a place could become unbounded due to infinite sequence of invisible transitions. The construction resembles the coverability graph generation.

As in the case of the reachability graph, the states in Π_3 are ω-markings (see the proof of proposition 9.11). Then Π_3 is built by starting from the initial states $S_0 \setminus \phi_1$, and performing a depth-first exploration. At each visited ω-marking M, we find (for example, using a nested call to a coverability graph generation restricted to the invisible transitions starting from M) whether there exists an ω-marking $M' > M$ reachable from M through invisible transitions only[4] ($M' > M$ means that for any place p, $M'(p) \geq M(p)$ and M and M' are different, i.e. there exists a place p such that $M'(p) > M(p)$); then we set $M(p) = \omega$, for every place p such that $M'(p) > M(p)$.

Note that the above algorithm may be combined with the capacity based approach (i.e. replacing $M(p)$ with ω as soon as $M(p)$ is greater than a fixed bound) and then it always produces a finite Π_3. In general, however, Π_3 is not guaranteed to be finite.

It should be pointed out that Π_3 generated in this way is not, in general, a deterministic LTS, but this does not matter as the only thing we are interested in is the language it generates.

[4] This search does not have to be complete for the method to work, however, the more markings M' we find, the better the overall result is expected to be.

Proposition 9.12 $(\forall \xi \in run(\Pi_3) \setminus \phi_3)\ (\exists \xi' \in run(\Pi_1) \setminus \phi_1)\ \ obs_3(\xi) = obs_1(\xi')$.

Proof: It suffices to show that $obs_3(\mathcal{L}(\Pi_3)) \subseteq obs_1(\mathcal{L}(\Pi_1))$.
Suppose that $s_0 t_1 s_1 \ldots t_n s_n$ is a path in Π_3 starting from an initial state. We show, by induction on n that, for every $k \geq 1$, there is $(s_0, \lambda) \in run(\Pi_1)$ such that $obs_1(\lambda) = obs_3(t_1 \ldots t_n)$ and, for every place p, either $s_n(p) \leq (s_0 \oplus \lambda)(p)$ or $\omega = s_n(p) > (s_0 \oplus \lambda)(p) \geq k$.

Since the base case trivially holds, assume that the property holds for n, $s_0 t_1 s_1 \ldots t_n s_n t s$ is a path in Π_3 starting from an initial state, and $k \geq 1$. Moreover, let $t'_1 \ldots t'_q$ $(q \geq 0)$ be a sequence of invisible transitions which was 'responsible' for replacing the marking resulting from executing t in s_n by s.
Given now an arbitrary $m \geq 1$ we can choose sufficiently large $k' \geq k$ such that, after applying the induction hypothesis, there is $(s_0, \lambda) \in run(\Pi_1)$ satisfying:

- $obs_1(\lambda) = obs_3(t_1 \ldots t_n)$ and, for every place p, either $s_n(p) \leq (s_0 \oplus \lambda)(p)$ or $\omega = s_n(p) > (s_0 \oplus \lambda)(p) \geq k'$.
- t followed by m repetitions of the sequence $t'_1 \ldots t'_q$ is executable in Σ from the marking $s_0 \oplus \lambda$.

The latter, in particular, means that the overall effect of the sequence $t'_1 \ldots t'_q$ on the marking of any place is that the number of tokens never decreases (otherwise 'negative' place markings would be eventually generated which is impossible). This in turn means that, by executing $t'_1 \ldots t'_q$ sufficiently many times from the marking $s_0 \oplus (\lambda t)$, we may reach a marking s' such that, for every place p, either $s(p) \leq s'(p) \in \mathbb{N}$ or $\omega = s(p) > s'(p) \geq k$. ■

Deciding uo-opacity Assuming that we have successfully generated over- and under-approximations Π_2 and Π_3, uo-opacity holds *iff*

$$obs_2(\mathcal{L}(\Pi_2)) \subseteq obs_3(\mathcal{L}(\Pi_3))$$

When Π_2 and Π_3 are *finite* LTSs, then $obs_2(\mathcal{L}(\Pi_2))$ $obs_3(\mathcal{L}(\Pi_3))$ are regular languages. Inclusion of two regular languages is a decidable problem [5] thus the previous inclusion can also be decided.

9.2.3 Examples

To illustrate our work, we give two examples. The first one is inspired by an anonymity requirement required in the chemical industry. The second describes a simple voting system.

9.2.4 A Scenario from Chemical Engineering

Figure 9.1 is a Petri net representation of a scenario in the chemical industry. It is adapted from an example presented in [PTE04]. In the example, a chemical development company A asks company B (transition a_1) to prepare a feasibility study into the development of a new chemical. When this is completed (transition b_1) company A is informed of the conclusions (transition a_2). On the basis of these conclusions company A decides to commission a chemical safety report, from either company C (transition a_3) or company C' (transition a'_3). The relevant law allows the chosen company to question company B on aspects of the feasibility study. However, the chosen company is not allowed to reveal its identity to company B, in order to protect the integrity of B's answers. In our example, there are only two possible companies, C and C', so our intention is that from B's point of view, the visible interactions do not reveal the identity of the chosen company.

We may assume that the actions a_3, a'_3, a_4 and a'_4 are not visible to B, as these actions concern only companies A and B.

[5] Regular languages are accepted by finite automatons. "Intersection" of two such automatons can be computed as well as "complement". Moreover emptiness is decidable. The result follows as $A \subseteq B$ is equivalent to $\neg \cap B = \varnothing$.

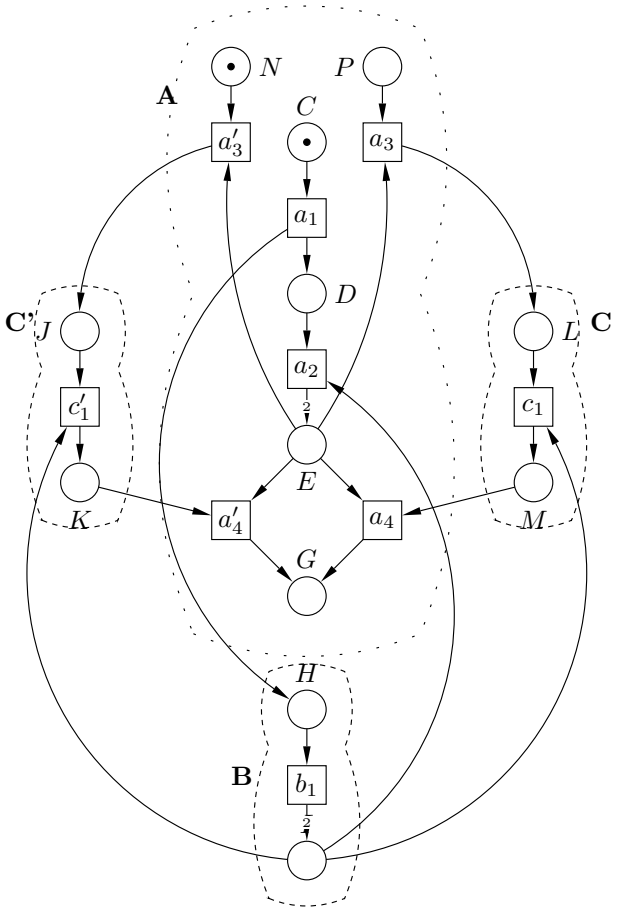


Figure 9.1: Petri Net for the chemical industry scenario.

We choose the (static) observation function of B to be the identity function, except for

$$\begin{array}{lllllllll} obs'(a_3) & = & obs'(a_4) & = & \epsilon & & obs'(c_1) & = & \gamma \\ obs'(a'_3) & = & obs'(a'_4) & = & \epsilon & & obs'(c'_1) & = & \gamma \end{array}$$

We now demonstrate the set of transitions $\{c_1, c'_1\}$ to be O-anonymous.
If $\lambda = l_i \dots l_n$, the properties that we require to be opaque w.r.t. *obs* are:

$$\phi(s,\lambda) = (\exists i : l_i = c_1) \quad \text{and} \quad \phi'(s,\lambda) = (\exists i : l_i = c'_1)$$

The two possible sequences of actions of this system are $a_1b_1a_2a_3c_1a_4$ and $a_1b_1a_2a'_3c'_1a'_4$, and so the two possible observations of the system are

$$\begin{array}{lll} obs(a_1b_1a_2a_3c_1a_4) & = & a_1b_1a_2\epsilon\gamma\epsilon \\ obs(a_1b_1a_2a'_3c'_1a'_4) & = & a_1b_1a_2\epsilon\gamma\epsilon \end{array}$$

which are observationally equivalent. The properties ϕ and ϕ' are therefore opaque, and the set $\{c_1, c'_1\}$ is strongly anonymous w.r.t. *obs*.

9.2.5 A Simple Voting Scheme

In this example, we consider a vote session allowing only two votes: 1 and 2. We then describe a simple voting scheme in the form of a Petri net (see figure 9.2). The voting scheme contains two

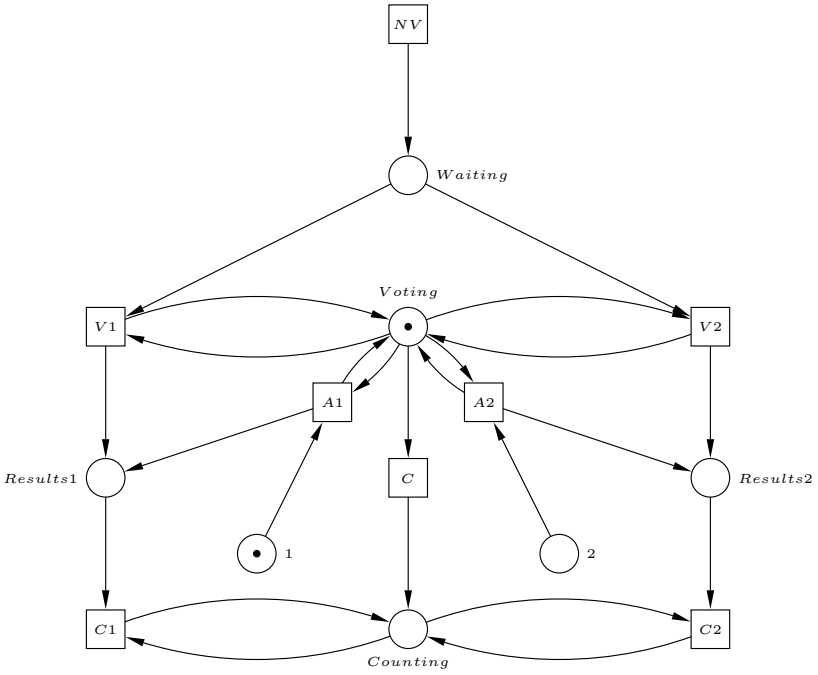


Figure 9.2: Net for the voting system.

phases. The first one called *voting phase* (when there is a token in Voting) allows any new voter to enter the polling station (transition NV) and vote (transitions $V1$ and $V2$). Votes are stored in two places $Results1$ and $Results2$. A particular voter A is identified, and we formulate our properties with respect to A. After an indeterminate time, the election enters the *counting phase* (when there is a token in Counting, after executing transition C, and no token in Voting). Then the different votes are counted. Votes for 1 are seen via transition $C1$ and vote for 2 via $C2$. This net has one obvious limitation. At the end, there still can be some tokens left in places Results1 and Results2 so this scheme does not ensure that every vote is counted.

We want to verify that the vote cast by A is secret: the two possible initial markings are $\{Voting, 1\}$ and $\{Voting, 2\}$. We prove that it is impossible to detect that "1" was marked (a symmetric argument would show that it is impossible to detect whether "2" was marked). The observation function is static and only transitions $C1$ and $C2$ are visible, i.e., $obs(C1) = C1$, $obs(C2) = C2$ and $obs(t) = \epsilon$ for any other transition t.

To verify opacity, we use the under/over approximation method. The coverability graph (over-approximation) can be computed (see figure 9.3) using, for example, Tina [tin04]. After application of the observation function and simplification, we obtain that $obs_2(\mathcal{L}(\Pi_2)) = \{C_1, C_2\}^*$(see section 9.2.2 for the definition of Π_2).

However, the simple under approximation using bounded capacity places does not work in this case, as for any chosen maximal capacity max, the language $\mathcal{L}(\Pi_3)$ is finite whereas $obs_2(\mathcal{L}(\Pi_2))$ is infinite. Thus, we use the second under approximation technique. The following array represents the reachable states of the system starting from marking $\{Voting, 2\}$ using this technique.

	$Waiting$	$Voting$	$Results1$	$Results2$	1	2	$Counting$
A	$\mathbb{N}$	1	$\mathbb{N}$	$\mathbb{N}$	0	1	0
B	$\mathbb{N}$	1	$\mathbb{N}$	$\mathbb{N}$	0	0	0
C	$\mathbb{N}$	0	$\mathbb{N}$	$\mathbb{N}$	0	1	1
D	$\mathbb{N}$	0	$\mathbb{N}$	$\mathbb{N}$	0	0	1

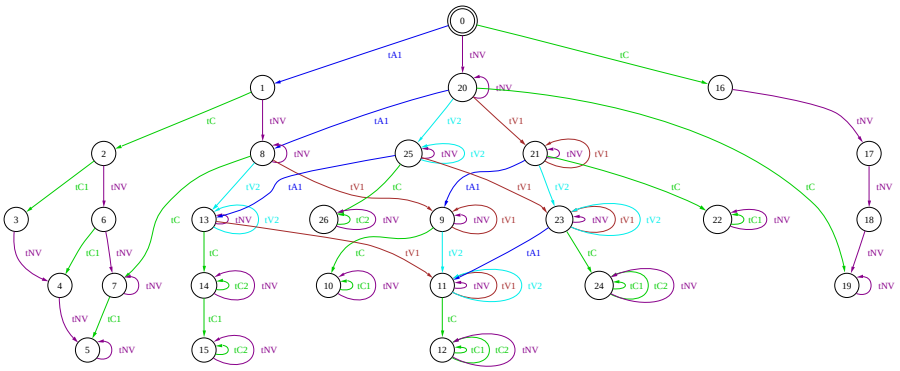
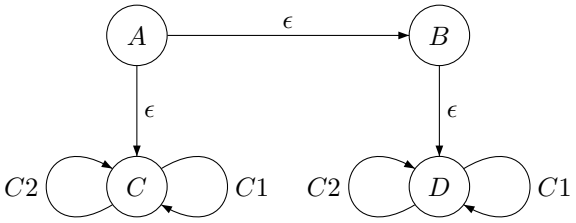


Figure 9.3: Coverability graph for the voting system.

The behavior of this reachability graph, i.e. $obs_3(\mathcal{L}(\Pi_3))$, is simple:



Thus, the under-approximation is in this case: $obs_3(\mathcal{L}(\Pi_3)) = \{C1, C2\}^*$, and so $obs_2(\mathcal{L}(\Pi_2)) \subseteq obs_3(\mathcal{L}(\Pi_3))$ holds. We can now conclude that opacity of ϕ w.r.t. *obs* is verified and so the vote cast by A is kept secret.

9.3 Opacity as Indistinguishability

In this section, we are only interested in initial opacity. Protocols as we described them earlier are not general enough to represent more complex systems such as voting schemes. Hence, we consider general LTSs where labels are messages. In order to link opacity to indistinguishability, a first step consists in defining probabilistic opacity.

9.3.1 Probabilistic Opacity

Let Σ be a finite alphabet representing actions made by a system. A trace is a finite sequence of actions, i.e., a word over Σ. Let Σ^* be the set of words over Σ and ϵ be the empty word.

A *system* is a random function Δ from a finite set S of initial states to sequences of actions. Random means that the system can perform some non-deterministic operations (with discrete probabilistic choices). For example it can pick up a random bit b. If $b = 0$, it performs action a else action a'. The set of possible traces of a system Δ is denoted by $\Delta(S)$. In a similar way, $\Delta(\{s\})$ is the set of possible traces starting from s in S. This set can have more than one element as function Δ is random. We suppose that Δ can be represented using a PRTM.

An *observation function* allows the eavesdropper to see only limited information about traces produced by the studied system. These functions are mappings from Σ to $\Sigma \cup \{\epsilon\}$. Hence, it is possible for an action to be totally invisible from the outside if the observation function replaces it with ϵ.

Opacity Let us consider a system Δ with possible initial states S and an observation function *obs*. A property ψ is a predicate over S which satisfiability is a polynomial time problem. A property ψ is *opaque* if given $s \in S$ and $t \in \Delta(\{s\})$, it is not possible, for an adversary that has access uniquely access to $obs(t)$ to know whether s verifies ψ. Here, not possible means that it should not be possible to achieve this with "reasonable" probability. This notion of opacity corresponds to initial opacity as introduced in section 9.1.

More than in opacity itself, we are interested here in the advantage that an adversary can get by having access to the observation of the trace. If a vast majority of initial states in S verify ψ, the adversary can assume that s verifies ψ even without looking at the trace. However, by looking at the trace, it is possible to get some new information and to deduce the result for sure. To define this advantage, we consider that the adversary $\mathcal{A}$ tries to win the following game:

1. An initial state s is chosen randomly in S;
2. The adversary $\mathcal{A}$ is given the observation of a trace in $\Delta(\{s\})$ and has to output a bit b;
3. $\mathcal{A}$ wins his challenge, when b is equivalent to the property "s satisfies ψ".

This game is represented by an experiment which is a random Turing machine. The experiment related to adversary $\mathcal{A}$ and to *obs* is the following PRTM.

Game $\mathbf{G}_{\mathcal{A}}^{obs}$:
$s \xleftarrow{R} S$
$t := \Delta(s)$
$b := \mathcal{A}(obs(t))$
return $b \Leftrightarrow (s \in \psi)$

As in chapter 4 the advantage is the difference between the probability that $\mathcal{A}$ solves his challenge and the best probability that one can get without access to the observation. Hence, it is defined by the following formula.

$$\mathbf{Adv}_{\mathcal{A}}^{obs} = 2\left(Pr[\mathbf{G}_{\mathcal{A}}^{obs} \to true] - PrRand^{\psi}\right)$$

Where $PrRand^{\psi}$ is the greatest possible value for $Pr[\mathbf{G}_{\mathcal{B}}^{\epsilon} \to true]$ for any $\mathcal{B}$ and ϵ represents the observation function that associates ϵ to any action in Σ.

Note that the above definitions for the experiment and the advantage can easily be defined in an equivalent way by using the general notion of security criterion.

- Θ randomly generates an initial parameter s and a trace t;
- F gives access to the trace observation $obs(t)$;
- V verifies that the output bit b correctly answers the question: does s verify ψ ?

Criterion $(\Theta; F; V)$ has exactly the same related experiment and advantage as those given above.

Using the definition of advantage, it is possible to tell whether an observation function is of any use in trying to solve the challenge.

Definition 9.10 *Let Δ be a system, S be the set of its initial states and ψ a property over S. An observation function obs is called*

- safe for **strict** opacity *of ψ , if for any random Turing machine $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{obs} = 0$*
- safe for **cryptographic** opacity *of ψ, if for any polynomial random Turing machine $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{obs}$ is negligible*
- safe for **plausible deniability** *of ψ, if for any random Turing machine $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{obs} \neq 2 - 2PrRand^{\psi}$ i.e. $Pr[\mathbf{Exp}_{\mathcal{A}}^{obs} \to true]$ is different from 1.*

Plausible deniability coincides with the opacity notion introduced in section 9.1. However in this section we are more interested in cryptographic opacity than in plausible deniability.

For strict opacity, if an observation function is safe then an adversary gets no advantage at all by looking at the observation. This is for example the case of information exchanged by cryptographers during their dinner (in the original proposal of Chaum) if we consider that one of them paid the diner for any element of S.

For cryptographic opacity, an observation function may return some relevant information that cannot be exploited in a reasonable (i.e. polynomial) time. For example, if we consider that all the actions made by a system are encrypted using a safe encryption scheme, then the observation is safe. In this context, a safe encryption scheme is an IND-CPA encryption scheme as introduced earlier in this document.

The idea is that with plausible deniability, if the adversary observes some trace t, then he cannot conclude for sure whether property ψ is verified or not. There exists at least one initial state satisfying ψ and one not satisfying ψ that both produce the observation $obs(t)$.

There is no clear hierarchy among strict opacity and plausible deniability as the first notion does not imply the second one (this implication is only true when $PrRand$ is different from one).

9.3.2 Decidability of Strict Opacity for Finite Systems

Let us consider the case where only a finite number of traces can occur. Thus, we suppose that both S and $\Delta(S)$ are finite. With this assumption, the greatest advantage for any adversary can be computed. Moreover, there exists an adversary that reaches this advantage.

Let O be the set of all possible observations, i.e. $O = obs(\Delta(S))$. We first define the interest of an observation function obs. This definition is rather intuitive as an observation function can bring some advantage if the probability for ψ to be true knowing the observation is different from the general probability of ψ. This explains why this definition uses the term $|Pr[\psi] - Pr[\psi|o]|$.

Definition 9.11 *The interest I_{obs} of an observation function obs is given by:*

$$I_{obs} = 2 \sum_{o \in O} Pr[o] \big| Pr[\psi] - Pr[\psi|o] \big|$$

Then, the main result of this section is that the interest is the least possible advantage. For this reason, as it is possible to effectively compute the interest of a given observation, safety for strict opacity of an observation function is a decidable problem. The following proposition states the main result.

Proposition 9.13 *For any adversary $\mathcal{A}$ and observation function obs, $\mathbf{Adv}^{obs}_{\mathcal{A}} \geq I_{obs}$. Moreover, there exists an adversary $\mathcal{A}_{obs}$ whose advantage is exactly I_{obs}.*

Proof: This proof is achieved in three steps:

Step 1 First, consider the case where there is only one possible observation, $|O| = 1$. Adversary $\mathcal{A}$ has a probability p (resp. $1-p$) to answer 1 (resp. 0).

$$\begin{aligned} Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \rightarrow true] &= Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \rightarrow true | s \in \psi] Pr[\psi] + Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \rightarrow true | s \notin \psi] Pr[\neg\psi] \\ &= p.Pr[\psi] + (1-p)(1 - Pr[\psi]) \end{aligned}$$

Then, if $Pr[\psi] \geq \frac{1}{2}$, $1 - 2Pr[\psi]$ is negative hence:

$$\begin{aligned} Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \rightarrow true] &= (1-p)(1-2Pr[\psi]) + Pr[\psi] \\ &\leq Pr[\psi] \end{aligned}$$

In the other case, $2Pr[\psi] - 1$ is negative,

$$\begin{aligned} Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \rightarrow true] &= (1 - Pr[\psi]) + p(2Pr[\psi] - 1) \\ &\leq 1 - Pr[\psi] \end{aligned}$$

These two inequalities allow us to deduce the following.

$$\begin{aligned} Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \to true] &\leq max(Pr[\psi], 1 - Pr[\psi]) \\ &= |Pr[\psi] - \frac{1}{2}| + \frac{1}{2} \end{aligned}$$

$max(Pr[\psi], 1 - Pr[\psi])$ is exactly the best advantage that an adversary which does not have access to any oracle can get. This adversary always returns 1 if $Pr[\psi]$ is greater than $1/2$ and 0 otherwise.

$$Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \to true] \leq PrRand^{\psi}$$

Hence, the advantage is negative.

Step 2 Now, it is possible to generalize the above result for any set O.

$$\begin{aligned} Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \to true] &= \sum_{o \in O} Pr[o] Pr[\mathbf{Exp}^{obs}_{\mathcal{A}} \to true | o] \\ &\leq \sum_{o \in O} Pr[o] \big(\big| Pr[\psi|o] - \frac{1}{2} \big| - \frac{1}{2} \big) \\ &\leq \frac{1}{2} + \sum_{o \in O} Pr[o] \big| Pr[\psi|o] - \frac{1}{2} \big| \end{aligned}$$

We introduce $PrRand^{\psi}$ in the former equation using its form $|Pr[\psi] - \frac{1}{2}| + \frac{1}{2}$. Hence,

$$\begin{aligned} \mathbf{Adv}^{obs}_{\mathcal{A}} &\leq \sum_{o \in O} Pr[o] \big(\big| Pr[\psi|o] - \frac{1}{2} \big| - \big| Pr[\psi] - \frac{1}{2} \big| \big) \\ |\mathbf{Adv}^{obs}_{\mathcal{A}}| &\leq \sum_{o \in O} Pr[o] \big| \big| Pr[\psi|o] - \frac{1}{2} \big| - \big| Pr[\psi] - \frac{1}{2} \big| \big| \\ &\leq \sum_{o \in O} Pr[o] \big| Pr[\psi|o] - Pr[\psi] \big| \\ &\leq I_{obs} \end{aligned}$$

Step 3 The machine $\mathcal{A}_{obs}$ which advantage is exactly I_{obs} is very simple:

Adversary $\mathcal{A}_{obs}(o)$:
  $p := Pr[\psi|o]$
  if $p \geq 0.5$, **return** $true$
  else return $false$

Probability for the different *obs* equivalence classes are hardwired in the machine. As there are only a finite number of classes, $\mathcal{A}_{obs}$ works in polynomial time (in the size of o). The advantage of this adversary can be computed in a similar way as step 1 and 2. ■

It is important to notice that the second statement of the previous proposition asserts the existence of an RTM $\mathcal{A}_{obs}$ with $\mathbf{Adv}^{obs}_{\mathcal{A}} = I_{obs}$. This RTM is not necessarily a legal adversary. Indeed, $\mathcal{A}_{obs}$ has an execution time which is linear in the number of possible observations. This is not a problem when considering strict opacity or plausible denying as adversaries are RTM. However, for cryptographic opacity, we only admit adversaries in PRTM. Worse, even if we assume the quite fair hypothesis (for an eavesdropper) that there is only a bounded number of messages which all have some bounded size, the number of possible observations may be exponential in the security parameter η, and hence, $\mathcal{A}_{obs}$ may not a PRTM.

Nevertheless, this result is interesting at least for strict opacity as shown now. Indeed, a consequence of this proposition is that no adversary has an advantage if and only if for every o in O, $Pr[\psi] = Pr[\psi|o]$. When one wants to verify strict opacity, it is possible to test that for any observation, the probability for ψ to be true assuming this observation is exactly the general probability for ψ to be true. Hence, we have

Proposition 9.14 *Let obs be an observation function and ψ a property. Then, obs is safe for strict opacity of ψ if and only if for any $o \in obs(\Delta(S))$, $Pr[s \in \psi] = Pr[s \in \psi | obs(s) = o]$.*

9.3.3 An Approach to the Verification of Cryptographic Opacity

As we noted in the previous section, we make the finite behavior hypothesis for cryptographic systems (with passive adversaries). That is, we assume that S and $\Delta(S)$ are finite. The approach proposed for proving strict opacity using the interest of the observation function and the existence of an adversary matching this interest is not applicable for cryptographic opacity. Indeed, when considering cryptographic opacity, adversaries are polynomially bounded and hence cannot perform brute-force attacks. Therefore, we present here a different approach. The main idea in this approach is to decompose the verification of cryptographic opacity into the verification of strict opacity for an abstracted system on one hand and the safety of the underlying encryption scheme on the other hand.

Specifications and Patterns In the cryptographic setting, the alphabet Σ consists of the symbols 0 and 1. Thus, an action of the alphabet is a bit-string. We consider systems that produce some finite size bit-strings (usually, their size is polynomial in the security parameter η).

To define the abstract systems, we introduce a new kind of patterns. There is a difference with previously introduced patterns as we are interested in opacity and want to tackle the Chaum voting scheme: we have to be careful in handling the random coins[6] used in encryptions. Let us describe this precisely: in the simplest Dolev-Yao model, encryption of a message m with key pk is represented by term $\{m\}_{pk}$. Thus, two messages $\{m_1\}_{pk_1}$ and $\{m_2\}_{pk_2}$ are equal iff $m_1 = m_2$ and $pk_1 = pk_2$. Moreover, an adversary who does not know the inverse key of pk cannot get any information from $\{m\}_{pk}$. This means that the randomness used to perform the encryption is completely abstracted away. In some refinements of this model, however, labels are introduced to distinguished encryptions made at different instants during a protocol execution [CW05]. Such labels are only an approximation of random coins as the latter may be equal even when two encryptions are performed at different instants. As we want to verify the Chaum voting scheme (in which random coins used for some encryptions are sent afterwards to allow verification), we have to include explicitly random coins in our patterns and we want to be able to send and receive such coins. Therefore, we write $\{m; N\}_{pk}$ to represent the result of encrypting m with key pk using nonce N as random coins for the encryption algorithm.

Patterns are defined by the following grammar where k is a key, bs a bit-string and N is a nonce:

$pat ::= bs$		bit-string
$\mid$	N	nonces
$\mid$	k	k may be a public or private key
$\mid$	$\langle pat, pat \rangle$	pairing of two patterns
$\mid$	$\{pat; N\}_{pk}$	encryption of pat using key pk with randomness N

Without loss of generality, we consider abstract systems that only produce one pattern and not a list of patterns as it is possible to concatenate patterns using pairing. Thus, a *specification* Δ_s is a function from S to pat.

Obviously, given a pattern pat the information that can be extracted from pat depends on the set of private keys that can be computed from pat. We define the set of patterns $dec(p)$ that can be learned/computed from a pattern p.

Definition 9.12 *Let p be a pattern, the set $dec(p)$ is inductively defined by the following inferences.*

[6] Random coins are also nonces but some times we use rather random coins to insist on the fact they are used to randomize encryptions.

- *p is in $dec(p)$.*
- *If $\langle p_1, p_2\rangle$ is in $dec(p)$, then p_1 and p_2 are in $dec(p)$.*
- *If $\{p_1; N\}_k$ and k^{-1} are in $dec(p)$, then p_1 and N are in $dec(p)$.*
- *If $\{p_1; N\}_k$ and N are in $dec(p)$, then p_1 is in $dec(p)$.*

Notice that since we only consider atomic keys, we only have to consider decompositions. It is also useful to notice that the last clause is usually not considered in the Dolev-Yao model. This clause is motivated by the existence of IND-CPA algorithms such that the knowledge of the random number used for encryption allows one to decrypt the message. An example of such an algorithm is presented in [BCP03].

A pattern has also a denotation in the cryptographic setting. This depends on a context θ that associates keys and nonces with their corresponding bit-string values. Thus, the cryptographic (or computational) value of a term $\{pat; N\}_k$ is $\mathcal{E}(m, bs, bs')$, where m is the value of pat, bs the value of pk, bs' the value N and $\mathcal{E}$ is the deterministic encryption scheme whose third argument consists in random coins. Let θ be a mapping associating bit-strings with nonces and keys. The value of a pattern in the context θ is defined recursively by the *concr* function:

$$\begin{array}{ll} concr(bs, \theta) = bs & concr(\langle p_1, p_2\rangle, \theta) = concr(p_1, \theta).concr(p_2, \theta) \\ concr(N, \theta) = \theta(N) & concr(\{p; N\}_k, \theta) = \mathcal{E}(concr(p, \theta), \theta(k), \theta(N)) \quad concr(k, \theta) = \theta(k) \end{array}$$

Let us briefly summarize what we have introduced. We defined the systems we want to consider whose behavior in each initial state s is a set of patterns and we have associated with each pattern its value, a bit-string, in a given context.

We now turn our attention to the observations we can make about a pattern. We define two observations. The concrete observation of a pattern pat in a context θ is defined as follows: $obs_c(pat, \theta) = concr(pat, \theta)$, that is, obs_c corresponds to the observations that can be made in the cryptographic setting. The abstract observation obs_a applied to a pattern outputs the skeleton of the pattern, it replaces every sub-term of the form $\{pat; N\}_{pk}$ with $\diamondsuit^N_{pk}$ (i.e. it simply replaces it by a black box) when pk^{-1} is not deducible. Formally, patterns are transformed into obfuscated patterns which are given by the following grammar:

$$opat ::= bs | N | \langle opat, opat\rangle | \{opat; N\}_k | \diamondsuit^N_k | k$$

And observation obs_a of a pattern pat is recursively defined by the following rules.

$$\begin{array}{ll} obs_a(bs) = bs & obs_a(\langle p_1, p_2\rangle) = \langle obs_a(p_1), obs_a(p_2)\rangle \\ obs_a(N) = N & obs_a(\{p; N\}_k) = \{obs_a(p); N\}_k \text{ if } k^{-1} \in dec(pat) \vee N \in dec(pat) \\ obs_a(k) = k & obs_a(\{p; N\}_k) = \diamondsuit^N_k \text{ else} \end{array}$$

As encryption cycles may lead to some vulnerabilities, we restrict ourselves to *well-formed* patterns. For this purpose, we define an ordering on pairs consisting of a key and a nonce. Let pat be a pattern and let $E_<$ be the set of pairs (pk, N) such that there is a pattern of the form $\{pat'; N\}_{pk}$ in $dec(pat)$ with pk and N not in $dec(pat)$. Then, for $(pk, N), (pk', N') \in E_<$, $(pk, N) < (pk', N')$ iff there exist two patterns $\{pat_1; N\}_{pk}$ and $\{pat_2; N'\}_{pk'}$ in $dec(pat)$ verifying one of the following conditions:

1. N, pk or pk^{-1} is a sub-term of $pat_2; N'$;
2. $N = N'$ and $\{pat_1; N\}_{pk} \neq \{pat_2; N\}_{pk'}$.

A pattern pat is *well-formed*, if the projection of $<$ on keys is acyclic. Finally, we only consider *well-formed specifications*, i.e. specifications that output well-formed patterns.

The conditions above imply that if pat is well-formed, then for $(pk, N) \in E_<$, there is only one encoding using each N (and a non-deducible key) in $dec(pat)$. Hence when obs_a transforms an encoding into $\diamondsuit^N_k$, this always denotes the exact same encoding (in particular, there is no

randomness-reuse as described in [BBS03]). Thus the N label can be seen as a constraint over encodings (specifying possible bit-to-bit equalities). This is why, equality between two *opat* is defined modulo renaming of the nonces.

To illustrate this, let us consider two patterns $pat_0 = \langle \{m; N\}_k, \{m; N\}_k \rangle$ and $pat_1 = \langle \{m; N''\}_k, \{m; N'\}_k \rangle$. Then $obs_a(pat_0) = \langle \diamondsuit_k^N, \diamondsuit_k^N \rangle$ and $obs_a(pat_1) = \langle \diamondsuit_k^{N'}, \diamondsuit_k^{N''} \rangle$. As $obs_a(pat_1)$ and $obs_a(pat_2)$ are different, pat_0 and pat_1 are distinguishable. If we consider $pat_0 = \{m; N\}_k$ and $pat_1 = \{m; N'\}_k$, then $obs_a(pat_0) = \diamondsuit_k^N$ and $obs_a(pat_1) = \diamondsuit_k^{N'}$. In this case, $obs_a(pat_1)$ and $obs_a(pat_2)$ are equal (modulo renaming), and hence, pat_0 and pat_1 are indistinguishable. And in fact, if the encryption scheme is IND-CPA, pat_0 and pat_1 are indistinguishable even in the computational setting.

Main result The main result of this chapter is that for each adversary $\mathcal{A}$ (observing both obs_c and obs_a, there exist two adversaries $\mathcal{A}^o$ (observing obs_a) and $\mathcal{B}$ (against a variant of IND-CPA) such that

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

Where n is the number of keys, n-RPAT-CPA is a security criterion verified by any IND-CPA algorithm, observable $obs_c \times obs_a$ gives access to both obs_c and obs_a. This means that if the encryption scheme used is IND-CPA, strict opacity in the symbolic world implies cryptographic opacity in the computational world.

In order to prove our main result, we proceed in two steps. We first define n-RPAT-CPA and relate it to IND-CPA. Then, we prove that the advantage of any adversary who accesses the observation functions obs_a and obs_c is bounded by a linear combination of the advantage of an adversary that has access to obs_a and the advantage of an adversary that has access to obs_c. In both steps, we apply Theorem 5.1. Although the criterion we introduce is implied by IND-CPA, it is technically more appealing to use it to prove the main result. Besides this, our new criterion is of interest on its own as it clarifies and discloses some subtleties related to the treatment of random coins.

The RPAT Extension to IND-CPA

In IND-CPA, the experiment consists of generating a random bit b and a random public key pk. The adversary tries to guess the value of b. For this purpose, it accesses a *left-right oracle* submitting two bit-strings bs_0 and bs_1 and receives the encryption of bs_b using pk. The adversary also has access to the public key. An encryption scheme $\mathcal{AE}$ is said secure against IND-CPA if any PRTM has a negligible advantage in trying to find b (the advantage is twice the probability to answer correctly minus one). The criterion we introduce below allows the adversary to ask for encryption of patterns where challenged keys may be included and insisting on using the same random coins in different encryptions. Moreover, patterns may include encryption with the adversary keys. As we show later these extensions do not give more power to an adversary, if he only makes queries with well-formed patterns.

Let us now introduce n-RPAT-CPA. To do so, let n be a non-negative integer. We first define R-patterns:

$$rpat ::= bs | N | \langle rpat, rpat \rangle | \{rpat; N\}_k | \{rpat; N\}_{bs} | \{rpat; bs\}_k | \{rpat; bs\}_{bs'} | k$$

The only difference with respect to patterns introduced in section 9.3.3 is the encryption with a non-challenge key or a non-challenge nonce. The evaluation function v is extended to R-patterns.

The experiment defining the criterion n-RPAT-CPA is as follows:

$$
\begin{array}{ll}
pat_0, pat_1, \sigma \leftarrow \mathcal{A}_1; & \\
b \leftarrow \{0,1\}; & \\
(bs_i, bs'_i) \leftarrow \mathcal{KG}(\eta); & \text{for } i = 1, \cdots, n \\
bs''_i \leftarrow \{0,1\}^\eta; & \text{for } i = 1, \cdots, l \\
\theta \leftarrow [b \mapsto b, (pk_1, sk_1) \mapsto (bs_1, bs'_1), \cdots, (pk_n, sk_n) \mapsto (bs_n, bs'_n), & \\
\quad N_1 \mapsto bs''_1, \cdots, N_l \mapsto bs''_l]; & \\
y \leftarrow v(pat_b, \theta); & \\
d \leftarrow \mathcal{A}_2(y, \sigma) & \\
V(d, \theta) \leftarrow b = d &
\end{array}
$$

The adversary is split up in two parts $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{A}_1$ outputs two patterns pat_0 and pat_1. Pattern pat_b is computed (l is the number of nonces used by the pattern and n is the maximal number of keys that a pattern can use), it is given to $\mathcal{A}_2$ which has to answer the value of b. It is also possible to consider a single adversary $\mathcal{A}$ that access a left-right oracle $\mathcal{O}_{LR}$, giving it the two patterns. In this case, oracle $\mathcal{O}_{LR}$ only answers its first call.

In the experiment of n-RPAT-CPA, it is required that $\langle pat_0, pat_1 \rangle$ is well-formed and that $obs_a(pat_0)$ and $obs_a(pat_1)$ are equal.

We show that algorithms secure w.r.t. IND-CPA are secure w.r.t. n-RPAT-CPA and as there are algorithms strongly believed to verify IND-CPA, these algorithms also verify n-RPAT-CPA.

Proposition 9.15 *If an asymmetric encryption scheme is secure against IND-CPA, then it is secure against n-RPAT-CPA for any number of keys n.*

Proof: Let $\mathcal{AE}$ be an encryption scheme. We proceed in three steps.

Let n-RPAT$_c$-CPA be the same criterion as n-RPAT-CPA except that adversaries can only output clean patterns, i.e. $\langle pat_0, pat_1 \rangle$ such that $dec(pat_0, pat_1)$ does not contain any nonce or any private key. Our first step consists in proving that IND-CPA implies 1-RPAT$_c$-CPA.

Lemma 9.1 *If $\mathcal{AE}$ is secure w.r.t. IND-CPA then it is secure w.r.t. 1-RPAT$_c$-CPA.*

Proof: Let us consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against 1-RPAT$_c$-CPA. We construct an adversary $\mathcal{B}$ against IND-CPA whose advantage is the same as the advantage of $\mathcal{A}$.

The only challenge key pair is (pk_1, sk_1). As there are no cycles among keys, there does not exist any pair of nonces N, N' such that $(pk_1, N) < (pk_1, N')$. Hence relation $<$ is empty. For any nonce N such that $(pk_1, N) \in E_<$, N appears in exactly one encoding (but this encoding can be used several times as in $\langle \{m; N\}_{pk_1}, \{m; N\}_{pk_1} \rangle$) and in this case it appears as a random coin.

The adversary $\mathcal{B}$ uses $\mathcal{A}_1$ and $\mathcal{A}_2$ as sub-machines. However, as $\mathcal{A}_1$ outputs patterns while $\mathcal{B}$ has to output messages, $\mathcal{B}$ has to simulate the evaluation function v using the IND-CPA left-right oracle. This is done using the function $vsim$ in the description of $\mathcal{B}$:

$$
\begin{array}{l}
pat_0, pat_1, \sigma \leftarrow \mathcal{A}_1; \\
y \leftarrow vsim(pat_0, pat_1); \\
d \leftarrow \mathcal{A}_2(y, \sigma); \\
\textbf{return } d
\end{array}
$$

We now have to describe the function $vsim$. First notice that nonces N that do not appear in $E_<$ appear encrypted in the patterns. Therefore $vsim$ generates some random values for these nonces and creates the corresponding environment θ_{sim}. The context θ_{sim} is extended with public key k. Next, as pat_0 and pat_1 have the same obs_a (modulo renaming), the following recursive function $vrec_{\theta_{sim}}$ is applied to pat_0, pat_1:

$$
\begin{array}{rcl}
vrec_{\theta_{sim}}(bs, bs) & = & bs \\
vrec_{\theta_{sim}}(m_1.m_2, m'_1.m'_2) & = & vrec_{\theta_{sim}}(m_1, m'_1).vrec_{\theta_{sim}}(m_2, m'_2) \\
vrec_{\theta_{sim}}(\{m; N\}_{pk_1}, \{m'; N\}_{pk_1}) & = & \mathcal{O}_{LR}\big(v(m, \theta_{sim}), v(m', \theta_{sim})\big)
\end{array}
$$

Note that for the last line, if $vrec_{\theta_{sim}}$ is called twice on the exact same patterns, then the same value has to be returned (so it is necessary to store the value although this is not done here to preserve simplicity). Finally, $vsim(pat_0, pat_1)$ returns $vrec_{\theta_{sim}}(pat_0, pat_1)$.

The experiments involving $\mathcal{B}$ and $\mathcal{A}$ are the same and as $PrRand$ is equal to 1/2 for both criteria, the advantages of $\mathcal{B}$ and $\mathcal{A}$ are equal. ∎

The second step is to show that 1-RPAT$_c$-CPA implies n-RPAT$_c$-CPA for any n.

Lemma 9.2 *If $\mathcal{AE}$ is secure w.r.t. 1-RPAT$_c$-CPA then it is secure w.r.t. n-RPAT$_c$-CPA.*

Proof: Let us consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against n-RPAT$_c$-CPA.

Using the reduction theorem 5.1, we split up the advantage between an advantage against $(n-1)$-RPAT$_c$-CPA and an advantage against 1-RPAT$_c$-CPA. We assume that adversary $\mathcal{A}$ accesses the left-right oracle F exactly once. Let (pk_1, sk_1) be a maximal key for $<$ (i.e. there exists a nonce N such that (pk_1, N) is maximal). The partition of θ is defined as follows: θ_1 contains key pair (pk_1, sk_1), any nonce N such that $(pk_1, N) \in E_<$. On the other hand, θ_2 contains the other informations from θ including the challenge bit. Oracle F_2 generates the encodings related to keys in θ_2 and F_1 those related to (pk_1, sk_1).

As (pk_1, sk_1) is maximal, there are no other key (pk_j, sk_j) such that for a nonce N in θ_1 and any nonce N', $(pk_1, N) < (pk_j, N')$. This is why nonce N is only used as the random coins of an encryption using pk_1.

F_1 can be separated into two layers G and H defined by:

$$H(\langle pat_0, pat_1\rangle, \theta_2, \theta_2') = \langle v(pat_{b_2}, \theta_2), v(pat_{b_2'}, \theta_2')\rangle$$

$$G(\langle pat_0, pat_1\rangle, b, \theta_1) = v(pat_b, \theta_1)$$

Where b_2 and b_2' are the challenge bits contained respectively in θ_2 and θ_2'.

Let pat_0 and pat_1 be two R-patterns such that $obs_a(pat_0) = obs_a(pat_1)$. Then both patterns are the concatenation of encodings and similar bit-strings. The call to F has to be simulated using F_1 and F_2. For this purpose, the valuation of their encodings is performed in a similar way as in $vrec$ except that F_1 and F_2 should only be called once. To achieve this, requests to F_1 and F_2 are stored in a single pattern as described for F_1 by function $vrec2$ which outputs a list of pairs of patterns:

$$\begin{aligned} vrec2(bs, bs) &= [\,] \\ vrec2(m_1.m_2, m_1'.m_2') &= vrec2(m_1, m_1').vrec2(m_2, m_2') \\ vrec2(\{m; N\}_k, \{m'; N\}_k) &= \langle \{m; N\}_k, \{m'; N\}_k\rangle \end{aligned}$$

Then this list of pairs $(\langle p_1, p_1'\rangle; ...; \langle p_n, p_n'\rangle)$ is transformed into the pair of lists $\langle p_1; ...; p_n, p_1'; ...; p_n'\rangle$ which is the argument given to F_1. Another function should perform the same operation for keys different from k. After submitting the results to oracle F_1 and F_2, it it easy to rebuild the output of F.

Note that patterns submitted to F_1 and F_2 are pairs of encodings. F_2 receives two well-formed patterns that have the same obs_a and this is the same thing for G (both receive a concatenation of some $\diamondsuit^N$).

As F_2 only depends on θ_2, our partition theorem applies, criterion $(\theta_2; F_2; V_2)$ is $(n-1)$-RPAT-CPA and $(\theta_1, b; G; V_b)$ is 1-RPAT-CPA. The partition theorem applies and gives that there exist two PRTMs $\mathcal{A}^o$ and $\mathcal{B}$ such that

$$|\mathbf{Adv}_{\mathcal{A}}^{n-RPAT}(\eta)| \leq 2|\mathbf{Adv}_{\mathcal{B}}^{1-RPAT}(\eta)| + |\mathbf{Adv}_{\mathcal{A}^o}^{(n-1)-RPAT}(\eta)|$$

A simple induction proves that as $\mathcal{AE}$ is secure against 1-RPAT$_c$-CPA, it is secure against n-RPAT$_c$-CPA for any integer n. ∎

Finally, we show that n-RPAT$_c$-CPA implies n-RPAT-CPA.

Lemma 9.3 *If $\mathcal{AE}$ is secure w.r.t. n-RPAT$_c$-CPA then it is secure w.r.t. n-RPAT-CPA.*

Proof: Let us consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against n-RPAT-CPA. As in step 1, an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is built such that $\mathcal{B}_1$ returns clean patterns. For this purpose, $\mathcal{B}_2$ is similar to $\mathcal{A}_2$. $\mathcal{B}_1$ executes $\mathcal{A}_1$ and computes $dec(pat_0, pat_1)$. Then it generates some keys and nonces and uses them for elements of dec on the answer of pat_0 and pat_1. The patterns remain well-formed and still have the same obs_a. $\mathcal{B}$ and $\mathcal{A}$ have the same advantage but $\mathcal{B}$ is an adversary against n-RPAT$_c$-CPA. As $\mathcal{AE}$ is secure against n-RPAT$_c$-CPA, it is also secure against n-RPAT-CPA. ■

Proposition 9.15 is a simple consequence of the above three lemmas. ■

Composition Result

Our main result states that given a specification, the advantage of an adversary against the concrete system is lower than the advantage of the abstract system and the advantage of another adversary against n-RPAT-CPA.

Theorem 9.1 *For each adversary $\mathcal{A}$, there exist two adversaries $\mathcal{A}^o$ and $\mathcal{B}$ such that*

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

Proof: This theorem is an application of the partition theorem 5.1. Let Δ_s be a specification. Let n be the maximal number of keys used by Δ_s. Then the experiment related to $obs_c \times obs_a$ can be reformulated as the following experiment:

- Θ is split up on two parts: Θ_1 generates the n pairs of keys (pk_i, sk_i) and l nonces n^i; Θ_2 generates the initial state s in S and the pattern $p = \Delta_s(s)$.
- We have two oracles: F_2 gives access to $obs_a(p)$, F_1 gives access to $v(p, \theta_1)$ which is $obs_c(p)$.
- V_2 verifies that the output b made by the adversary is equivalent to $s \in \phi$.

F_1 can be be cut in two layers. G corresponds to the left-right encryption algorithm for n-RPAT-CPA, $H(x, \theta_2, \theta_2')$ takes any argument as input x and outputs the pair $\langle p', p\rangle$ where p and p' are the patterns respectively contained in θ_2 and θ_2'.

It is now possible to apply the reduction theorem 5.1 to obtain that for each adversary $\mathcal{A}$, there exist two adversaries $\mathcal{A}^o$ and $\mathcal{B}$ such that

$$|\mathbf{Adv}_{\mathcal{A}}^{\gamma}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{\gamma_2}| + 2|\mathbf{Adv}_{\mathcal{B}}^{\gamma_1}|$$

Moreover, γ is equivalent to the criterion related to $obs_c \times obs_a$, γ_2 is equivalent to the one related to obs_a. Finally, $\gamma_1 = (b, \theta_1; G; \lambda x.x = b)$, G is only the left-right oracle, hence this criterion is the n-RPAT-CPA criterion except that there is no oracle to view the public keys. As this criterion is weaker than n-RPAT-CPA, it is possible to conclude that with a different machine $\mathcal{B}$ (but still of comparable complexity),

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

■

Using proposition 9.15, it is clear that if an encryption scheme is secure against IND-CPA, then it is secure against n-RPAT-CPA for any integer n. Therefor, we have

Corollary 9.1 *If the encryption scheme $\mathcal{AE}$ used in v is IND-CPA and obs_a brings negligible advantage to any adversary then obs_c brings negligible advantage to any adversary.*

Application: the Classical Abadi-Rogaway Result Using this result, it is possible to prove a slightly extended version of the seminal result of Abadi and Rogaway [AR00]. This result states that provided the used encryption scheme is IND-CPA indistinguishability in the formal (Dolev-Yao) model implies indistinguishability in the computational model. In fact, obfuscated patterns are close to patterns as introduced in [AR00]. The main difference is that our patterns explicitly

represent random coins. However, it is still possible to get exactly Abadi and Rogaway's result by assuming fresh distinct nonces for every encryption. If we consider messages with no encryption cycles, then the corresponding patterns (using fresh nonces) are well-formed. Moreover as nonces used for encryption are fresh, each $\diamondsuit_k^N$ has a different label, thus to test equality these labels are not considered. The Abadi-Rogaway theorem can be stated as an opacity problem. Let m_0 and m_1 be two well-formed patterns. There are two initial states in S: 0 and 1. Specification $\Delta_S(s)$ outputs m_s. Then m_0 and m_1 are *indistinguishable* if for any adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{obs_c}$ is negligible.

Proposition 9.16 *Let m_0 and m_1 be two well-formed patterns such that $obs_a(m_0) = obs_a(m_1)$. If the encryption scheme $\mathcal{AE}$ used in v is IND-CPA then m_0 and m_1 are indistinguishable.*

This result is immediate if we apply the above theorem: as obs_a returns the same result for the two patterns, $|\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}|$ is equal to zero. Hence as $|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$ is negligible, the advantage of $\mathcal{A}$ is also negligible and we get the desired result.

9.3.4 An Example, the Chaum Voting Scheme

To illustrate our results, we consider a slightly modified version of the electronic voting scheme proposed by David Chaum [Cha04] and improved by David Chaum, Peter Ryan and Steve Schneider [CRS04]. The main advantage of this scheme is that it is verifiable using an audit procedure which preserves opacity of the votes [GKK03], i.e. even after the audit procedure it is not possible to answer correctly to question: what did voter V vote? However, this paper still makes the perfect cryptography hypothesis, encryptions are considered as black-box and are not taken into account. We give here a proof of security for Chaum's voting scheme in a computational setting. For this purpose, we assume that the encryption scheme is IND-CPA and prove that then, security results still hold (but we may have to add some negligible terms representing brute force attacks against the encryption scheme as we only prove cryptographic opacity).

9.3.5 Description of the Chaum Voting Scheme

Let us briefly recall how the Chaum's voting scheme works. We omit some important pieces (mostly the visual aspect) that are not relevant in this context. The interested reader may consider reading [Cha04] or [CRS04] for details.

A vote session uses n trustees to guarantee the security of the procedure. Each trustee C_i has a public key pk_i and an associated secret key sk_i. The vote procedure works as follows: in a first step, each voter x chooses a vote value v_x in a finite set V of possible votes. Using this value v_x, the voting machine creates an onion by encrypting v_x with the public key of all the trustees. The resulting ballot $b_{1,x}$ is therefore:

$$b_{1,x} = \left\{...\{v_x; N^n\}_{pk_n}; ...; N^1\right\}_{pk_1}$$

More generally ballot b_i is defined by:

$$\begin{aligned} b_{i,x} &= \{b_{i+1,x}; N_i\}_{pk_i} \text{ for } i \in [1, n] \\ b_{n+1,x} &= v_x \end{aligned}$$

Of course, the different nonces N_i are fresh and are not used for anything else. The voting machine does not store anything but $b_{1,x}$, all the nonces values and vote v are forgotten immediately.

Then the voting machine sends all the different ballots $b_{1,x}$ to trustee C_1. Trustee C_1 publishes the values of the different ballots $b_{1,x}$. Then he decodes ballot $b_{1,x}$ using its secret key sk_1. Therefore, trustee C_1 obtains ballot $b_{2,x}$. However in order to ensure anonymity, $b_{2,x}$ cannot be published directly. Instead C_1 performs an *anonymizing mix*: C_1 performs a random permutation of the different ballots before publishing its output. The input of C_1 was composed of $v_x^1 = b_{1,x}$

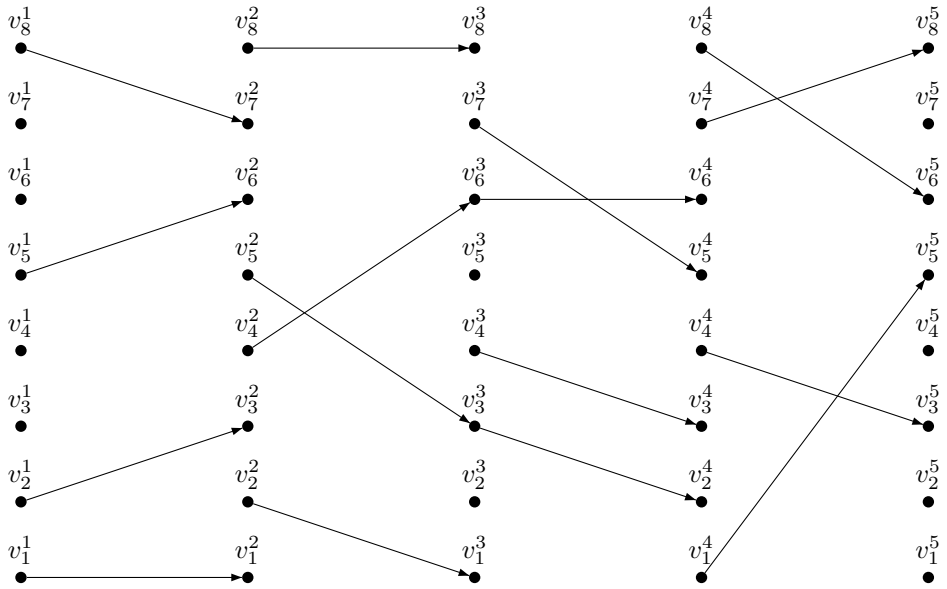


Figure 9.4: Audit for the first four Trustees

for all the possible voters x in X. Then the output of C_1 is $v_x^2 = b_{2,x\sigma}$ where σ is a random permutation over X.

Each trustee C_{i+1} takes as input the output of the previous trustee C_i. He decodes his layer then performs an anonymizing mix of all the votes. The resulting output list is published so that it could be used by the next trustee. All the intermediate lists are made public and the last list allows anyone to compute the result of the vote.

After the decoding phase, an audit process allows one to verify that trustees behave correctly with great probability. Hence each trustee C_i has to reveal the permutation σ it used for half the ballots. Thus he has to show for these ballots the link between the input ballot $b_{i,x}$ and the output $b_{i+1,x}$, the trustee also has to output nonce N^i. Therefore it is possible for anyone to check that the link is valid (it is supposed that the encryption algorithm allows the trustee to get this nonce). Verified ballots are not chosen randomly but as described in figure 9.4. The first set of verified ballots (for step 1) is chosen randomly. For step 2, verified ballots correspond to unconnected ballots w.r.t. step 1. For step 3, verified ballots are half unconnected ballots and half connected ones, the halves are chosen randomly. Finally, for step 4, verified ballots are unconnected ballots w.r.t. step 3. In the figure, v_j^i is the i^{th} ballot in the input of the j^{th} trustee and σ_j is the permutation chosen by this trustee. The set I_j consists of integers k such that the transition that reaches v_k^j is revealed.

Verification of the system The property we are interested in is opacity of the vote. However, it should be possible to generalize our results to more complex properties like the bound over variation distance given in [GKK03].

To simplify, let us suppose that there are two possible values for the vote: y and n. Then the set S of initial states contains all the vote distributions that give a fixed final result, i.e. for any element of S the number of voters that choose y is fixed, all the other variables are chosen at random (permutations, audit sets).

We study the opacity of property $\psi = (v_1^1 = y)$: are we able to deduce that the vote chosen by voter 1 is y? We want to prove that the audit information cannot bring any advantage to an attacker. This requires that for any observation o, $Pr[\psi|o] = Pr[o]$. Then, as $PrRand$ is given by the vote result, it is clear that it is impossible to guess the value of v_1 with better efficiency than

when answering the most probable vote with respect to the result. The specification of the system is pretty straightforward: Δ_s outputs the revealed permutations, the ballots lists and the nonces used to check the link for any $k \in I_j$. The output pattern is well-formed (there are no cycles for $<$, the form of the ballots gives $pk_n < pk_{n-1} < ... < pk_1$).

After applying obs_a, the abstract system gives information on the permutations and the final ballot line, indeed there are two cases for remaining encrypted ballots: they can be abstracted to $\diamondsuit_k^N$ or they can be linked by a permutation to a vote in the final (unencrypted) line and so these ballots are useless to the description because it would be possible from the rest of the description to rebuild them using the final vote and the revealed nonce. Let o be an observation in the formal world. Let p_y be the percentage of voter that choose y. Then a quick computation gives us that $Pr[v_1^1 = y|o] = p_y$. Let us detail this: there are two cases to consider. First case, the link starting from v_1^1 is revealed.

$$\begin{aligned} Pr[v_1^1 = y|o] &= Pr[v_{1\sigma_1}^2 = y|o] \\ &= \frac{2}{n}\sum_{i \notin I_3} Pr[v_i^3 = y|o] \\ &= \frac{1}{n}\sum_{i=1}^{n} Pr[v_i^4 = y|o] \\ &= p_y \end{aligned}$$

In the second case, a similar computation can be done.

$$\begin{aligned} Pr[v_1^1 = y|o] &= \frac{2}{n}\sum_{i \notin I_2} Pr[v_i^2 = y|o] \\ &= \frac{2}{n}\sum_{i \in I_3} Pr[v_i^3 = y|o \\ &= \frac{1}{n}\sum_{i=1}^{n} Pr[v_i^4 = y|o] \\ &= p_y \end{aligned}$$

This proves opacity of ψ in the symbolic world.

Security in the computational world is easy to obtain by applying our composition theorem: let $\mathcal{A}$ be a PRTM, then there exist $\mathcal{A}^o$ and $\mathcal{B}$ two PRTM such that:

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

The advantage related to obs_a is zero. Moreover, if we consider that the encryption scheme used is IND-CPA, then $\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}$ is negligible. Thus the advantage of $\mathcal{A}$ is negligible and we can conclude that the observable obs_c is safe for the cryptographic opacity of ψ.

Chapter 10

Conclusions

Contents

10.1 Achievements

The first achievement of this thesis is a formalization of security criteria. Using this formalism, we were able to represent classical security requirements such as semantic security or unforgeability. This formalism also allowed us to introduce various extensions of these classical requirements. The main interest of these variants is key dependability: using patterns, an adversary can ask for encryption of some secret information as long as he does not introduce any cycles. Patterns can be seen as a particular case of key-dependent messages [BRS01]. Even if patterns are less general, they are perfectly suited when attempting to link symbolic and computational cryptography. We also introduce criteria representing joint security of multiple cryptographic primitives. These criteria are useful when considering protocols that involve both asymmetric and symmetric encryption schemes for example.

Moreover, we proved that our new criteria are equivalent to classical ones. For this purpose, we formulated a criterion partition theorem. This theorem can be used to show safety for a criteria when assuming safety for some sub-criteria. For example, if an asymmetric encryption scheme and a symmetric encryption scheme are both secure (for semantic security and unforgeability), their combination is also secure. The main interest of this theorem is that it can be used without having to describe new adversaries. In fact, the necessary adversaries are embedded in the theorem's proof thus using this result is very simple. Using the theorem, we were able to prove that if all the cryptographic primitives are secure on their own, the cryptographic library combining these primitives is secure for a joint criterion.

The second achievement directly concerns the link between symbolic and computational views of cryptographic protocols. The main result is that if the cryptographic library is secure for our joint criterion, if the protocol is secure in the symbolic setting, then the implementation of this protocol (using the secure cryptographic library) is also secure in the computational setting. The symbolic view of the protocol is given by the classical Dolev-Yao model. Hence various properties can be checked automatically on the symbolic side, even when considering an unbounded number of sessions. Although our approach does not directly apply to complex properties, we were able to prove the soundness result for trace properties (e.g. authentication), weak secrecy of any message,

strong secrecy of nonces (SecNonce) and keys (SecKey). The SecNonce property implies that a nonce has to be indistinguishable from a random nonce after running the protocol. As this kind of requirement is false in general when considering keys, we introduced the SecKey property which states that the key still has to be secure after executing the protocol.

Finally, our third achievement consists of three possible extensions of the previous results: modular exponentiation, polynomial challenges and opacity.

The first extension is modular exponentiation. In this chapter, we studied a classical cryptographic primitive, modular exponentiation as used for example in the Diffie-Hellman protocol. The classical security requirements for this primitive are the computational and decisional Diffie-Hellman problems. We were able to prove that if the decisional Diffie-Hellman problem is hard then a generalization called the dynamic decisional Diffie-Hellman is also hard. We also gave an extended version of the Dolev-Yao model in order to tackle the specificities of exponentiation. Finally, using the dynamic Diffie-Hellman assumption, we were able to relate this modified symbolic model and the classical computational model in the passive setting. This result can be extended to the active setting by using Katz and Yung's protocol compiler [KY03].

In the second extension, our objective was to consider an unbounded number of sessions. For this purpose, we extended the criterion formalism in order to handle criteria where a polynomial number of challenges are generated. We also gave an adapted version of the criterion partition theorem. The main complexity when considering protocols involving a polynomial number of keys is adaptive corruption: it is impossible to guess with non-negligible probability which key is going to be corrupted by the adversary. Using the criterion partition theorem, we were able to prove that for asymmetric encryption, there are no problems of adaptive corruption when using semantic security. The adaptive corruption problem is closely linked to the classical selective decommitment problem. An interesting point is that the proof technique used for polynomial criteria can be used to prove that semantic security implies selective decommitment (this was an open question since 1986).

Finally, the third extension is a more complex security properties called opacity. The idea of opacity is to represent indistinguishability in the symbolic model. We defined several flavors of opacity and proved various decidability and undecidability results. Moreover, in order to consider the Chaum voting scheme, we introduced encryption randomness in the symbolic setting and a specific formalism where output of the message are patterns which can either be instantiated in the abstract symbolic model or in the computational model. We were able to prove soundness of this symbolic model in the passive case: outputs from the computational model do not bring significant information compared to output from the abstract symbolic model. This has been applied to give the first computational security proof for the Chaum voting scheme.

10.2 Future Work

Our results can be extended in several directions. First let us discuss the possible work directions when proving computational soundness of the Dolev-Yao model.

First it would be of interest to investigate whether the soundness result can be proved under weaker assumptions on the cryptographic primitives. Almost any work linking the symbolic and computational models for protocols in the active case uses semantic security against chosen-cipher-text attacks for encryption schemes. It is clear that the result is false when considering chosen-plain-text attacks. However, we can notice that some form of malleability is useless to the adversary, for example if the adversary is able to transform a cipher-text into another cipher-text but he does not have enough information on the new encoded message, then it might be impossible for the adversary to perform his attack. Hence it might be possible to find an intermediate security definition which is necessary and sufficient in order to prove computational soundness of the symbolic model.

Moreover, it would be significant to extend the soundness result to other cryptographic primitives. We achieved this for modular exponentiation and block chaining encryption in this document. However there are several other different primitives which are commonly used in real-life

protocols. For example, adding the XOR operator for protocols in the dynamic case is quite challenging, Baudet et al. considered the passive case in [BCK05], Backes and Pfitzmann proved an impossibility result in [BP05a] but this result only holds in their simulatability-based framework. Other primitives of interest are for example Message Authentication Codes (MAC), blind signature or encryption schemes with low-entropy passwords. In order to add a new primitive, one has to enhance the Dolev-Yao deduction rules in the symbolic setting, then one has to properly define computational security for this primitive. Finally, it might be possible to prove computational soundness of the enhanced Dolev-Yao model by assuming that the computational requirement is verified on the newly added primitive.

Another interesting line of work would be to give a machine checkable formalization of security criteria. Then the objective would be to make "formal" proofs of the link between criteria. Automatic proof of such links has been studied in [Bla05] but the proposed tool does not output the necessary information to build a complete formal proof. Early work has been made on the Generic Model and on the Random Oracle Model [BCT04] but these lacks the generality of our formalism for security criteria.

Concerning the three extensions proposed in this document, each of them has several possible follow-ups.

10.2.1 Modular Exponentiation

In the whole modular exponentiation chapter, we only considered exponentiations of "power-free" polynomials. Hence message $exp(x^2)$ was forbidden. This is linked to an open question formulated in [BDZ03]: if the decisional Diffie-Hellman problem is hard, then is it possible for an adversary to distinguish $(g^x, g^{x.x})$ from (g^x, g^r) with non-negligible probability (where x and r are two randomly sampled elements)? If the answer to this question is yes, then it might be possible to consider non-power-free messages. If this is not the case, we could either modify the symbolic model or add some computational hypotheses.

Moreover, we proved soundness of an extension of the Abadi-Rogaway logic for messages using modular exponentiation. An interesting result is the converse: completeness of the Abadi-Rogaway result. This completeness result has been proven by Micciancio and Warinschi in [MW04b] in the case of symmetric encryption. For this purpose, Micciancio and Warinschi had to add an hypothesis on the encryption scheme. We believe that with this hypothesis, a similar result holds when adding modular exponentiation.

In [MP05], Micciancio and Panjwani proposed an adaptive model extending the Abadi-Rogaway result. Their main soundness result (which we extended in chapter 8) still only considers symmetric encryption hence it seems feasible to add modular exponentiation in this model. In particular, this can be applied in order to represent group protocols using Diffie-Hellman exponentiation.

Finally, we put the hypothesis that the order of the group is prime. This might be false in some specific groups like the RSA group. When the order of the group is prime, g^x and g^{2x} are indistinguishable, however if 2 divides the order of the group, this is not necessarily the case (this can still be the case as the group order might be hard to compute). Thus, it can be interesting to modify our symbolic model so as to consider non-prime order groups. In such groups, g^{2x} is indistinguishable from g^{2y} but not from g^y, hence we have to modify the 3DH criterion and the renaming used to define the symbolic equivalence.

10.2.2 Polynomial Challenges

We proposed a new security criterion P-AC-PAT-IND-CPA. This criterion handles adaptive corruption as the adversary may ask for revelation of one of the $P(\eta)$ challenge keys if this key has not been used to hide some information. Using the polynomial version of the partition theorem, we proved that this new criterion is implied by classical semantic security.

However, this new criterion has an obvious limitation: patterns can be used with the left-right encryption oracles but cannot be used with the unary encryption oracles. Thus the adversary

cannot ask for the encryption of some key k under k' then for revelation of k'. Intuitively, if k and k' have never been used to hide important information, revealing k' (and thus k) should not be a problem. The proof technique used by our partition theorem must not work here: as k' is used to encrypt k, the theorem works by encrypting either k or another randomly sampled key with k'. Hence in the proof of the theorem, when using patterns, we have to use the left-right encryption oracle even if there is a single argument. It is not clear to us whether semantic security is enough to ensure a generalized version of the adaptive corruption criterion where one could ask for encryption of keys then for revelation of the key used to perform the encryption.

Another interesting point which seems more complicated than adaptive corruption is the case of dynamic cycles. In this whole document, cycles are fixed by a static ordering among keys. Thus it is impossible with these results to prove computational soundness of the symbolic model for protocols with an unbounded number of session where key cycles are forbidden but the ordering among keys can be adaptively chosen by the adversary. A way to tackle this problem is to introduce a new criterion based on semantic security: a polynomial number of keys are generated, the arguments given to left-right encryption oracles are still patterns. The acyclicity restriction is not fixed at the beginning anymore, the only requirement is that the adversary when issuing a query does not produce a key cycle with previous queries. Then an open question is whether this new criterion is implied by classical semantic security.

10.2.3 Opacity

Opacity has mostly been studied in the passive setting. Hence it would be interesting to generalize all the results in the case of an active adversary. In particular, the link between the symbolic and computational models only holds in the passive setting, an extension to the active setting would definitely be a plus.

A current restriction of opacity in the symbolic setting is that it only tells you that an adversary cannot deduce *for sure* that a property is verified. The adversary cannot put probabilities on the likelihood of ϕ and $\neg\phi$. A potential further line of research is therefore to consider probabilistic opacity even in symbolic setting. For this purpose, it is necessary to introduce probabilities on the initial parameters or to consider probabilistic LTSs. Then the computational soundness result would state that the difference of probability for opacity is only negligible if the cryptographic primitives used in the implementation are secure.

Bibliography

[ABW06] M. Abadi, M. Baudet, and B. Warinschi. Guessing attacks and the computational soundness of static equivalence. In *Proceeding of the Ninth International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2006)*. Springer, 2006. 1.3.3

[AG99] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1), 1999. An extended abstract appeared in the *Proceedings of the Fourth ACM Conference on Computer and Communications Security (Zürich, April 1997)*. 1.3.1

[AJ01] M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proceedings of the Fourth International Symposium on Theoretical Aspects of Computer Software (TACS 2001)*. Springer-Verlag, 2001. 1.3.3, 7.4.2

[AL00] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proceedings of the International Conference on Concurrency Theory (CONCUR 2000)*, 2000. 2.5.1, 2.6, 2.6.1

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, 2000. Springer-Verlag. 1.3.3, 1.4.2, 4.5.1, 6.2.1, 7, 7.3, 7.3.2, 7.3.3, 7.4.2, 7.4.2, 2, 7.4.2, 8.4.1, 8.4.1, 8.4.1, 9.3.3

[AW05] M. Abadi and B. Warinschi. Password-based encryption analyzed. In *Proceedings of the Thirty Second International Colloquium on Automata, Languages and Programming (ICALP 2005)*. Springer, 2005. 1.3.3

[BAF05] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Proceedings of the Twentieth IEEE Symposium on Logic in Computer Science (LICS 2005)*. IEEE Computer Society, 2005. 8.4.2

[BAN96] M. Burrows, M. Abadi, and R. Needham. A logic of authentication, from proceedings of the royal society, volume 426, number 1871, 1989. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996*. 1996. 2.2

[Bar84] H. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier Science Publishers B.V., 1984. 3.3.2

[BB03] M. Boreale and M. Buscemi. Symbolic analysis of crypto-protocols based on modular exponentiation. In *Proceedings of Mathematical Foundations of Computer Science (MFCS 2003)*, 2003. 7

[BBM00] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Proceedings of Advances in Cryptology (EUROCRYPT 2000)*, pages 259–274, 2000. 3.4.1, 3.4.1, 8

[BBS03] M. Bellare, A. Boldyreva, and J. Staddon. Randomness re-use in multi-recipient encryption schemes. In *Proceedings of the Public Key Cryptography Conference (PKC 2003)*. Springer-Verlag, 2003. 9.3.3

[BCK05] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proceedings of the Thirty second International Colloquium on Automata, Languages and Programming (ICALP 2005)*. Springer, 2005. 10.2

[BCP02] E. Bresson, O. Chevassut, and D. Pointcheval. The group Diffie-Hellman problems. In *Proceedings of Selected Areas in Cryptography (SAC 2002)*. Springer-Verlag, 2002. 7, 7.2.1

[BCP03] E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *Proceedings of the Ninth International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2003)*. Springer-Verlag, 2003. 9.3.3

[BCPQ01] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS 2001)*, 2001. 7, 7.4.3

[BCT04] G. Barthe, J. Cederquist, and S. Tarento. A Machine-Checked Formalization of the Generic Model and the Random Oracle Model. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2004)*. Springer, 2004. 10.2

[BD94] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system (extended abstract). In *EUROCRYPT*, pages 275–286, 1994. 7, 7.2.1, 7.4.3

[BD05] M. Burmester and Y. Desmedt. A secure and scalable group key exchange system. *Inf. Process. Lett.*, 94(3):137–143, 2005. 7.4.3, 7.4.3

[BDJR97] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Prooceding of the Eighth Annual Symposium on Foundations of Computer Science (FOCS 1997)*. IEEE, Computer Society Press, 1997. 3.5, 3.5.1

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Proceedings of the Eighteenth Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 1998)*. Springer-Verlag, 1998. 3.5, 3.5.2, 3.5.2

[BDZ03] F. Bao, R. Deng, and H. Zhu. Variations of Diffie-Hellman problem. In *Proceedings of the Fifth Conference on Information and Communications Security (ICIS 2003)*, pages 301–312, 2003. 7.2.1, 7.2.1, 10.2.1

[BG01] M. Bellare and S. Goldwasser. Lecture notes on cryptography, 2001. Course notes of summer course Cryptography and Computer Security at MIT, 1996–2001. 1.2.1

[BGH+93] R. Bird, I. S. Gopal, A. Herzberg, Ph. A. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 1993. 7

[BKMR05] J. Bryans, M. Koutny, L. Mazaré, and P. Ryan. Opacity generalised to transition systems. In *Proceedings of The Second Workshop on Formal Aspects of Security and Trust (FAST 2004)*. Springer-Verlag, 2005. 9

[BKR] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In *Proceedings of the Fourteenth Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 1994)*. Springer-Verlag. 1.3.2

[BKR04a] J. Bryans, M. Koutny, and P. Ryan. Modelling dynamic opacity using Petri nets with silent actions. In *Proceedings of The Second Workshop on Formal Aspects of Security and Trust (FAST 2004)*. Kluwer Academic Press, 2004. 9, 9.1.1

[BKR04b] J. Bryans, M. Koutny, and P. Ryan. Modelling opacity using Petri nets. In *Proceedings of WISP*, 2004. 9, 9.1.1, 9.2.1, 9.2.1

[Bla01] B. Blanchet. Abstracting cryptographic protocols by Prolog rules. In *Proocedings of the Eighth International Static Analysis Symposium (SAS 2001)*. Springer-Verlag, 2001. 1.3.2

[Bla05] B. Blanchet. A computationally sound mechanized prover for security protocols. Cryptology ePrint Archive, Report 2005/401, 2005. http://eprint.iacr.org/. 1.3.2, 10.2

[BLP03a] L. Bozga, Y. Lakhnech, and M. Périn. Abstract interpretation for secrecy using patterns. In *Proceedings of the Ninth International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*. Springer-Verlag, 2003. 2

[BLP03b] L. Bozga, Y. Lakhnech, and M. Périn. Hermes: An automatic tool for verification of secrecy in security protocols. In *Proceedings of the Fifteenth International Conference on Computer Aided Verification (CAV 2003)*. Springer-Verlag, 2003. 1.3.2, 8.4.2

[BN00] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Proceedings of the sixth International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2000)*, pages 531–545, 2000. 1.3.3, 3.4.3, 4.5.3

[BP04] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proceedings of the Seventeenth IEEE Computer Security Foundations Workshop (CSFW 2004)*. IEEE Computer Society, 2004. 1.3.3

[BP05a] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style xor. In *Proceedings of the Tenth European Symposium on Research in Computer Security (ESORICS 2005)*. Springer, 2005. 10.2

[BP05b] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proceedings of the IEEE Symposium on Security and Privacy (SP 2005)*. IEEE Computer Society, 2005. 1.3.3

[BPW03a] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proceedings of the Tenth ACM conference on Computer and Communication Security*, pages 220–230. ACM Press, 2003. 1.3.3, 6.3.2

[BPW03b] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In Springer-Verlag, editor, *Proceedings of the Eighth European Symposium on Research in Computer Security (ESORICS 2003)*, volume 2808 of *LNCS*, 2003. 1.3.3, 6.2.4

[BPW04] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In Springer-Verlag, editor, *Proceedings of the First Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *LNCS*, pages 271–290, 2004. 1.3.3

[BPW06] M. Backes, B. Pfitzmann, and M. Waidner. Limits of the reactive simulatability/uc of Dolev-Yao models with hashes. Cryptology ePrint Archive, Report 2006/068, 2006. http://eprint.iacr.org/. 1.3.3

[BR93] M. Bellare and Ph. Rogaway. Entity authentication and key distribution. In *Proceedings of the Thirteenth Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 1993)*, 1993. 7.4.3

[BR03] M. Bellare and P. Rogaway. *Introduction to Modern Cryptography*. 2003. 1.2.1, 6.2.5

[BRS01] J. Black, P. Rogaway, and T. Shrimpton. Encryption scheme security in the presence of key-dependent messages, 2001. 4.5.1, 7.4.2, 10.1

[Can97] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Proceedings of the Seventeenth International Cryptology Conference (CRYPTO 1997)*. Springer-Verlag, 1997. 6.2.5

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the Forty second IEEE symposium on Foundations of Computer Science (FOCS 2001)*. IEEE Computer Society, 2001. 1.3.3

[Can04] R. Canetti. Universally composable signature, certification, and authentication. In *Proceedings of the Seventeenth IEEE Computer Security Foundations Workshop (CSFW 2004)*. IEEE Computer Society, 2004. 1.3.3

[CBH05] K. Choo, C. Boyd, and Y. Hitchcock. Errors in computational complexity proofs for protocols. In *Proceedings of the Eleventh International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2005)*. Springer-Verlag, 2005. 1.3.2

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth ACM symposium of Programming Languages*. ACM Press, 1977. 9.2.2

[CC92] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computing*, 1992. 9.2.2

[CFGN96] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth annual ACM Symposium on Theory of Computing (SToC 1996)*. ACM Press, 1996. 1.3.3, 1.4.2, 8

[CH04] R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004. 6.3.2

[Cha04] D. Chaum. E-voting: Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy*, 2004. 9.3.4, 9.3.5

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997. 1.3.2, 1.3.3, 2.2, 6.2.4

[CJM98] E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *IFIP Working Conference on Programming Concepts and Methods*, 1998. 2

[CK01] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 2001)*. Springer-Verlag, 2001. 1.3.3

[CKRT03a] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of the Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 261–270. IEEE, Computer Society Press, 2003. 2.5.1, 2.6.1, 2.6.3, 6.7, 7, 7

[CKRT03b] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of the Twenty-Third Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003)*. Springer-Verlag, 2003. 2.6.1, 7.3.1

[CLC02] H. Comon-Lundh. and V. Cortier. Security properties: Two agents are sufficient. Technical report, Laboratoire Spécification et Vérification (LSV), ENS de Cachan, 2002. 2

[CLC03] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proceedings of the Fourteenth International Conference on Rewriting Techniques and Applications (RTA 2003)*. Springer-Verlag, 2003. 2

[CLS03] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of the Eighteenth Annual IEEE symposium on Logic In Computer Science (LICS 2003)*. IEEE Computer Society Press, 2003. 2.5.1, 2.6, 2.6.1

[Cor03] V. Cortier. A guide for SECURIFY. Technical Report 13, projet RNTL EVA, 2003. 8.4.2

[CRS04] D. Chaum, P. Ryan, and S. Schneider. A practical, voter-verifiable election scheme. Technical Report 880, University of Newcastle upon Tyne, School of Computing Science, 2004. 9.3.4, 9.3.5

[CS02] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 2002. 1.3.2

[CW05] V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *Proceeding of the Fourtennth European Symposium on Programming (ESOP 2005)*, pages 157–171. Springer-Verlag, 2005. 1.3.1, 1.3.3, 1.3.3, 1, 1.3.3, 6.1.3, 6.2, 6.3.2, 6.3.2, 8.4.2, 9.3.3

[DDM+05] A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proceedings of Thirty Second International Colloquium on Automata, Languages and Programming (ICALP 2005)*, 2005. 7

[DDMP03] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of the Sixteenth Computer Security Foundations Workshop (CSFW 2003)*. IEEE Computer Society, 2003. 7

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proceedings of the Twenty-Third Symposium on Theory of Computing (STOC 1991)*. ACM Press, 1991. 3.5.2

[DM00] G. Denker and J. Millen. The CAPSL integrated protocol environment. Technical Report SRI-CSL-2000-02, Computer Science Laboratory, SRI International, 2000. 2

[DNRS99] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer. Magic functions. In *Proceedings of the Fortieth IEEE Symposium on Foundations of Computer Science (FoCS 1999)*, 1999. 1.3.3, 8, 8.4.1

[DS81] D. Denning and G. Sacco. Timestamps in key distribution systems. *Communications of the ACM*, 1981. 6.1.2

[DvOW92] W. Diffie, P. C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 1992. 7

[DY83] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983. 1.2.3, 1.3.1, 1.3.2, 2, 2.4, 7.3.1

[FG94a] R. Focardi and R. Gorrieri. A classification of security properties for process algebra. *Journal of Computer Security*, 1994. 1.3.1

[FG94b] R. Focardi and R. Gorrieri. A taxonomy of trace-based security properties for CCS. In *Proceedings of the Computer Security Foundations Workshop Workshop (CSFW 1994)*. IEEE, 1994. 1.3.1, 9.1.3

[FG01] R. Focardi and R. Gorrieri. Classification of security properties (part i: Information flow). In *FOSAD '00: Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design*. Springer-Verlag, 2001. 9.1.3

[Fin93] A. Finkel. The minimal coverability graph for Petri nets. *Lecture Notes in Computer Science; Advances in Petri Nets 1993*, 1993. 9.2.2

[FOPS01] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 2001)*. Springer-Verlag, 2001. 3.4.1

[GKK03] M. Gomukiewicz, M. Klonowski, and M. Kutyowski. Rapid mixing and security of chaum's visual electronic voting. In Springer-Verlag, editor, *Proceedings of the Eighth European Symposium on Research in Computer Security (ESORICS 2003)*, 2003. 9.3.4, 9.3.5

[GL00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of the Fifteenth International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA 2000)*. Springer-Verlag, 2000. 1.3.2, 2

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 1984. 1.3.2, 1.3.3, 3.4.1, 7.4

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988. 3.4.2, 3.4.2

[GS05] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *Proceeding of the Third ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE 2005)*, 2005. 7

[Her03] J. Herzog. The Diffie-Hellman key-agreement scheme in the strand-space model. In *Proceedings of the Sixteenth Computer Security Foundations Workshop (CSFW 2003)*. IEEE Computer Society, 2003. 7

[Her04] J. Herzog. *Computational soundness for standard assumptions of formal cryptography*. PhD thesis, Massachusetts Institute of Technology, 2004. 7

[IT94] ITU-T. Recommendation Z.120. message sequence charts. Technical Report Z-120, International Telecommunication Union – Standardization Sector, Genève, 1994. 1.2.2

[Jan06] R. Janvier. *Certification de Protocoles Cryptographiques*. PhD thesis, Université Joseph Fourier - VERIMAG, 2006. 1.3.3, 1.3.3, 6.2.1, 6.2.4, 6.2.5, 6.3.1, 6.3.2

[JE96] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In *Automata, Languages and Programming*, 1996. 9.2.1

[JLM05a] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proceedings of the Fourteenth European Symposium on Programming (ESOP 2005)*, pages 172–185. Springer-Verlag, 2005. 1.3.3, 4.2, 8.4.2

[JLM05b] R. Janvier, Y. Lakhnech, and L. Mazaré. (de)compositions of cryptographic schemes and their applications to protocols. Cryptology ePrint Archive, Report 2005/020, 2005. http://eprint.iacr.org/. 1.3.3

[KY03] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *Proceedings of the Twenty-Third Annual International Cryptology Conference (CRYPTO 2003)*. Springer, 2003. 1.3.3, 7, 7, 7.4.3, 10.1

[Lau04] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 71–85, 2004. 1.3.3

[Low95] G. Lowe. An attack on the Needham-Schroeder public-key authentification protocol. *Information Processing Letters*, 1995. 1.2.3, 1.3.2, 2.2

[Low96] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In *Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1996)*. Springer-Verlag, 1996. 1.3.2, 2.2

[Low97a] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proceeding of the Tenth IEEE Computer Security Foundations Workshop (CSFW 1997)*. IEEE, 1997. 2

[Low97b] G. Lowe. A hierarchy of authentication specifications. In *Proceeding of the Tenth IEEE Computer Security Foundations Workshop (CSFW 1997)*. IEEE Computer Society Press, 1997. 1.3.1

[Maz04a] L. Mazaré. Satisfiability of Dolev-Yao constraints. In *Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2004)*. Elsevier, 2004. 2.6

[Maz04b] L. Mazaré. Using unification for opacity properties. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS 2004)*, 2004. 9

[McA93] D. McAllester. Automatic recognition of tractability in inference relations. *Journal of the ACM*, 1993. 2.6.3

[Mea00] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *Proceedings of the Thirteenth Computer Security Foundations Workshop (CSFW 2000)*. IEEE Computer Society, July 2000. 1.3.2

[Mil95] R. Milner. *Communication and concurrency*. Prentice Hall International (UK) Ltd., 1995. 1.3.1

[MP05] D. Micciancio and S. Panjwani. Adaptive security of symbolic encryption. In *Proceedings of the Theory of cryptography conference (TCC 2005)*. Springer-Verlag, 2005. 1.3.3, 7.3.2, 8.4, 8.4.1, 8.4.1, 8.4.1, 8.4.1, 8.4.1, 8.4.1, 10.2.1

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS 2001)*, 2001. 2.6

[MS03] J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proceedings of the Sixteenth Computer Security Foundations Workshop (CSFW 2003)*, 2003. 7, 7.3.1

[MvOV96] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. 1.2.1

[MW04a] D. Micciancio and B. Warinschi. Completeness theorems for the abadi-rogaway logic of encrypted expressions. *Journal of Computer Security*, 2004. 1.3.3

[MW04b] D. Micciancio and B. Warinschi. Completeness theorems for the abadi-rogaway logic of encrypted expressions. *Journal of Computer Security*, 2004. Preliminary version in WITS 2002. 7.4.2, 10.2.1

[MW04c] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings of the Theory of Cryptography Conference (TCC 2004)*, pages 133–151. Springer, 2004. 1.3.3, 1.4, 1.4.1, 6.1.3, 6.2, 6.2.2, 6.3.1, 8.4.2

[NS78] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 1978. 1.3.2, 2.2, 6.1.2

[O'H90] C. O'Halloran. A calculus of information flow. In Springer-Verlag, editor, *Proceedings of the European Symposium on Research in Computer Security (ESORICS 1990)*, 1990. 9.1.3

[Pet81] J. Peterson. *Petri Net Theory and The Modeling of Systems*. Prentice Hall, Inc., 1981. 9.2.1

[PQ01] O. Pereira and J. Quisquater. A security analysis of the Cliques protocols suites. In *Proceeding of the Fourteenth IEEE Computer Security Foundations Workshop (CSFW 2001)*, 2001. 7, 7

[PQ03] O. Pereira and J. Quisquater. Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 2003. 7

[PQ04] O. Pereira and J. Quisquater. Generic insecurity of Cliques-type authenticated group key agreement protocols. In *Proceedings of the Seventeenth Computer Security Foundations Workshop (CSFW 2004)*, 2004. 7, 7

[PSW00] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. In *Proceedings of the Workshop on Secure Architectures and Information Flow*, 2000. 1.3.3

[PTE04] P. Periorellis, C. Townson, and P. English. Structural concepts for trust, contract and security management for a virtual chemical engineering. Technical Report 854, University of Newcastle upon Tyne, School of Computing Science, Jul 2004. 9.2.4

[Rei98] Lectures on Petri nets. W.Reisig and G.Rozenberg (Eds.) LNCS 1491 & 1492, 1998. 9.2, 9.2.1, 3

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceeding of the Fourteenth IEEE Computer Security Foundations Workshop (CSFW 2001)*, 2001. 1.3.2, 2.5.1, 2.6.1, 2.6.3

[Rus03] M. Rusinowitch. Automated analysis of security protocols. *Electronic Notes in Theoretical Computer Science*, 2003. 8.4.2

[Rya01] P. Ryan. Mathematical models of computer security. In *Foundations of Security Analysis and Design: Tutorial Lectures*. Springer-Verlag, 2001. 2, 9.1.3

[Sch94] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 1994. 1.2.1

[Sho99] V. Shoup. On formal models for secure key exchange (version 4). Technical report, 1999 revision of IBM Research Report RZ 3120 (April 1999), 1999. 3.3.2

[Sho04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs, 2004. 3.5

[Son99] D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of the Twelfth IEEE Computer Security Foundations Workshop (CSFW 1999)*. IEEE, 1999. 1.3.2

[SS96] S. Schneider and A. Sidiropoulos. CSP and anonymity. In Springer-Verlag, editor, *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS 1996)*, 1996. 9.1.2

[STW96] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the ACM Conference on Computer and Communications Security*, 1996. 7, 7.2.1

[THG98] J. Thayer, J. Herzog, and J. Guttman. Honest Ideals on Strand Spaces. In *Proceedings of the Eleventh Computer Security Foundations Workshop (CSFW 1998)*. IEEE Computer Society, 1998. 7

[THG99] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 1999. 1.3.3

[tin04] TIme petri Net Analyzer. http://www.laas.fr/tina/, 2004. 9.2.5

[War03] B. Warinschi. A computational analysis of the Needham-Schroeder(-Lowe) protocol. In *Proceedings of the Sixteenth Computer Security Foundations Workshop (CSFW 2003)*, pages 248–262. ACM Press, 2003. 1.3.3, 6.1.3, 6.2, 6.2.2