

Formal Certification of Code-Based Cryptographic Proofs

Gilles Barthe¹

Benjamin Grégoire² Santiago Zanella Beguelin¹

Daniel Hedin¹ Sylvain Heraud² Federico Olmedo¹

¹ IMDEA Software, Madrid, Spain

² INRIA Sophia Antipolis - Méditerranée, France

Thanks to Yassine Lakhnech and Christine Paulin.

Also to David Nowak and Laurent Théry.

Many potential sources of failures using cryptography

- applications may make an improper use of cryptography
- implementations may introduce weaknesses
- protocols may be logically flawed
- primitives may be insecure

Formal methods help detect failures in using cryptography

- Program analyses can guarantee that cryptography is used correctly, or that implementations preserve security
- Symbolic methods help discovering flaws in protocols

Can we build tools to reason about primitives?

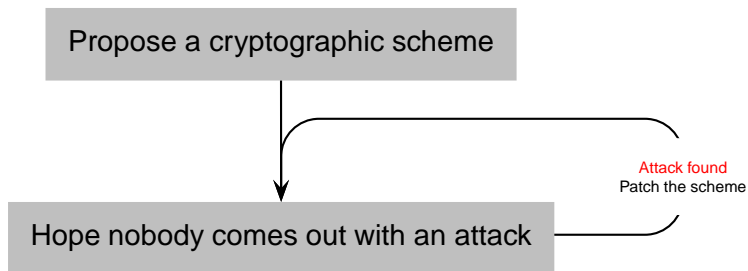
Cryptanalysis-driven security

Propose a cryptographic scheme

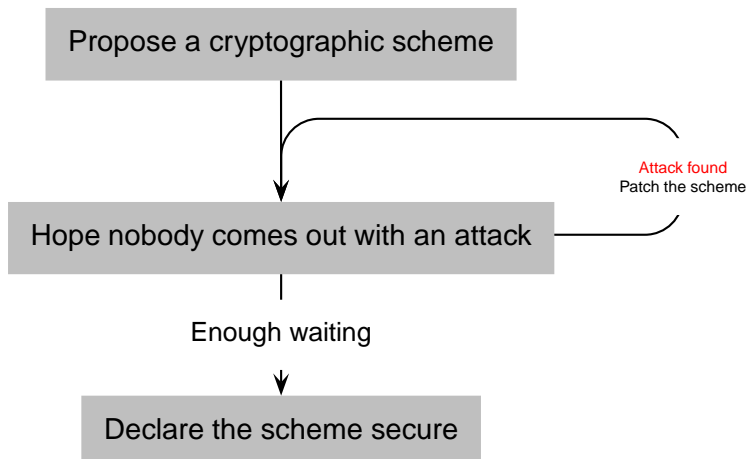


Hope nobody comes out with an attack

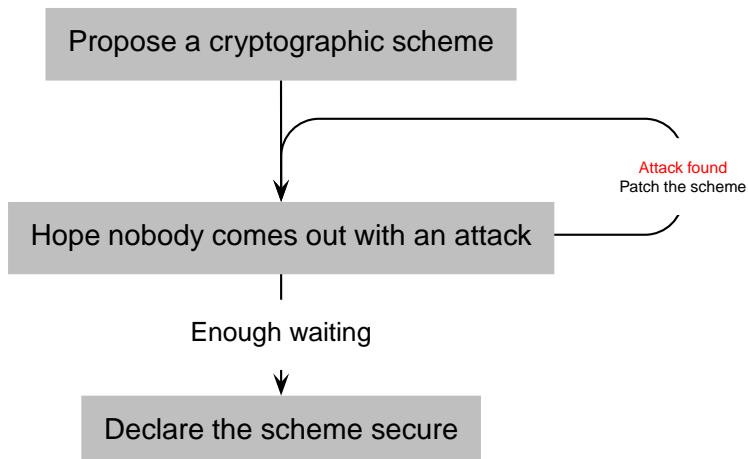
Cryptanalysis-driven security



Cryptanalysis-driven security

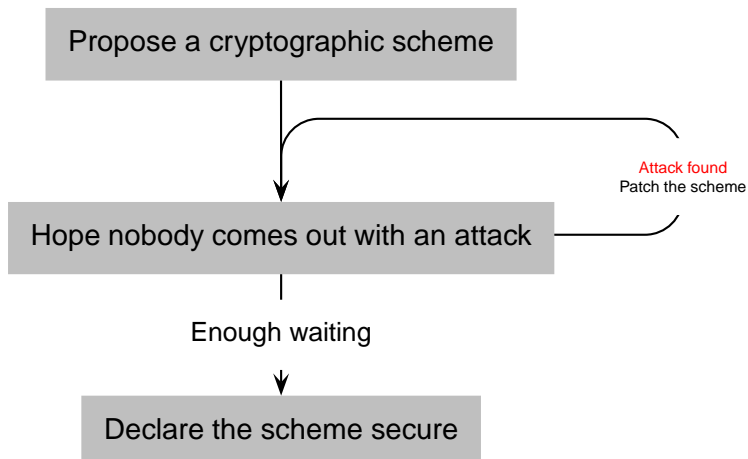


Cryptanalysis-driven security



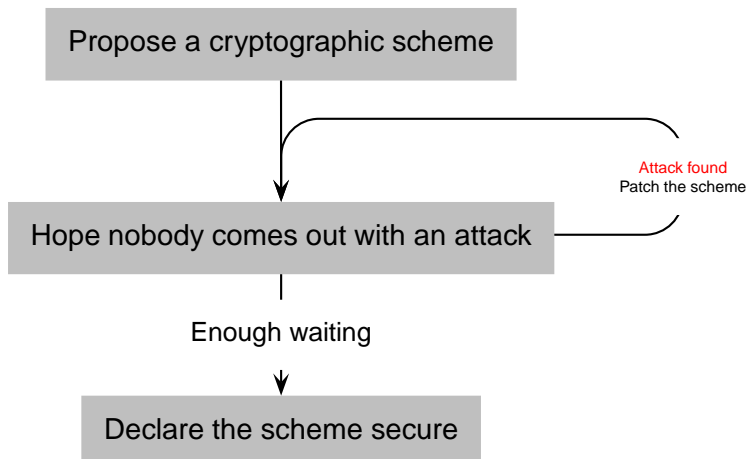
How much time is *enough*?

Cryptanalysis-driven security



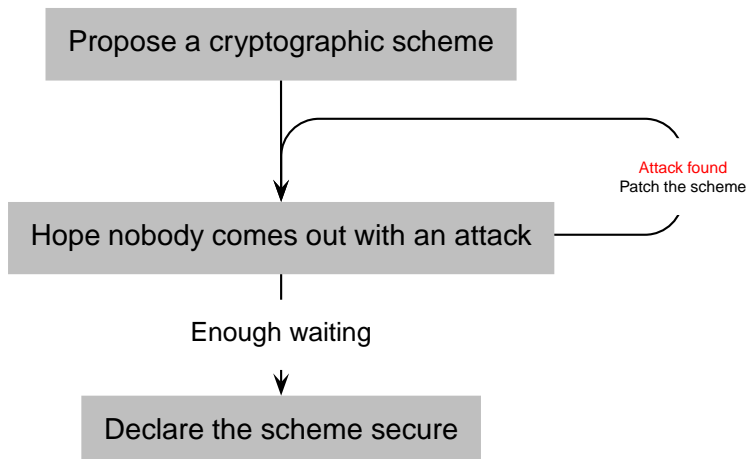
6 months, 1 year, 2 years?

Cryptanalysis-driven security



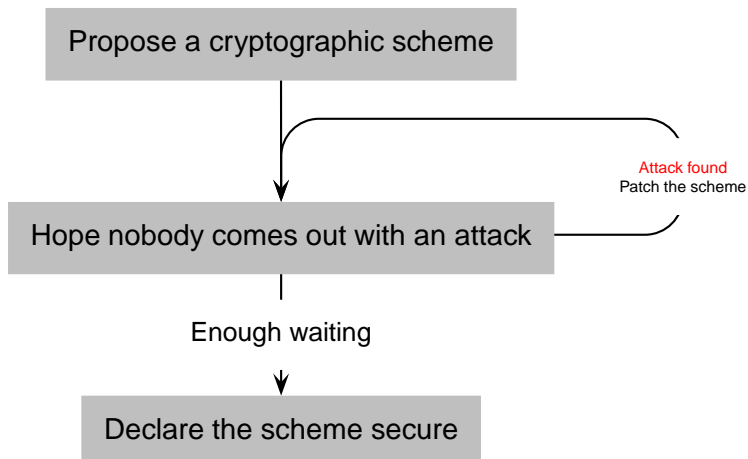
It took **5 years** to break the Merkle-Hellman cryptosystem

Cryptanalysis-driven security



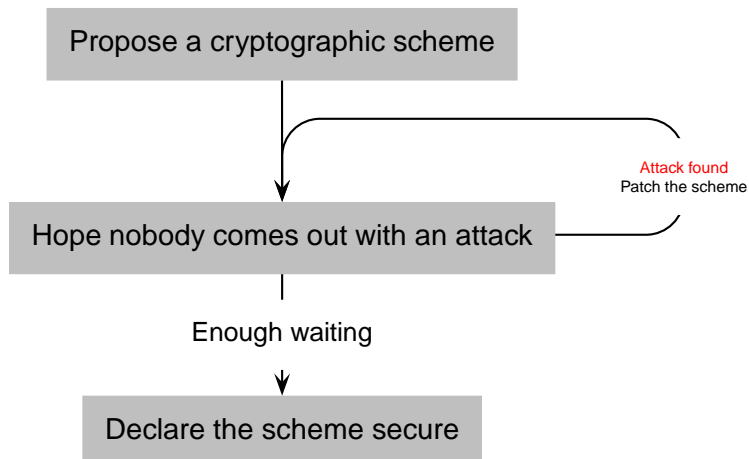
Ok, let's say **7 years** to be on the safe side

Cryptanalysis-driven security



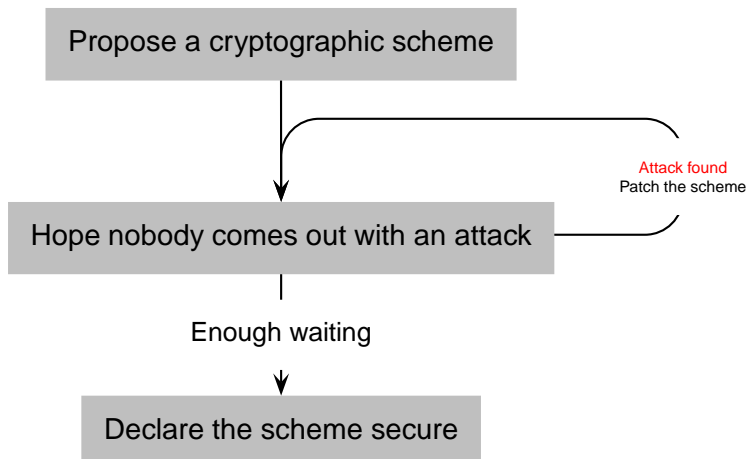
It took **10 years** to break the Chor-Rivest cryptosystem

Cryptanalysis-driven security



Can't we do better?

Cryptanalysis-driven security



Yes, we can!

Provable security

A mathematical approach to correctness

Theorem

IF the security assumptions hold
THEN the scheme is secure against adversary \mathcal{A}

- Security assumptions are explicit and (ideally) standard
- Goal is clearly stated and (ideally) standard
- Adversarial model is well-defined and (usually) standard
- Proof is rigorous

Provable security

A mathematical approach to correctness

Theorem

IF the security assumptions hold
THEN the scheme is secure against adversary \mathcal{A}

- Security assumptions are explicit and (ideally) standard
- Goal is clearly stated and (ideally) standard
- Adversarial model is well-defined and (usually) standard
- Proof is rigorous

Provable security

A mathematical approach to correctness

Theorem

IF the security assumptions hold
THEN the scheme is secure against adversary \mathcal{A}

- Security assumptions are explicit and (ideally) standard
- Goal is clearly stated and (ideally) standard
- Adversarial model is well-defined and (usually) standard
- Proof is rigorous

Typical exact security statement

FOR ALL adversary \mathcal{A} that can break the scheme
THERE EXISTS an adversary \mathcal{B} that can break some security assumption with *little extra effort*

$$\Pr[\mathcal{A} \text{ breaks scheme in time } t] \leq \Pr[\mathcal{B} \text{ breaks assumption in time } t + \Delta] + \epsilon$$

where Δ and ϵ depend on the number of oracles queries by \mathcal{A}

Proofs are constructive: \mathcal{B} uses \mathcal{A} as a subroutine

Provable security

Prove a cryptographic scheme secure



Hope that nobody finds a bug in the proof

Provable security

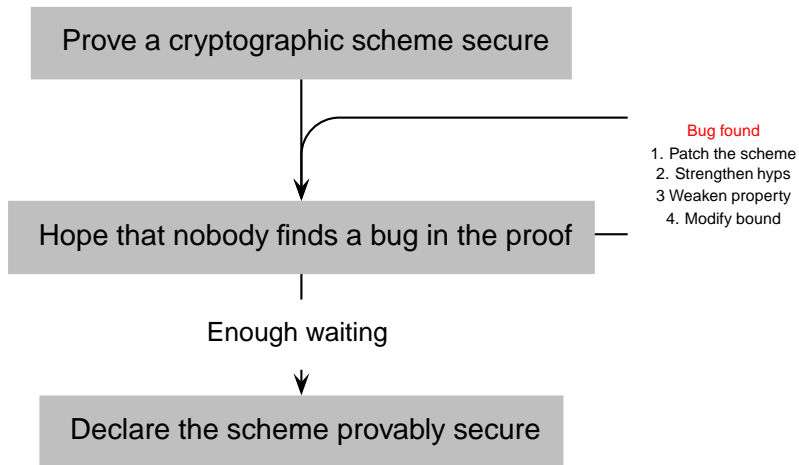
Prove a cryptographic scheme secure

Hope that nobody finds a bug in the proof

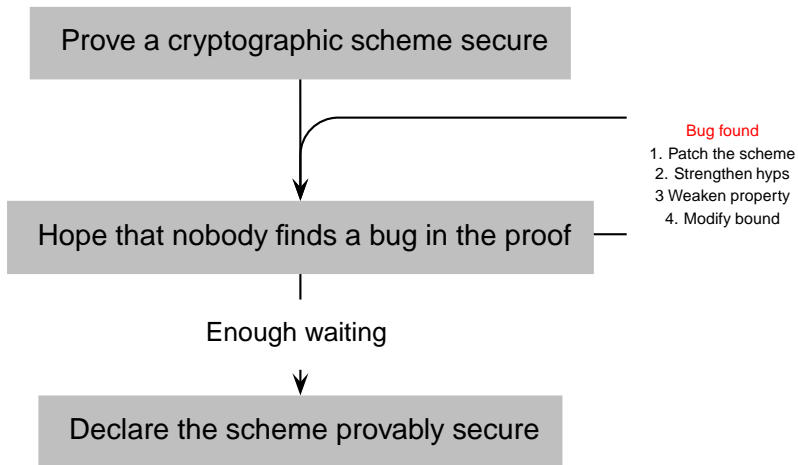
Bug found

1. Patch the scheme
2. Strengthen hyps
3. Weaken property
4. Modify bound

Provable security

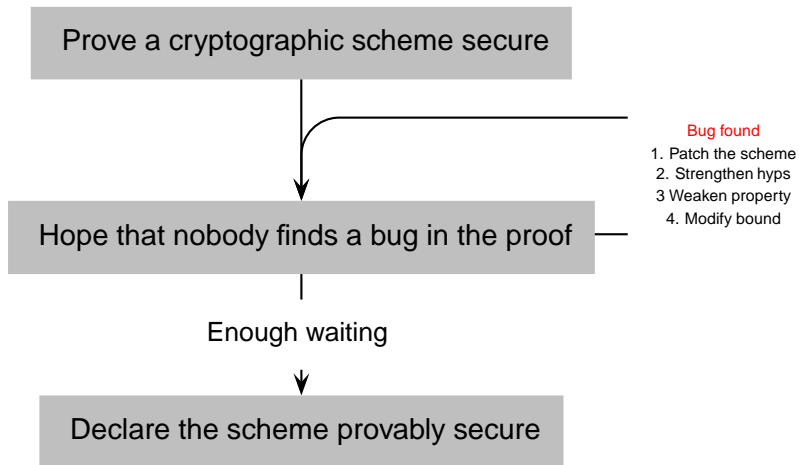


Provable security



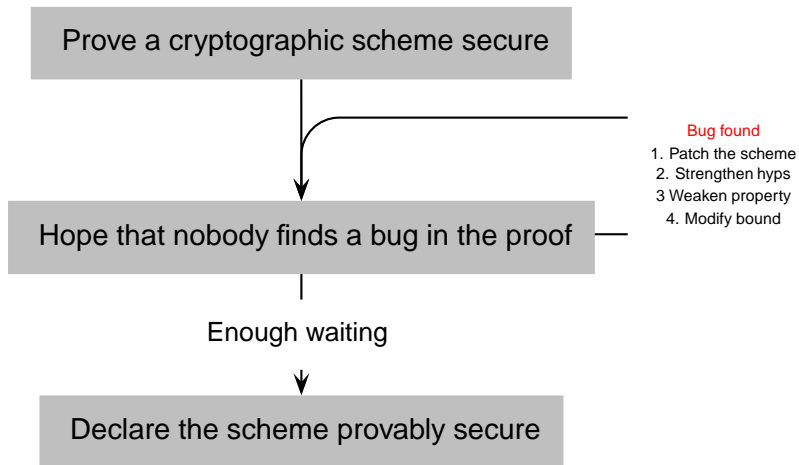
How much time is *enough*?

Provable security



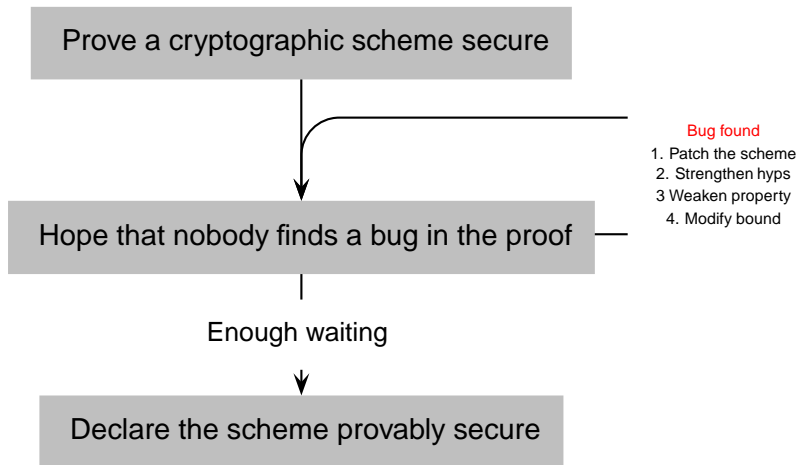
6 months, 1 year, 2 years?

Provable security



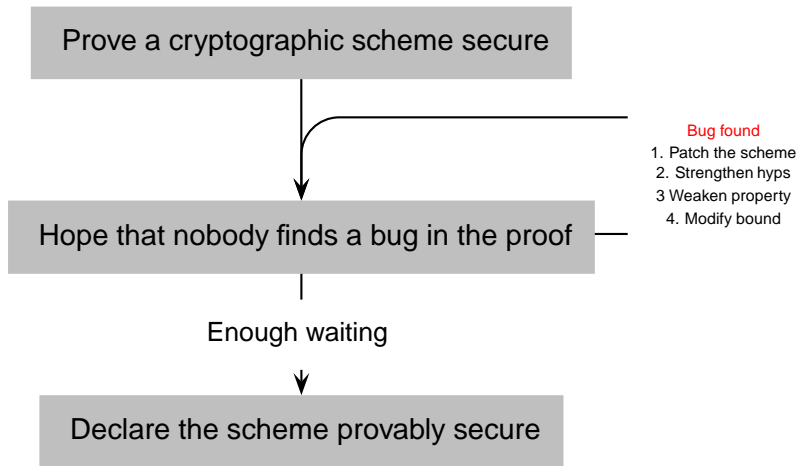
It took **6 years** to discover that the original proof of OAEP “only” established IND-CCA1

Provable security



Ok, let's say **7 years** to be on the safe side

Provable security



It took more than **10 years** to produce an almost correct proof of IND-CCA2 proof of OAEP

Provable security

Increasing complexity and number of proofs



Hope that nobody finds a bug in the proof



Enough waiting



Declare the scheme provably secure

Provable security

Increasing complexity and number of proofs



No time to check long handmade proofs carefully



Enough waiting



Declare the scheme provably secure

Provable security

Increasing complexity and number of proofs



No time to check long handmade proofs carefully



Enough waiting **brings little guarantees**



Provable security

Increasing complexity and number of proofs



No time to check long handmade proofs carefully



Enough waiting **brings little guarantees**



Cryptographic proofs are becoming unverifiable!

Provable security

Increasing complexity and number of proofs



No time to check long handmade proofs carefully



Enough waiting brings little guarantees



Cryptographic proofs are becoming unverifiable!

Can't we do better?

Provable security

Increasing complexity and number of proofs



No time to check long handmade proofs carefully



Enough waiting brings little guarantees



Cryptographic proofs are becoming unverifiable!

Yes, we can!

CertiCrypt: machine-checking provable security

Goal

Build on top of a general purpose proof assistant a certified framework for checking exact provable security proofs

- Security goals, properties and hypotheses are explicit
- All proof steps are conducted in a unified formalism
- The tool provides independently checkable certificates

For now, we do not attempt:

- to find proofs fully automatically
- to provide a nice interface, generate efficient code. . .

Proof assistants

Proof assistants implement a unifying formalism for:

- defining mathematical objects
- performing computations on these objects
- reasoning about these objects

Sample applications

- Mathematics and logic:
 - 4-color theorem
 - Kepler's conjecture
 - Category theory, Galois theory, FTA
 - Type theory
- Programming languages: semantics, compilers, reasoning tools (abstract interpreters, static analyses, type systems, Hoare logics, verification condition generators)
- Cryptographic protocols
- Operating systems, floating-point arithmetic

Proof assistants: definitions and computations

The underlying formalism of proof assistants is type theory, an expressive language in which mathematical objects can be formalized

- Data types: natural numbers, lists, polynomials. . .
- Algebraic structures: groups, fields. . .

Type theory also supports executable definitions, hence proof assistants can be used as (restricted) typed functional programming language

Rule of thumb

- Any mathematical notion can be formalized
- A lot of functions can be specified in executable form

Proof assistants: proof mode

- Interactive proofs, with mechanisms to guarantee that
 - theorems are applied with the right hypotheses
 - functions are applied to the right arguments
 - no missing cases in proofs or in function definitions
 - no illicit logical step

Proof assistants include domain-specific tactics that help solving specific problems efficiently (ring, omega. . .)

Proof by reflection

Perform deduction by computation: more automation, smaller proof objects

- A predicate $P : T \rightarrow \mathbf{Prop}$
- A decision procedure $f : T \rightarrow \mathbf{bool}$
- A correctness lemma $C : \prod x : T. f\ x = \mathbf{true} \rightarrow P\ x$

If $f\ a$ reduces to true, then $C\ a$ (`eq_refl true`) is a proof of $P\ a$

The game-playing methodology

- How do we formalize the statement?

The game-playing methodology

- Game = Probabilistic program

Game G_0^η :

...

... $\leftarrow \mathcal{A}(\dots)$;

...

$\Pr_{G_0^\eta}[A_0]$

The game-playing methodology

- Game = Probabilistic program
- How do we perform the proof?

Game G_0^η :

...

... $\leftarrow \mathcal{A}(\dots)$;

...

$\Pr_{G_0^\eta}[A_0]$

The game-playing methodology

- Game = Probabilistic program
- Game transformation = Program transformation

Game G_0^η :
...
... $\leftarrow \mathcal{A}(\dots)$;
...

Game G_1^η :
...
...
...

...

Game G_n^η :
...
... $\leftarrow \mathcal{B}(\dots)$;
...

$$\Pr_{G_0^\eta}[A_0] \leq h_1(\Pr_{G_1^\eta}[A_1]) \leq \dots \leq h_n(\Pr_{G_n^\eta}[A_n])$$

The game-playing methodology

- Game = Probabilistic program
- Game transformation = Program transformation

Game G_0^η :
...
... $\leftarrow \mathcal{A}(\dots)$;
...

Game G_1^η :
...
...
...

...

Game G_n^η :
...
... $\leftarrow \mathcal{B}(\dots)$;
...

$$\Pr_{G_0^\eta}[A_0] \leq h_1(\Pr_{G_1^\eta}[A_1]) \leq \dots \leq h_n(\Pr_{G_n^\eta}[A_n])$$

The game-playing methodology

- Bellare and Rogaway
- Shoup
- Halevi

Example: semantic security of ElGamal

- Key generation: $\mathcal{KG}() \stackrel{\text{def}}{=} x \xleftarrow{\$} \mathbb{Z}_q$; return (x, g^x)
The public key is g^x . The private key is x .
- Encryption: $\text{Enc}(\alpha, m) \stackrel{\text{def}}{=} y \xleftarrow{\$} \mathbb{Z}_q$; return $(g^y, \alpha^y \times m)$
- Decryption: $\text{Dec}(\beta, (c_1, c_2)) \stackrel{\text{def}}{=} s \leftarrow c_1^\beta$; return $(c_2 \times s^{-1})$

DDH assumption

Let G be a cyclic group of order q , let g be a generator of G

$$DDH_0 = x \xleftarrow{\$} \mathbb{Z}_q; y \xleftarrow{\$} \mathbb{Z}_q; b \leftarrow \mathcal{A}(g^x, g^y, g^{x*y});$$

$$DDH_1 = x \xleftarrow{\$} \mathbb{Z}_q; y \xleftarrow{\$} \mathbb{Z}_q; z \xleftarrow{\$} \mathbb{Z}_q; b \leftarrow \mathcal{A}(g^x, g^y, g^z);$$

For all PPT adversaries,

$$\epsilon_{DDH} \stackrel{\text{def}}{=} |\Pr_{DDH_0}[b = 1] - \Pr_{DDH_1}[b = 1]|$$

is negligible

IND-CPA game

EIGamal is IND-CPA secure under the DDH assumption

IND-CPA game

Game IND-CPA :

$(sk, pk) \leftarrow \mathcal{KG}(\cdot);$
 $(m_0, m_1) \leftarrow \mathcal{A}(pk);$
 $b \xleftarrow{\$} \{0, 1\};$
 $\zeta \leftarrow \text{Enc}(pk, m_b);$
 $b' \leftarrow \mathcal{A}'(pk, \zeta);$
 $d \leftarrow b = b'$

Statement

For all well-formed adversary $(\mathcal{A}, \mathcal{A}')$,

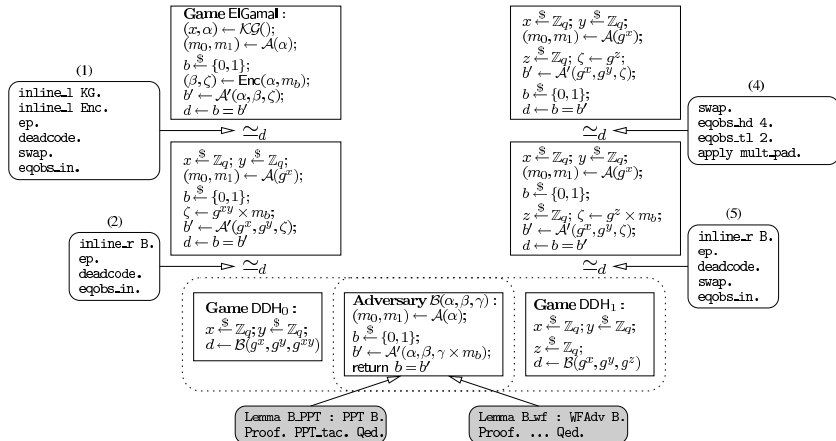
$$|\Pr_{\text{IND-CPA}}[d] - \frac{1}{2}| \leq \epsilon_{DDH}$$

where

$\mathcal{KG}(\cdot) \stackrel{\text{def}}{=} x \xleftarrow{\$} \mathbb{Z}_q; \text{ return } (x, g^x)$

$\text{Enc}(\alpha, m) \stackrel{\text{def}}{=} y \xleftarrow{\$} \mathbb{Z}_q; \text{ return } (g^y, \alpha^y \times m)$

Overview of machine-checked proof



Example: ElGamal encryption

Game ElGamal₀ :

$x \xleftarrow{\$} \mathbb{Z}_q; y \xleftarrow{\$} \mathbb{Z}_q;$
 $(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
 $b \xleftarrow{\$} \{0, 1\};$
 $\zeta \leftarrow g^{xy} \times m_b;$
 $b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
 $d \leftarrow b = b'$

$\simeq_{\emptyset}^{\{d\}}$

Game DDH₀ :

$x \xleftarrow{\$} \mathbb{Z}_q;$
 $y \xleftarrow{\$} \mathbb{Z}_q;$
 $d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})$

Example: ElGamal encryption

$$\begin{aligned}x &\leftarrow_{\mathcal{S}} \mathbb{Z}_q; y \leftarrow_{\mathcal{S}} \mathbb{Z}_q; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\leftarrow_{\mathcal{S}} \{0, 1\}; \\ \zeta &\leftarrow g^{xy} \times m_b; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, \zeta); \\d &\leftarrow b = b'\end{aligned}$$
$$\simeq_{\{d\}}^{\emptyset}$$
$$\begin{aligned}x &\leftarrow_{\mathcal{S}} \mathbb{Z}_q; \\y &\leftarrow_{\mathcal{S}} \mathbb{Z}_q; \\d &\leftarrow \mathcal{B}(g^x, g^y, g^{xy})\end{aligned}$$

Lemma foo: $\models \text{ElGamal}_0 \simeq_{\{d\}}^{\emptyset} \text{DDH}_0$

Proof .

Example: ElGamal encryption

$$\begin{aligned}x &\leftarrow_{\$} \mathbb{Z}_q; y \leftarrow_{\$} \mathbb{Z}_q; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\leftarrow_{\$} \{0, 1\}; \\ \zeta &\leftarrow g^{xy} \times m_b; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, \zeta); \\d &\leftarrow b = b'\end{aligned}$$
$$\simeq_{\{d\}}^{\emptyset}$$
$$\begin{aligned}x &\leftarrow_{\$} \mathbb{Z}_q; \\y &\leftarrow_{\$} \mathbb{Z}_q; \\ \alpha &\leftarrow g^x; \beta \leftarrow g^y; \gamma \leftarrow g^{xy}; \\(m_0, m_1) &\leftarrow \mathcal{A}(\alpha); \\b &\leftarrow_{\$} \{0, 1\}; \\b' &\leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b); \\d &\leftarrow b = b'\end{aligned}$$

Lemma foo: $\models \text{ElGamal}_0 \simeq_{\{d\}}^{\emptyset} \text{DDH}_0$

Proof.

inline_r B.

Example: ElGamal encryption

$$\begin{aligned}x &\stackrel{\$}{\leftarrow} \mathbb{Z}_q; y \stackrel{\$}{\leftarrow} \mathbb{Z}_q; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\stackrel{\$}{\leftarrow} \{0, 1\}; \\ \zeta &\leftarrow g^{xy} \times m_b; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, g^{xy} \times m_b); \\d &\leftarrow b = b'\end{aligned}$$
$$\stackrel{\emptyset}{\sim}_{\{d\}}$$
$$\begin{aligned}x &\stackrel{\$}{\leftarrow} \mathbb{Z}_q; \\y &\stackrel{\$}{\leftarrow} \mathbb{Z}_q; \\ \alpha &\leftarrow g^x; \beta \leftarrow g^y; \gamma \leftarrow g^{xy}; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\stackrel{\$}{\leftarrow} \{0, 1\}; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, g^{xy} \times m_b); \\d &\leftarrow b = b'\end{aligned}$$

Lemma foo: $\models \text{ElGamal}_0 \stackrel{\emptyset}{\sim}_{\{d\}} \text{DDH}_0$

Proof.

inline_r B.
ep.

Example: ElGamal encryption

$$\begin{aligned}x &\leftarrow_{\$} \mathbb{Z}_q; y \leftarrow_{\$} \mathbb{Z}_q; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\leftarrow_{\$} \{0, 1\}; \\ \zeta &\leftarrow g^{xy} \times m_b; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, g^{xy} \times m_b); \\d &\leftarrow b = b'\end{aligned}$$
$$\simeq_{\{d\}}^{\emptyset}$$
$$\begin{aligned}x &\leftarrow_{\$} \mathbb{Z}_q; \\y &\leftarrow_{\$} \mathbb{Z}_q; \\ \alpha &\leftarrow g^x; \beta \leftarrow g^y; \gamma \leftarrow g^{xy}; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\leftarrow_{\$} \{0, 1\}; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, g^{xy} \times m_b); \\d &\leftarrow b = b'\end{aligned}$$

Lemma foo: $\models \text{ElGamal}_0 \simeq_{\{d\}}^{\emptyset} \text{DDH}_0$

Proof.

inline_r B.
ep.
deadcode.

Example: ElGamal encryption

$$\begin{aligned}x &\leftarrow_{\$} \mathbb{Z}_q; y \leftarrow_{\$} \mathbb{Z}_q; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\leftarrow_{\$} \{0, 1\}; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, g^{xy} \times m_b); \\d &\leftarrow b = b'\end{aligned}$$
$$\simeq_{\{d\}}^{\emptyset}$$
$$\begin{aligned}x &\leftarrow_{\$} \mathbb{Z}_q; y \leftarrow_{\$} \mathbb{Z}_q; \\(m_0, m_1) &\leftarrow \mathcal{A}(g^x); \\b &\leftarrow_{\$} \{0, 1\}; \\b' &\leftarrow \mathcal{A}'(g^x, g^y, g^{xy} \times m_b); \\d &\leftarrow b = b'\end{aligned}$$

Lemma foo: $\models \text{ElGamal}_0 \simeq_{\{d\}}^{\emptyset} \text{DDH}_0$

Proof.

inline_r B.
ep.
deadcode.
eqobs_in.

Qed.

$$\Pr_{\text{ElGamal}_0, m}[b = b'] = \Pr_{\text{DDH}_0, m}[b = b']$$

Random oracles

- ElGamal involves adversary calls and IND-CPA involves encryption and key generation oracles
- Often games involves random oracles

```
Oracle  $H(\lambda)$  :  
if  $\lambda \notin \text{dom}(L)$  then  
   $h \xleftarrow{\$} \{0, 1\}^\ell$ ;  
   $L \leftarrow (\lambda, h) :: L$   
else  $h \leftarrow L(\lambda)$   
return  $h$ 
```

Random oracles are an idealization of hash functions

Hashed ElGamal

$$\begin{aligned} (x, \alpha) &\leftarrow \mathcal{KG}(\cdot) && \stackrel{\text{def}}{=} x \xleftarrow{\$} \mathbb{Z}_q; \text{ return } (x, g^x) \\ (\beta, v) &\leftarrow \text{Enc}(\alpha, m) && \stackrel{\text{def}}{=} y \xleftarrow{\$} \mathbb{Z}_q; h \leftarrow H(\alpha^y); \text{ return } (g^y, h \oplus m) \\ m &\leftarrow \text{Dec}(x, \beta, v) && \stackrel{\text{def}}{=} h \leftarrow H(\beta^x); \text{ return } h \oplus v \end{aligned}$$

$$\text{Dec}(x, g^y, H(g^{xy}) \oplus m) = H((g^y)^x) \oplus H(g^{xy}) \oplus m = m$$

What do we need?

- Formalization of probabilistic programs
 - Probability library
 - Libraries of arithmetic
 - Semantics of probabilistic programs
- Formalization of adversarial model
 - Complexity and termination
 - Usage policies (variables and procedures)
 - Well-formed adversaries
- Formalization of security definitions
- Tools to reason about probabilistic programs
 - Semantics-preserving program transformations
 - Observational equivalence and relational logic
 - Game-based lemmas (failure events)
- Tools to reason about mathematics

ALEA library (Christine Paulin)

The ALEA library provides a formalization of probabilities.

- Formalization of ω -complete partial orders
 - Monotonic sequences have a least upper bound
- Axiomatization of $U = [0, 1]$ interval as an ω -CPO
 - Bounded addition

$$x +_U y = \min(x +_{\mathbb{R}} y, 1)$$

- Multiplication
- Division

$$x /_U y = \min(x /_{\mathbb{R}} y, 1)$$

and

$$x /_U 0 = 0$$

- Inverse

Good starting point for CertiCrypt

Formalizing (sub)distributions: intuition

Let A be a set and $E : A \rightarrow \text{bool}$ be an event over A

Naive formalisation

$$\mathcal{D}_A = A \rightarrow [0, 1]$$

$$d E = \sum_{a \in A} d(a) \mathbb{1}_{Ea}$$

Sounds good, but A must be finite and we need big operators (cf. `ssreflect`)

A better definition:

Monadic formalisation

$$\mathcal{D}_A = (A \rightarrow [0, 1]) \rightarrow [0, 1]$$

$$d E = d (\lambda a. \mathbb{1}_{Ea})$$

Works smoothly, no need for finiteness.

Both definitions are easily proved equivalent (informally)

(Sub)distributions formally

Distributions are monotonic functions of type

$$(A \rightarrow [0, 1]) \xrightarrow{m} [0, 1]$$

satisfying the following properties:

- $d(f_1 + f_2) = d(f_1) + d(f_2)$ when $f_1 \leq 1 - f_2$
- $d(k \times f) = k \times d(f)$
- $d(1 - f) \leq 1 - d(f)$
- $d(\text{lub } (\lambda n. f\ n)) \leq \text{lub } (\lambda n. d(f\ n))$
when f is a monotonic sequence of functions

Distribution monad

$$\hat{\mathcal{D}}(X) \stackrel{\text{def}}{=} (X \rightarrow [0, 1]) \rightarrow [0, 1]$$

$$\begin{aligned} \text{unit} &: X \rightarrow \hat{\mathcal{D}}(X) \\ &\stackrel{\text{def}}{=} \lambda x. \lambda f. f \ x \end{aligned}$$

$$\begin{aligned} \text{bind} &: \hat{\mathcal{D}}(X) \rightarrow (X \rightarrow \hat{\mathcal{D}}(Y)) \rightarrow \hat{\mathcal{D}}(Y) \\ &\stackrel{\text{def}}{=} \lambda \mu. \lambda M. \lambda f. \mu(\lambda x. M \ x \ f) \end{aligned}$$

unit and bind satisfy the required properties, i.e.

$$\text{unit} : X \rightarrow \mathcal{D}(X)$$

$$\text{bind} : \mathcal{D}(X) \rightarrow (X \rightarrow \mathcal{D}(Y)) \rightarrow \mathcal{D}(Y)$$

PWHILE: a probabilistic programming language

| | | | |
|---------------|-----|---|-----------------|
| \mathcal{C} | ::= | skip | nop |
| | | $\mathcal{C}; \mathcal{C}$ | sequence |
| | | $\mathcal{V} \leftarrow \mathcal{E}$ | assignment |
| | | $\mathcal{V} \overset{\$}{\leftarrow} T$ | random sampling |
| | | if \mathcal{E} then \mathcal{C} else \mathcal{C} | conditional |
| | | while \mathcal{E} do \mathcal{C} | while loop |
| | | $\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \dots, \mathcal{E})$ | procedure call |

- $x \overset{\$}{\leftarrow} T$: sample the value of x according to uniform distribution over T
- The language of expressions admits user-defined extensions. Likewise for distributions
- Expressions and commands are typed
 - Avoids partial semantics of expressions
 - Enforces size constraints (e.g. length of bitstrings)

Deep embedding

The syntax of programs is formalized as an inductive type

Formalisation of expressions

Use dependent types for enforcing well-typed expressions

Inductive type : **Type** := ...

Inductive \mathcal{V} : type \rightarrow **Type** := ...

Inductive \mathcal{E} : type \rightarrow **Type** :=

| **Enat** : $\mathbb{N} \rightarrow \mathcal{E}_{\text{Nat}}$

| **Ebool** : $\mathbb{B} \rightarrow \mathcal{E}_{\text{Bool}}$

| **Evar** : $\forall t, \mathcal{V}_t \rightarrow \mathcal{E}_t$

| **Eop** : $\forall op, \mathcal{E}_{(\text{targs } op)}^* \rightarrow \mathcal{E}_{(\text{tres } op)}$

| **Eexists** : $\forall t, \mathcal{V}_t \rightarrow \mathcal{E}_{\text{Bool}} \rightarrow \mathcal{E}_{(\text{List } t)} \rightarrow \mathcal{E}_{\text{Bool}}$

| **Eforall** : $\forall t, \mathcal{V}_t \rightarrow \mathcal{E}_{\text{Bool}} \rightarrow \mathcal{E}_{(\text{List } t)} \rightarrow \mathcal{E}_{\text{Bool}}$

| **Efind** : $\forall t, \mathcal{V}_t \rightarrow \mathcal{E}_{\text{Bool}} \rightarrow \mathcal{E}_{(\text{List } t)} \rightarrow \mathcal{E}_t$.

Security parameter

Also use dependent types for dealing with security parameter in the interpretation of expressions

Definition $\text{interp} : \forall k, \forall t, \mathbf{Type} := \dots$

Parameter $\text{Mem} : \mathbb{N} \rightarrow \mathbf{Type}$.

Parameter $\text{get} : \forall k, \text{Mem } k \rightarrow \forall t, \mathcal{V}_t \rightarrow \text{interp } k \ t$.

Fixpoint $\llbracket \cdot \rrbracket k \ t \ (e : \mathcal{E}_t) \ (m : \text{Mem } k) : \text{interp } k \ t := \dots$

(Ommitted in the rest of the talk)

Execution

Given a command, an initial memory, and an expectation function, returns the probability for final memory:

$$\llbracket \cdot \rrbracket : \mathcal{C} \rightarrow \mathcal{M} \rightarrow \mathcal{D}_{\mathcal{M}}$$

The definition of the semantics is done in 3 steps:

- States with stack frames for context calls
- Small-step semantics on states

$$\llbracket \cdot \rrbracket_1 : \mathcal{C} \rightarrow \mathcal{S} \rightarrow \mathcal{D}_{\mathcal{S}}$$

- Let $f!$ is the restriction of f to final states. Then

$$\llbracket c \rrbracket \mu f = \text{lub } (\lambda n \cdot \llbracket c \rrbracket_n \mu f!)$$

- Nice to have semantics as a function
- But there may be a prettier construction

Semantics: what we really use

$$\begin{aligned} \llbracket \text{nil} \rrbracket m &= \text{unit } m \\ \llbracket i; c \rrbracket m &= \text{bind } (\llbracket i \rrbracket m) \llbracket c \rrbracket \\ \llbracket x \leftarrow e \rrbracket m &= \text{unit } m\{\llbracket e \rrbracket_{\text{expr}} m/x\} \\ \llbracket x \xleftarrow{\$} A \rrbracket m &= \text{bind } \mathcal{U}_A (\lambda v. \text{unit } m\{v/x\}) \\ \llbracket \text{if } e \text{ then } c_1 \text{ else } c_2 \rrbracket m &= \begin{cases} \llbracket c_1 \rrbracket m & \text{if } \llbracket e \rrbracket_{\text{expr}} m = \text{true} \\ \llbracket c_2 \rrbracket m & \text{if } \llbracket e \rrbracket_{\text{expr}} m = \text{false} \end{cases} \\ \llbracket \text{while } e \text{ do } c \rrbracket m &= \llbracket \text{if } e \text{ then } c; \text{ while } e \text{ do } c \rrbracket m \end{aligned}$$

and the rule for function calls. Also:

$$\llbracket \text{while } e \text{ do } c \rrbracket m f = \text{lub}\{\llbracket [\text{while } e \text{ do } c]_n \rrbracket m f_{|-e} : n \in \mathbb{N}\}$$

where $[\text{while } e \text{ do } c]_n$ is the n -step unrolling of the loop, i.e.

$$\begin{aligned} [\text{while } e \text{ do } c]_0 &= \text{nil} \\ [\text{while } e \text{ do } c]_{n+1} &= \text{if } e \text{ then } c; [\text{while } e \text{ do } c]_n \end{aligned}$$

Cost

Semantics is instrumented with cost monad

$$\llbracket \cdot \rrbracket : \mathcal{C} \rightarrow (\mathcal{S} \times \mathbb{N}) \rightarrow (\mathcal{S} \times \mathbb{N} \rightarrow [0, 1]) \rightarrow [0, 1]$$

For example

$$\llbracket x \leftarrow e \rrbracket (m, n) = \text{unit} (m\{\llbracket e \rrbracket_{\text{expr}} m\}/x\}, n + \llbracket e \rrbracket_{\text{exprcost}} m)$$

Currently

Assuming a cost function for each function symbol of the expression language

Eventually

Would be neat to use type systems to verify the cost of a function, e.g. Cook and Bellantoni, or Hofmann

PPT programs

- Define for each type the size of its elements
- Let p and q be polynomials on the security parameter. A state $(m, n)_k$ is (p, q) bounded if $p(k)$ bounds the size of values in m and $n \leq q(k)$
- Let $F : (\mathbb{N}[x] \times \mathbb{N}[x]) \rightarrow (\mathbb{N}[x] \times \mathbb{N}[x])$
- A program c is strict *PPT* with witness F iff it is lossless and for all $d : \mathcal{D}_{S \times \mathbb{N}}$ and $p, q : \mathbb{N}[x]$

$\text{range}(\text{bounded } p \ q) \ d \Rightarrow \text{range}(\text{bounded } (F \ p \ q)) \ (\text{bind } d \llbracket c \rrbracket)$

Methodology

- Semantic definition
- Rules for each construct of the programming language
- Tactic for generating proofs

Adversaries

Adversaries are uninterpreted procedures. One must also specify for each adversary:

- which variables it can access/modify
- which oracles it can call
- how many times each oracle may be called
- with which arguments oracles may be called

We use:

- a static analysis for the first two
- instrumented semantics and postconditions for the last two

Characterizing well-formed adversaries

w.r.t. a set \mathcal{O} of oracles, a set \mathcal{V}_{rw} of read-write global variables, and a set \mathcal{V}_{ro} of read-only variables.

$$\begin{array}{c}
 I \vdash \text{nil} : I \quad \frac{I \vdash i : I' \quad I' \vdash c : O}{I \vdash i; c : O} \\
 \\
 \frac{\text{Writable}(x) \quad \text{fv}(e) \subseteq I}{I \vdash x \leftarrow e : I \cup \{x\}} \quad \frac{\text{Writable}(x) \quad \text{fv}(d) \subseteq I}{I \vdash x \leftarrow d : I \cup \{x\}} \\
 \\
 \frac{\text{fv}(e) \subseteq I \quad I \vdash c_i : O_i \quad i = 1, 2}{I \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : O_1 \cap O_2} \quad \frac{\text{fv}(e) \subseteq I \quad I \vdash c : I}{I \vdash \text{while } e \text{ do } c : I} \\
 \\
 \frac{\text{fv}(\vec{e}) \subseteq I \quad \text{Writable}(x) \quad p \in \mathcal{O}}{I \vdash x \leftarrow p(\vec{e}) : I \cup \{x\}} \\
 \\
 \frac{\text{fv}(\vec{e}) \subseteq I \quad \text{Writable}(x) \quad p \notin \mathcal{O} \quad \vdash_{\text{wf}} p}{I \vdash x \leftarrow p(\vec{e}) : I \cup \{x\}} \\
 \\
 \frac{\mathcal{V}_{\text{rw}} \cup \mathcal{V}_{\text{ro}} \cup \mathcal{A}.\text{params} \vdash \mathcal{A}.\text{body} : O \quad \text{fv}(\mathcal{A}.\text{re}) \subseteq O}{\vdash_{\text{wf}} \mathcal{A}}
 \end{array}$$

where $\text{Writable}(x) \stackrel{\text{def}}{=} \text{Local}(x) \vee x \in \mathcal{V}_{\text{rw}}$

Characterizing well-formed adversaries

If $\vdash_{\text{wf}} \mathcal{A}$, then adversary \mathcal{A} . . .

- always initializes local variables before using them
- only writes global variables in \mathcal{V}_{rw}
- only reads global variables in $\mathcal{V}_{\text{rw}} \cup \mathcal{V}_{\text{ro}}$
- may call oracles in \mathcal{O}
- may call a procedure not in \mathcal{O} , as long as it is itself a well-formed adversary

Example: IND-CCA game

Game $G_{\text{IND-CCA2}}$:

$L_{\text{Dec}} \leftarrow []$;

$(pk, sk) \leftarrow \mathcal{KG}(\eta)$;

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;

$b \xleftarrow{\$} \{0, 1\}$;

$c^* \leftarrow \text{Enc}(m_b)$;

$c_{\text{def}} \leftarrow \text{true}$;

$\bar{b} \leftarrow \mathcal{A}_2(pk, c^*)$

Oracle $\text{Dec}(c)$:

$L_{\text{Dec}} \leftarrow (c_{\text{def}}, c) :: L_{\text{Dec}}$;

...

$\forall \mathcal{A}, \text{WF}(\mathcal{A}) \wedge \text{range}((\text{true}, c^*) \notin L_{\text{Dec}}) \llbracket G_{\text{IND-CCA2}} \rrbracket \implies$

$$|\Pr_{G_{\text{IND-CCA2}}}[\bar{b} = b] - \frac{1}{2}| \leq \dots$$

Bounding the number of oracle calls

- Add counter for adversary calls; bound counter as postcondition
- Bound list of queries as postcondition

Reasoning tools

- Observational equivalence
- Relational Hoare Logic
- Program Transformations

All tools lie outside the Trusted Computing Base. They have been verified, and are not used in the security statements.

What's in the TCB?

- TCB is the part of the formalization that must be trusted to be confident in a proof.
- The library of probability and the semantics of `pWHILE` are part of the Trusted Computing Base.
- The statement itself must be inspected. For ElGamal, DDH, IND-CPA and the code for encryption and key generation.

Observational equivalence

Games G_1 and G_2 are observationally equivalent w.r.t. input variables I and output variables O iff:

- **IF** m_1 and m_2 coincide on I
- **THEN** $\llbracket G_1 \rrbracket m_1$ and $\llbracket G_2 \rrbracket m_2$ coincide on O (i.e. their projections on O are equal)

$$m_1 =_X m_2 \stackrel{\text{def}}{=} \forall x \in X, m_1 x = m_2 x$$

$$f =_X g \stackrel{\text{def}}{=} \forall m_1 m_2, m_1 =_X m_2 \implies f m_1 = g m_2$$

$$\models G_1 \simeq_O^I G_2 \stackrel{\text{def}}{=} \forall m_1 m_2 f g, m_1 =_I m_2 \wedge f =_O g \implies \llbracket G_1 \rrbracket m_1 f = \llbracket G_2 \rrbracket m_2 g$$

Observational equivalence: consequences

Assume $\models G_1 \simeq_O^I G_2$.

Reasoning about probabilities

- **IF** $m_1 =_I m_2$ and $A =_O A$ (A only depends on O),
- **THEN** $\Pr_{G_1, m_1}[A] = \Pr_{G_2, m_2}[A]$

Using \leq instead of $=$

- **IF** $m_1 =_I m_2$ and $f \leq_O g$
- **THEN** $\llbracket G_1 \rrbracket m_1 f \leq \llbracket G_2 \rrbracket m_2 g$

where $f \leq_O g \stackrel{\text{def}}{=} \forall m_1 m_2. m_1 =_O m_2 \Rightarrow f m_1 \leq g m_2$

As a special case of Relational Hoare Logic, we have built an equational theory for observational equivalence

Certified program transformations

Often needs to prove observational equivalence preserved by program transformations

- Transformation: $T(G_1, G_2, I, O) = (G'_1, G'_2, I', O')$
- Soundness theorem

$$\frac{T(G_1, G_2, I, O) = (G'_1, G'_2, I', O') \quad \models G'_1 \simeq_{O'}^{I'} G'_2}{\models G_1 \simeq_O^I G_2}$$

Supported transformations

- Dead code elimination (`deadcode`)
- Constant folding and propagation (`cp`)
- Procedure call inlining (`inline`)
- Code motion (`swap`)
- Common suffix/prefix elimination (`eqobs_hd`, `eqobs_tl`)

Self-equivalence

One often needs to prove that G is non-interfering i.e.

$$\models G \simeq'_0 G$$

`eqobs_in` checks self-equivalence

- It computes $I' \models G \simeq''_0 G$ (dependency analysis)
- it verifies $I' \subseteq I$

Example

if $x = 0$ then $y \leftarrow x$ else $y \leftarrow 1$

$$\simeq_{\substack{\{x\} \\ \{x,y\}}}$$

if $x = 0$ then $y \leftarrow 0$ else $y \leftarrow 1$

is proved by expression propagation and `eqobs_in`.

Observational equivalence and equational theories

Optimistic sampling

$$\models x \stackrel{\$}{\leftarrow} \{0, 1\}^k; y \leftarrow x \oplus z \simeq_{\{z\}}^{\{x, y, z\}} y \stackrel{\$}{\leftarrow} \{0, 1\}^k; x \leftarrow y \oplus z$$

How to prove observational equivalence?

- For A finite, and $f, g : A \rightarrow B$

$$x \stackrel{\$}{\leftarrow} A; y \leftarrow f x \simeq_{\emptyset}^{\{y\}} x \stackrel{\$}{\leftarrow} A; y \leftarrow g x$$

iff there exists $h : A \xrightarrow{1-1} A$ such that $g = f \circ h$

- Theory specific decision procedure, and connection to matching modulo equational theories (joint work with Marion Daubignard, Bruce Kapron, Yassine Lakhnech, Vincent Laporte)

Relational Hoare Logic

A generalization of observational equivalence

- Deterministic setting: $\models c_1 \sim c_2 : P \Rightarrow Q$ iff

$$\forall m_1 m_2, P m_1 m_2 \rightarrow Q \llbracket c_1 \rrbracket_{m_1} \llbracket c_2 \rrbracket_{m_2}$$

where P and Q are relations on memories

- Probabilistic setting: $\models c_1 \sim c_2 : P \Rightarrow Q$ iff

$$\forall m_1 m_2, P m_1 m_2 \rightarrow \text{lift } Q \llbracket c_1 \rrbracket_{m_1} \llbracket c_2 \rrbracket_{m_2}$$

(P and Q still are relations on memories)

Lifting of a relation

(inspired from probabilistic process algebra)

- Lifting relation

$$\begin{aligned} \text{lift } R (d_1 : \mathcal{D}_A) (d_2 : \mathcal{D}_B) &:= \\ \exists (d : \mathcal{D}_{A*B}), \pi_1(d) = d_1 \wedge \pi_2(d) = d_2 \wedge \text{range } R d \end{aligned}$$

- If R is a partial equivalence relation on A ,

$$\text{lift } R d_1 d_2 \Leftrightarrow \forall c : A, d_1[c] = d_2[c]$$

where $[c]$ is the equivalence class of c

Remarks on the relational Hoare judgments

- Restricting pre- and post-conditions to PERs makes the logic harder to use. E.g.

$$\vDash G_1 \sim G_2 : P_1 \wedge P_2 \Rightarrow Q_1 \wedge Q_2$$

Forcing pre- and post- to be PER amounts to require $P_1 = P_2$ and $Q_1 = Q_2$

- Composition is more useful than transitivity

$$\frac{\vDash G_1 \sim G : P' \Rightarrow Q' \quad \vDash G \sim G_2 : P'' \Rightarrow Q''}{\vDash G_1 \sim G_2 : P' \circ P'' \Rightarrow Q' \circ Q''} [\text{R-Comp}]$$

- Proof of transitivity/composition requires that semantics is discrete

Remarks on the relational Hoare judgments

$$\models x \stackrel{\$}{\leftarrow} \{0, 1\} \sim x \stackrel{\$}{\leftarrow} \{0, 1\} : \text{True} \Rightarrow =\{x\}$$

- Product distribution fails proving observational equivalence.
Pairs $(0, 1)$ and $(1, 0)$ violate the postcondition and have a non-null probability
- Choose the distribution that gives probability $1/2$ to $(0, 0)$ and $1/2$ to $(1, 1)$. Equivalence holds

Beware! We also have

$$\models x \stackrel{\$}{\leftarrow} \{0, 1\} \sim x \stackrel{\$}{\leftarrow} \{0, 1\} : \text{True} \Rightarrow \neq\{x\}$$

Relational Hoare Logic and probabilities

Assume $\models c_1 \sim c_2 : P \Rightarrow Q$

- If f and g do not distinguish memories related by Q , i.e.

$$\forall m_1 m_2, Q m_1 m_2 \rightarrow f m_1 = g m_2$$

then

$$\forall m_1 m_2, P m_1 m_2 \rightarrow \llbracket c_1 \rrbracket_{m_1} f = \llbracket c_2 \rrbracket_{m_2} g$$

- If

$$\forall m_1 m_2, Q m_1 m_2 \rightarrow f m_1 \leq g m_2$$

then

$$\forall m_1 m_2, P m_1 m_2 \rightarrow \llbracket c_1 \rrbracket_{m_1} f \leq \llbracket c_2 \rrbracket_{m_2} g$$

Selected rules

$$\frac{\vDash c_1 \sim c_2 : P \Rightarrow Q \quad P' \Rightarrow P}{\vDash c_1 \sim c_2 : P' \Rightarrow Q}$$

$$\frac{\vDash c_1 \sim c_2 : P \Rightarrow Q \quad Q \Rightarrow Q'}{\vDash c_1 \sim c_2 : P \Rightarrow Q'}$$

$$\frac{Q' := \lambda m_1 m_2, Q \ m_1 \{x_1 := \llbracket e_1 \rrbracket_{m_1}\} \ m_2 \{x_2 := \llbracket e_2 \rrbracket_{m_2}\}}{\vDash x_1 \leftarrow e_1 \sim x_2 \leftarrow e_2 : Q' \Rightarrow Q}$$

$$\frac{Q' := \lambda m_1 m_2, \mathbf{permut} \ f \ m_1 \ m_2 \ A \ \wedge \ \forall v \in \llbracket A \rrbracket, Q \ m_1 \{x := f \ m_1 \ m_2 \ v\} \ m_2 \{x := v\}}{\vDash x \stackrel{\$}{\leftarrow} A \sim x \stackrel{\$}{\leftarrow} A : Q' \Rightarrow Q}$$

$$\frac{\vDash c_1 \sim c'_1 : P \Rightarrow Q \quad \vDash c_2 \sim c'_2 : Q \Rightarrow R}{\vDash c_1; c_2 \sim c'_1; c'_2 : P \Rightarrow R}$$

$$\frac{\vDash c_1 \sim c'_1 : P|_e \Rightarrow Q \quad \vDash c_2 \sim c'_2 : P|_{\neg e} \Rightarrow Q \quad \llbracket e \rrbracket \simeq_P \llbracket e' \rrbracket}{\vDash \text{if } e \text{ then } c_1 \text{ else } c_2 \sim \text{if } e' \text{ then } c'_1 \text{ else } c'_2 : P \Rightarrow Q}$$

The Fundamental Lemma of Game-Playing

Assume two games G_1 and G_2 behave identically in an initial memory m unless a failure event “bad” fires, e.g.

Game G_1 :

...
bad \leftarrow true; c_1
...

Game G_2 :

...
bad \leftarrow true; c_2
...

- $\Pr_{G_1, m}[A \mid \neg \text{bad}] = \Pr_{G_2, m}[A \mid \neg \text{bad}]$
- $\Pr_{G_1, m}[\text{bad}] = \Pr_{G_2, m}[\text{bad}]$ (if G_1 and G_2 are lossless)

Fundamental lemma

$$|\Pr_{G_1, m}[A] - \Pr_{G_2, m}[A]| \leq \Pr_{G_1, m}[\text{bad}]$$

Failure Event lemma

The Fundamental Lemma is typically applied in games where only oracles trigger bad. The Failure Event lemma states:

- **IF** the probability an oracle to trigger bad can be bound as an expression of number of oracle calls
- **THEN** the probability of the game to trigger bad can also be bound as an expression of number of oracle calls

Failure Event Lemma (simplified)

Assume that $m \text{ bad} = \text{false}$

- **IF** $\Pr_{\mathcal{O},m}[\text{bad}] \leq k$ for every memory m such that $m \text{ bad} = \text{false}$
- **THEN** $\Pr_{\mathcal{G},m}[\text{bad}] \leq q_{\mathcal{O}} k$

Logic of Failure Events

We have developed a variant of Probabilistic Hoare Logic

$$\vdash \llbracket c \rrbracket g \preceq f \stackrel{\text{def}}{=} \forall m. \llbracket c \rrbracket m g \leq f m$$

Selected rules

$$\vdash \llbracket \text{nil} \rrbracket f \preceq f \quad \frac{f = \lambda m. g(m\{x := \llbracket e \rrbracket m\})}{\vdash \llbracket x \leftarrow e \rrbracket g \preceq f}$$

$$\frac{f = \lambda m. |\llbracket T \rrbracket|^{-1} \sum_{t \in \llbracket T \rrbracket} g(m\{x := t\})}{\vdash \llbracket x \stackrel{s}{\leftarrow} T \rrbracket g \preceq f}$$

$$\frac{\vdash \llbracket c_1 \rrbracket g \preceq f \quad \llbracket c_2 \rrbracket h \preceq g}{\vdash \llbracket c_1; c_2 \rrbracket h \preceq f}$$

$$\frac{\vdash \llbracket c_1 \rrbracket g \preceq f \quad \llbracket c_2 \rrbracket g \preceq f}{\vdash \llbracket \text{if } e \text{ then } c_1 \text{ else } c_2 \rrbracket g \preceq f}$$

Application: switching lemma

Game G_{RP} :

$L \leftarrow []$; $b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(L)$ then

$y \xleftarrow{\$} T \setminus \text{ran}(L)$;

$L \leftarrow (x, y) :: L$

return $L(x)$

Game G_{RF} :

$L \leftarrow []$; $b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(L)$ then

$y \xleftarrow{\$} T$;

$L \leftarrow (x, y) :: L$

return $L(x)$

Suppose that \mathcal{A} makes at most q queries to its oracle. Then

$$|\Pr_{G_{RP}}[b] - \Pr_{G_{RF}}[b]| \leq \frac{q(q-1)}{2(\#T)}$$

- First introduced by Impagliazzo and Rudich in 1989
- Proof fixed by Bellare and Rogaway (2006) and Shoup (2004)

Proof of the switching lemma

Failure Event Lemma (less simplified)

Let q be a counter for \mathcal{O} and $m \text{ bad} = \text{false}$

- **IF** $\Pr_{\mathcal{O},m}[\text{bad}] \leq f(m q)$ for all memories m such that $m \text{ bad} = \text{false}$
- **THEN** $\Pr_{G,m}[\text{bad}] \leq \sum_{0 \leq q < q_{\mathcal{O}}} f(q)$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(L)$ then

$y \xleftarrow{\$} T$; if $y \in \text{ran}(L)$ then ($\text{bad} \leftarrow \text{true}$; c)

$L \leftarrow (x, y) :: L$

return $L(x)$

- Take $c = \text{nil}$ and $c = y \xleftarrow{\$} T \setminus \text{ran}(L)$
- Prove that $\Pr_{G,m}[\text{bad}] \leq \frac{\text{length}(m L)}{\#T}$

Eager/lazy sampling

- Interprocedural code motion
- Eager sampling: move random sampling from oracle to main command
- Lazy sampling: move random sampling from main command to oracle

Lazy sampling is useful to know that some values are random and uniformly distributed at a given program point.

To prove correctness of eager and lazy sampling, we have developed a logic for swapping statements

$$\models E, c; S \simeq E', S; c'$$

Selected rules

Assume $\text{Modify}(E, S) \cup \text{Modify}(E', S) \subseteq X$ and $\models E, S \simeq_X^X E', S$

$$\frac{x \notin X \quad \text{fv}(e) \cap X = \emptyset}{\models E, (x \leftarrow e; S) \equiv E', (S; x \leftarrow e)}$$

$$\frac{x \notin X}{\models E, (x \leftarrow^s T; S) \equiv E', (S; x \leftarrow^s T)}$$

$$\frac{\models E, (c_1; S) \equiv E', (S; c'_1) \quad \models E, (c_2; S) \equiv E', (S; c'_2)}{\models E, (c_1; c_2; S) \equiv E', (S; c'_1; c'_2)}$$

$$\frac{\models E, (c_1; S) \equiv E', (S; c'_1) \quad \models E, (c_2; S) \equiv E', (S; c'_2) \quad \text{fv}(e) \cup X = \emptyset}{\models E, (\text{if } e \text{ then } c_1 \text{ else } c_2; S) \equiv E', (S; \text{if } e \text{ then } c'_1 \text{ else } c'_2)}$$

Example of application

Game G_1 :

$L \leftarrow [];$

c

$\mathcal{O}_1(x) \stackrel{\text{def}}{=}$

if $x \notin \text{dom}(L)$ then

$y \xleftarrow{\$} T;$

$L \leftarrow (x, y) :: L$

else $y \leftarrow L[x]$

return y

Game G_e :

$L \leftarrow [];$

$y' \xleftarrow{\$} T;$

c

$\mathcal{O}_e(x) \stackrel{\text{def}}{=}$

if $x \notin \text{dom}(L)$ then

if $x = x'$ then $y \leftarrow y'$ else $y \xleftarrow{\$} T;$

$L \leftarrow (x, y) :: L$

else $y \leftarrow L[x]$

return y

For every event that does not depend on y' , we have

$$\Pr_{E_1, c_1}[A] = \Pr_{E_e, c_e}[A]$$

Hashed ElGamal

$$\begin{array}{ll} (x, \alpha) \leftarrow \mathcal{KG}(\cdot) & \stackrel{\text{def}}{=} x \xleftarrow{\$} \mathbb{Z}_q; \text{ return } (x, g^x) \\ (\beta, v) \leftarrow \text{Enc}(\alpha, m) & \stackrel{\text{def}}{=} y \xleftarrow{\$} \mathbb{Z}_q; h \leftarrow H_k(\alpha^y); \text{ return } (g^y, h \oplus m) \\ m \leftarrow \text{Dec}(x, \beta, v) & \stackrel{\text{def}}{=} h \leftarrow H_k(\beta^x); \text{ return } h \oplus v \end{array}$$

$$\text{Dec}(x, g^y, H_k(g^{xy}) \oplus m) = H((g^y)^x) \oplus H(g^{xy}) \oplus m = m$$

Security of Hashed ElGamal in the ROM

Hashed ElGamal is IND-CPA secure (in the Random Oracle Model) under the List CDH assumption.

List CDH assumption

Game LCDH :

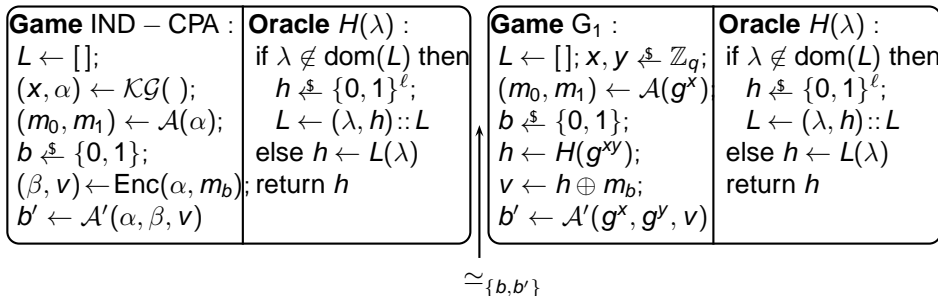
$$\begin{aligned}x, y &\stackrel{\$}{\leftarrow} \mathbb{Z}_q; \\ L &\leftarrow \mathcal{C}(g^x, g^y)\end{aligned}$$

Let

$$\epsilon_{\text{LCDH}} \stackrel{\text{def}}{=} \Pr_{\text{LCDH}}[g^{xy} \in L]$$

For every PPT adversary \mathcal{C} , ϵ_{LCDH} is negligible

Transition IND-CPA to G_1



By inlining \mathcal{KG} and Enc:

$$\Pr_{\text{IND-CPA}} [b = b'] = \Pr_{G_1} [b = b']$$

Transition G_1 to G_2

Game G_1 :

$L \leftarrow []$; $x, y \xleftarrow{\$} \mathbb{Z}_q$;
 $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
 $b \xleftarrow{\$} \{0, 1\}$;
 $h \leftarrow H(g^{xy})$;
 $v \leftarrow h \oplus m_b$;
 $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$

Oracle $H(\lambda)$:

if $\lambda \notin \text{dom}(L)$ then
 $h \xleftarrow{\$} \{0, 1\}^\ell$;
 $L \leftarrow (\lambda, h) :: L$
 else $h \leftarrow L(\lambda)$
 return h

Game G_2 :

$h^+ \xleftarrow{\$} \{0, 1\}^\ell$;
 $L \leftarrow []$; $x, y \xleftarrow{\$} \mathbb{Z}_q$;
 $\Lambda \leftarrow g^{xy}$;
 $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
 $b \xleftarrow{\$} \{0, 1\}$;
 $h \leftarrow H(\Lambda)$;
 $v \leftarrow h \oplus m_b$;
 $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$

Oracle $H_2(\lambda)$:

if $\lambda \notin \text{dom}(L)$ then
 if $\lambda = \Lambda$ then
 $h \leftarrow h^+$;
 else $h \xleftarrow{\$} \{0, 1\}^\ell$
 $L \leftarrow (\lambda, h) :: L$
 else $h \leftarrow L(\lambda)$
 return h

$\simeq_{\{b, b'\}}$

By lazy sampling:

$$\Pr_{G_1}[b = b'] = \Pr_{G_2}[b = b']$$

Transition G_2 to G_3

| | | | |
|---|--|--|--|
| <p>Game G_2 : $h^+ \stackrel{\\$}{\leftarrow} \{0, 1\}^\ell$; $L \leftarrow []$; $x, y \stackrel{\\$}{\leftarrow} \mathbb{Z}_q$; $\Lambda \leftarrow g^{xy}$; $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$; $b \stackrel{\\$}{\leftarrow} \{0, 1\}$; $h \leftarrow H(\Lambda)$; $v \leftarrow h \oplus m_b$; $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_2(\lambda)$: if $\lambda \notin \text{dom}(L)$ then if $\lambda = \Lambda$ then $h \leftarrow h^+$; else $h \stackrel{\\$}{\leftarrow} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> | <p>Game G_3 : $h^+ \stackrel{\\$}{\leftarrow} \{0, 1\}^\ell$; $L \leftarrow []$; $x, y \stackrel{\\$}{\leftarrow} \mathbb{Z}_q$; $\Lambda \leftarrow g^{xy}$; $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$; $b \stackrel{\\$}{\leftarrow} \{0, 1\}$; $h \leftarrow h^+$; $v \leftarrow h \oplus m_b$; $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_3(\lambda)$: if $\lambda = \Lambda$ then $h \leftarrow h^+$ else if $\lambda \notin \text{dom}(L)$ then $h \stackrel{\\$}{\leftarrow} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> |
|---|--|--|--|

$\sim \{b, b'\} \wedge \phi_{23}$

Invariant ϕ_{23} on H_2 and H_3

$(\Lambda \in \text{dom}(L) \implies L[\Lambda] = h^+) \langle 1 \rangle \wedge \forall \lambda, \lambda \neq \Lambda \langle 1 \rangle \implies L[\lambda] \langle 1 \rangle = L[\lambda] \langle 2 \rangle$

Transition G_2 to G_3

| | | | |
|--|--|---|--|
| <p>Game G_2 : $h^+ \xleftarrow{\\$} \{0, 1\}^\ell;$ $L \leftarrow []; x, y \xleftarrow{\\$} \mathbb{Z}_q;$ $\Lambda \leftarrow g^{xy};$ $(m_0, m_1) \leftarrow \mathcal{A}(g^x);$ $b \xleftarrow{\\$} \{0, 1\};$ $h \leftarrow H(\Lambda);$ $v \leftarrow h \oplus m_b;$ $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_2(\lambda)$: if $\lambda \notin \text{dom}(L)$ then if $\lambda = \Lambda$ then $h \leftarrow h^+;$ else $h \xleftarrow{\\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> | <p>Game G_3 : $h^+ \xleftarrow{\\$} \{0, 1\}^\ell;$ $L \leftarrow []; x, y \xleftarrow{\\$} \mathbb{Z}_q;$ $\Lambda \leftarrow g^{xy};$ $(m_0, m_1) \leftarrow \mathcal{A}(g^x);$ $b \xleftarrow{\\$} \{0, 1\};$ $h \leftarrow h^+;$ $v \leftarrow h \oplus m_b;$ $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_3(\lambda)$: if $\lambda = \Lambda$ then $h \leftarrow h^+$ else if $\lambda \notin \text{dom}(L)$ then $h \xleftarrow{\\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> |
|--|--|---|--|

$\sim \{b, b'\} \wedge \phi_{23}$

$$\Pr_{G_2}[b = b'] = \Pr_{G_3}[b = b']$$

Transition G_3 to G_4

| | | | |
|--|--|--|--|
| <p>Game G_3 : $h^+ \xleftarrow{\\$} \{0, 1\}^\ell$; $L \leftarrow []$; $x, y \xleftarrow{\\$} \mathbb{Z}_q$; $\Lambda \leftarrow g^{xy}$; $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$; $b \xleftarrow{\\$} \{0, 1\}$; $h \leftarrow h^+$; $v \leftarrow h \oplus m_b$; $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_3(\lambda)$: if $\lambda = \Lambda$ then $h \leftarrow h^+$ else if $\lambda \notin \text{dom}(L)$ then $h \xleftarrow{\\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> | <p>Game G_4 $\text{bad} \leftarrow \text{false}$; $h^+ \xleftarrow{\\$} \{0, 1\}^\ell$; $L \leftarrow []$; $x, y \xleftarrow{\\$} \mathbb{Z}_q$; $\Lambda \leftarrow g^{xy}$; $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$; $b \xleftarrow{\\$} \{0, 1\}$; $v \leftarrow h^+ \oplus m_b$; $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_4(\lambda)$: if $\lambda \notin \text{dom}(L)$ then if $\lambda = \Lambda$ then $\text{bad} \leftarrow \text{true}$; $h \leftarrow h^+$ else $h \xleftarrow{\\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> |
| $\sim_{\{b, b'\}}$ | | | |

We *undo* the modification to the hash oracle to prove:

$$\Pr_{G_3}[b = b'] = \Pr_{G_4}[b = b']$$

Transition G_4 to G_5

| | | | |
|---|--|---|---|
| Game G_4 $\text{bad} \leftarrow \text{false};$ $h^+ \xleftarrow{\$} \{0, 1\}^\ell;$ $L \leftarrow []; x, y \xleftarrow{\$} \mathbb{Z}_q;$ $\Lambda \leftarrow g^{xy};$ $(m_0, m_1) \leftarrow \mathcal{A}(g^x);$ $b \xleftarrow{\$} \{0, 1\};$ $v \leftarrow h^+ \oplus m_b;$ $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$ | Oracle $H_4(\lambda)$: if $\lambda \notin \text{dom}(L)$ then if $\lambda = \Lambda$ then $\text{bad} \leftarrow \text{true};$ $h \leftarrow h^+$ else $h \xleftarrow{\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h | Game G_5 $\text{bad} \leftarrow \text{false};$ $h^+ \xleftarrow{\$} \{0, 1\}^\ell;$ $L \leftarrow []; x, y \xleftarrow{\$} \mathbb{Z}_q;$ $\Lambda \leftarrow g^{xy};$ $(m_0, m_1) \leftarrow \mathcal{A}(g^x);$ $b \xleftarrow{\$} \{0, 1\};$ $v \leftarrow h^+ \oplus m_b;$ $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$ | Oracle $H_5(\lambda)$: if $\lambda \notin \text{dom}(L)$ then if $\lambda = \Lambda$ then $\text{bad} \leftarrow \text{true};$ $h \xleftarrow{\$} \{0, 1\}^\ell$ else $h \xleftarrow{\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h |
|---|--|---|---|

$\sim_{\{b, b'\}}$

Games G_4 and G_5 are syntactically equal to up to the point where the flag bad is raised. From the Fundamental Lemma:

$$|\Pr_{G_4}[b = b'] - \Pr_{G_5}[b = b']| \leq \Pr_{G_5}[\text{bad}]$$

Transition G_5 to G_6

| | | | |
|---|--|---|--|
| <p>Game G_5 $\text{bad} \leftarrow \text{false};$ $h^+ \xleftarrow{\\$} \{0, 1\}^\ell;$ $L \leftarrow []; x, y \xleftarrow{\\$} \mathbb{Z}_q;$ $\Lambda \leftarrow g^{xy};$ $(m_0, m_1) \leftarrow \mathcal{A}(g^x);$ $b \xleftarrow{\\$} \{0, 1\};$ $v \leftarrow h^+ \oplus m_b;$ $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_5(\lambda)$: if $\lambda \notin \text{dom}(L)$ then if $\lambda = \Lambda$ then $\text{bad} \leftarrow \text{true};$ $h \xleftarrow{\\$} \{0, 1\}^\ell$ else $h \xleftarrow{\\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> | <p>Game G_6 : $L \leftarrow []; x, y \xleftarrow{\\$} \mathbb{Z}_q;$ $\Lambda \leftarrow g^{xy};$ $(m_0, m_1) \leftarrow \mathcal{A}(g^x);$ $b \xleftarrow{\\$} \{0, 1\};$ $v \xleftarrow{\\$} \{0, 1\}^\ell;$ $h^+ \leftarrow v \oplus m_b;$ $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H(\lambda)$: if $\lambda \notin \text{dom}(L)$ then $h \xleftarrow{\\$} \{0, 1\}^\ell;$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> |
|---|--|---|--|

$$\sim \{L, \Lambda, b, b'\} \wedge (\text{bad} \implies \Lambda \in \text{dom}(L)) \langle 1 \rangle$$

$$\Pr_{G_5}[b = b'] = \Pr_{G_6}[b = b'] = \frac{1}{2}$$

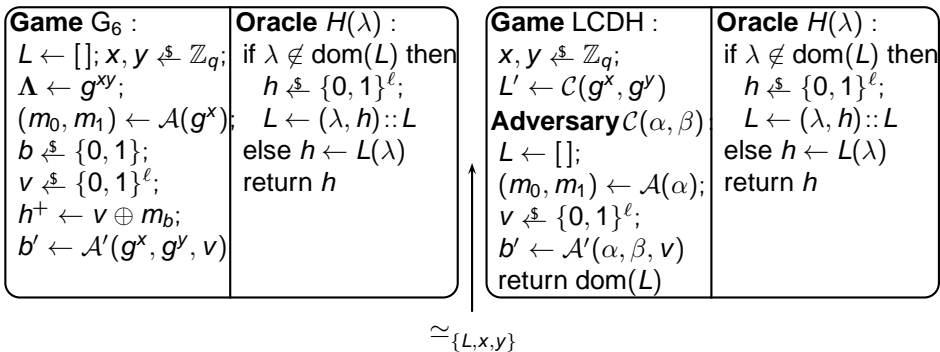
Transition G_5 to G_6

| | | | |
|--|--|--|--|
| <p>Game G_5</p> <p>bad \leftarrow false; $h^+ \xleftarrow{\\$} \{0, 1\}^\ell$; $L \leftarrow []$; $x, y \xleftarrow{\\$} \mathbb{Z}_q$; $\Lambda \leftarrow g^{xy}$; $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$; $b \xleftarrow{\\$} \{0, 1\}$; $v \leftarrow h^+ \oplus m_b$; $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H_5(\lambda)$:</p> <p>if $\lambda \notin \text{dom}(L)$ then if $\lambda = \Lambda$ then bad \leftarrow true; $h \xleftarrow{\\$} \{0, 1\}^\ell$ else $h \xleftarrow{\\$} \{0, 1\}^\ell$ $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> | <p>Game G_6 :</p> <p>$L \leftarrow []$; $x, y \xleftarrow{\\$} \mathbb{Z}_q$; $\Lambda \leftarrow g^{xy}$; $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$; $b \xleftarrow{\\$} \{0, 1\}$; $v \xleftarrow{\\$} \{0, 1\}^\ell$; $h^+ \leftarrow v \oplus m_b$; $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$</p> | <p>Oracle $H(\lambda)$:</p> <p>if $\lambda \notin \text{dom}(L)$ then $h \xleftarrow{\\$} \{0, 1\}^\ell$; $L \leftarrow (\lambda, h) :: L$ else $h \leftarrow L(\lambda)$ return h</p> |
|--|--|--|--|

$$\sim \{L, \Lambda, b, b'\} \wedge (\text{bad} \implies \Lambda \in \text{dom}(L)) \langle 1 \rangle$$

$$\Pr_{G_5}[\text{bad}] \leq \Pr_{G_6}[\Lambda \in \text{dom}(L)]$$

Transition G_6 to G_{LCDH}



$$\Pr_{G_6}[\Lambda \in \text{dom}(L)] = \Pr_{G_6}[g^{xy} \in \text{dom}(L)] = \Pr_{\text{LCDH}}[g^{xy} \in L']$$

Summarizing

$$\begin{aligned} |\Pr_{\text{IND-CPA}}[b = b'] - \frac{1}{2}| &= |\Pr_{G_4}[b = b'] - \frac{1}{2}| \\ &= |\Pr_{G_4}[b = b'] - \Pr_{G_6}[b = b']| \\ &= |\Pr_{G_4}[b = b'] - \Pr_{G_5}[b = b']| \\ &\leq \Pr_{G_5}[\text{bad}] \\ &\leq \Pr_{G_6}[\Lambda \in \text{dom}(L)] \\ &= \Pr_{\text{LCDH}}[g^{xy} \in L] \\ &= \epsilon_{\text{LCDH}} \end{aligned}$$

Proving the security of OAEP

- Bellare and Rogaway, 1994: first proof, under the assumption of one-way
- Shoup, 2001: proof works for IND-CCA1, but not for IND-CCA2. OAEP+ is IND-CCA2.
- Fujisaki, Okamoto, Pointcheval, and Stern, 2001: proof for IND-CCA2, under the stronger assumption of partial one-way
- Pointcheval, 2004: fills gaps in the 2001 proof, at the price of achieving a looser bound
- Bellare, Hofheinz, Kiltz, 2009: definition of IND-CCA2 needs to be clarified

OAEP padding scheme

$$\begin{aligned} \text{Enc}(M) &\stackrel{\text{def}}{=} R \xleftarrow{\$} \{0, 1\}^p; \\ &S \leftarrow G(R) \oplus (M \| 0^{k_1}); \\ &T \leftarrow H(S) \oplus R; \\ &Y \leftarrow f(S \| T); \\ &\text{return } Y \end{aligned}$$

- $(f, f^{-1}) : \{0, 1\}^k \rightarrow \{0, 1\}^k$ is a partial one-way trapdoor function
- G and H are random oracles
$$G(R) \stackrel{\text{def}}{=} \begin{cases} \text{if } R \notin L \text{ then } r \xleftarrow{\$} \{0, 1\}^k; L \leftarrow (R, r) \\ \text{else } r \leftarrow L[R] \end{cases}$$
$$\text{return } r$$

Exact IND-CCA security of OAEP

Adversary has access to random oracles and decryption oracle

Oracle Dec(c) :

$L_{\text{Dec}} \leftarrow (c_{\text{def}}, c) :: L_{\text{Dec}};$

$(s, t) \leftarrow f^{-1}(sk, c);$

$h \leftarrow H(s); r \leftarrow t \oplus h; g \leftarrow G(r);$

if $[s \oplus g]_{k_1} = 0^{k_1}$ then

 return $[s \oplus g]^n$

else return \perp

Security statement

$$|Pr_{\text{Game}}[b = b'] - \frac{1}{2}| \leq Pr_{I,f} + \frac{3q_D q_G + q_D^2 + 4q_D + q_G}{2^{k_0}} + \frac{2q_D}{2^{k_1}}$$

$Pr_{I,f}$ is the probability of an adversary I to invert f on a random element and q_X is the maximal number of queries to oracle X

Intuition of proof

- If $G(r(y))$ and $H(s(y))$ are not asked for a decryption query y , then $\text{Dec}(y) = \perp$ overwhelmingly
- Modify the decryption oracle to reject queries where $G(r(y))$ and $H(s(y))$ have not been asked previously
- Eventually decryption oracle acts as plaintext extractor
- First eliminate calls to G , and then calls to H (due to dependencies)

Key step: elimination of calls to G

Inline call to G and case analysis on $s \in \text{dom}(L_H)$. Reject ciphertexts with a fresh g or h

Oracle $\text{Dec}(c)$:

if $(c_{\text{def}} \wedge c^* = c) \vee q_{\text{Dec}} < |L_{\text{Dec}}| \vee q_{\text{Dec}} + q_G < |L_G|$ then return \perp
else

$L_{\text{Dec}} \leftarrow (c_{\text{def}}, c) :: L_{\text{Dec}}; (s, t) \leftarrow f^{-1}(sk, c);$

if $s \in \text{dom}(L_H)$ then

$r \leftarrow t \oplus H(s);$

if $r \in \text{dom}(L_G)$ then

$g \leftarrow L_G[r];$ if $[s \oplus g]_{k_1} = 0^{k_1}$ then return $[s \oplus g]^n$ else return \perp

else

if $r = r^*$ then bad \leftarrow true;

$g \xrightarrow{\$} \{0, 1\}^{n+k_1}; L_G[r] \leftarrow g;$ return \perp

else

$r \leftarrow t \oplus H(s);$ if $r \notin \text{dom}(L_G)$ then $g \xrightarrow{\$} \{0, 1\}^{n+k_1}; L_G[r] \leftarrow g;$ return \perp

is replaced by

Oracle $\text{Dec}(c)$:

if $(c_{\text{def}} \wedge c^* = c) \vee q_{\text{Dec}} < |L_{\text{Dec}}| \vee q_G < |L_G|$ then return \perp
else

$L_{\text{Dec}} \leftarrow (c_{\text{def}}, c) :: L_{\text{Dec}}; (s, t) \leftarrow f^{-1}(sk, c);$

if $s \in \text{dom}(L_H)$ then

$r \leftarrow t \oplus H(s);$

if $r \in \text{dom}(L_G)$ then

$g \leftarrow L_G[r];$ if $[s \oplus g]_{k_1} = 0^{k_1}$ then return $[s \oplus g]^n$ else return \perp

else

if $r = r^*$ then bad \leftarrow true; return \perp

else

$r \leftarrow t \oplus H(s);$ return \perp

Justifying elimination of calls to G

- Tag queries to G with origin (adversary vs. Dec)
- Set a bad flag in Dec when a valid ciphertext is produced with $G(r)$ not queried. In this case r uniformly distributed and independent from adversary's view (u.i.a.v.)
- Shift flag to G oracle
- Apply logic of swapping statements to show that values that are u.i.a.v. can be resampled
- Apply logic of failure events

Assessment

Our proof essentially follows Pointcheval (Barcelona'04). Two minor points:

- We eliminate both calls to G , whereas Pointcheval only eliminates the second call. Thus we obtain a better bound:

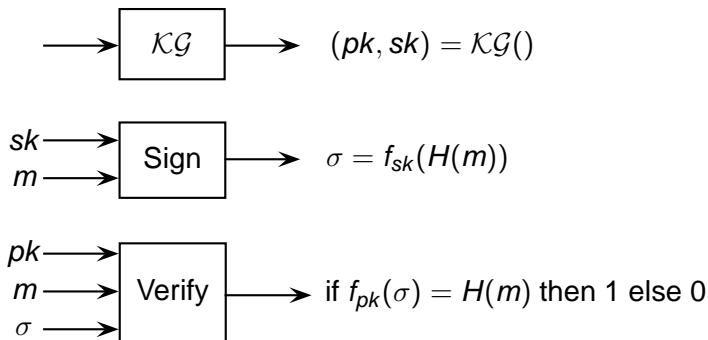
$$\epsilon' \geq \left(\frac{\epsilon}{2} - \frac{4q_{\text{Dec}}q_G + 2q_{\text{Dec}}^2 + 4q_{\text{Dec}} + 8q_G}{2^{k_0}} - \frac{3q_{\text{Dec}}}{2^{k_1}} \right)$$

- Elimination of calls to H requiring no more calls to G is overlooked in Pointcheval

Extensions

- Pointcheval specializes the proof to RSA-OAEP
- Boldyreva considers RSA-OAEP multi-query setting
- Backes, Dürmuth and Unruh consider security under key-dependent messages

The Full-Domain Hash Signature Scheme



$$\forall m, \text{Verify}(pk, m, \text{Sign}(sk, m)) = 1$$

Existential Unforgeability of FDH as a game

Forging the signature of message m should be hard, even if...

- ...the adversary knows the signatures of *many* messages
- ...the adversary chose those messages
- ...the adversary gets to choose m

| | |
|---|---|
| Game G_{EF} : $S \leftarrow \text{nil};$ $(pk, sk) \leftarrow \mathcal{KG};$ $(m, \sigma) \leftarrow \mathcal{A}(pk);$ $h \leftarrow H(m)$ | Oracle $H(m) \stackrel{\text{def}}{=} \text{return } H(m)$ Oracle $\text{Sign}(m) \stackrel{\text{def}}{=} S \leftarrow m :: S; \text{return } f_{sk}(H(m))$ |
|---|---|

$$\forall \mathcal{A}, \Pr [G_{EF} \mid f_{pk}(\sigma) = h \wedge m \notin S] \leq \epsilon$$

Existential unforgeability of FDH (after Coron)

Assume f is homomorphic w.r.t. the group operation. Consider an adversary \mathcal{A} s.t.

- \mathcal{A} makes at most q_H hash queries
- \mathcal{A} makes at most q_S signature queries

There exists an \mathcal{I} such that

$$\Pr_{\mathcal{I},f} \geq \frac{1}{q_S + 1} \left(1 - \frac{1}{q_S + 1}\right)^{q_S} \Pr[\mathcal{A} \text{ wins}]$$

Zero Knowledge Protocols

Given a relation R , a public input x and a witness w such that $(x, w) \in R$, convince you know w without revealing it

- Shown how to build Σ -protocols from special morphisms
- Formalized many existing examples of special morphisms
- Proved correctness of AND and OR constructions

Σ -protocol :

$r \leftarrow P1(x, w);$

$c \leftarrow V1(x, r);$

$s \leftarrow P2(x, w, c);$

accept $\leftarrow V2(x, r, c, s)$

Properties

- Completeness (accepting verifier)
- Special Honest Verifier ZK
- Soundness (no cheating prover)

Summary

- CertiCrypt supports code-based cryptographic proofs using programming language tools
 - observational equivalence,
 - relational Hoare logic,
 - certified program transformationsand a few non-standard tools
 - failure events, eager/lazy sampling
 - (others missing: “linearizing” adversary, etc)
- It has been used to verify exemplary schemes
 - FDH, OAEP, ZKand is being used for other examples
 - Identity-Based Encryption (Boneh and Franklin, 2001)
 - An Indifferentiable Hash Function into Elliptic Curves (Coron and Icart, 2009)
- It supports verifiable security!

Verifiable security

CertiCrypt allows breaking the symmetry between proof building and proof checking

Building

Sb with a lot of

- knowledge in crypto
- time

Verifying

Sb with a lot of

- knowledge in crypto
- time

Verifying with CertiCrypt

- Enough knowledge of cryptography to verify the statement (about 100 lines)
- A computer and 5' to independently verify the proof

Outlook

- Cryptographic proofs may be buggy
- Computer support can help achieve high confidence in cryptographic proofs

So is CertiCrypt useful for cryptographers?

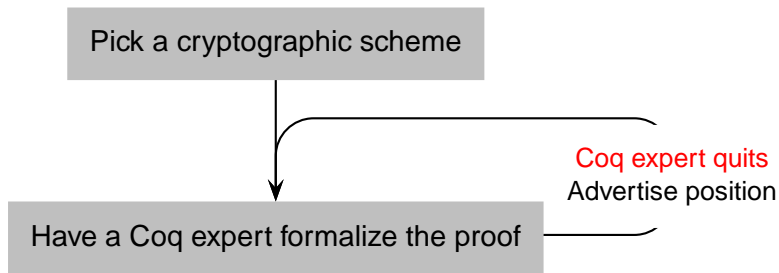
Verifiable security

Pick a cryptographic scheme

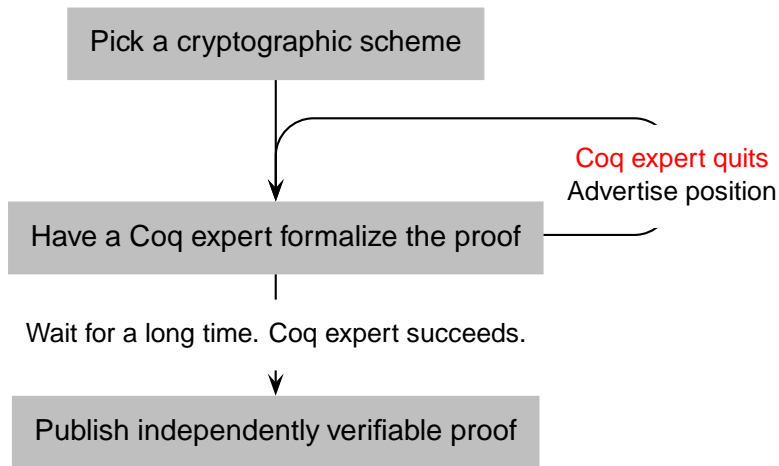


Have a Coq expert formalize the proof

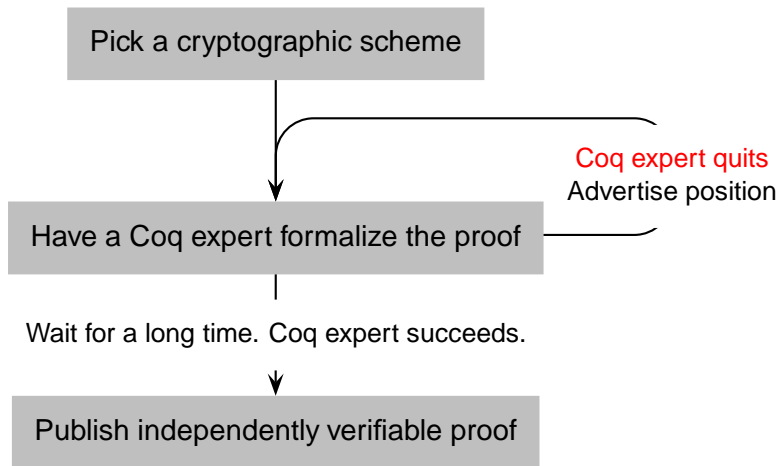
Verifiable security



Verifiable security

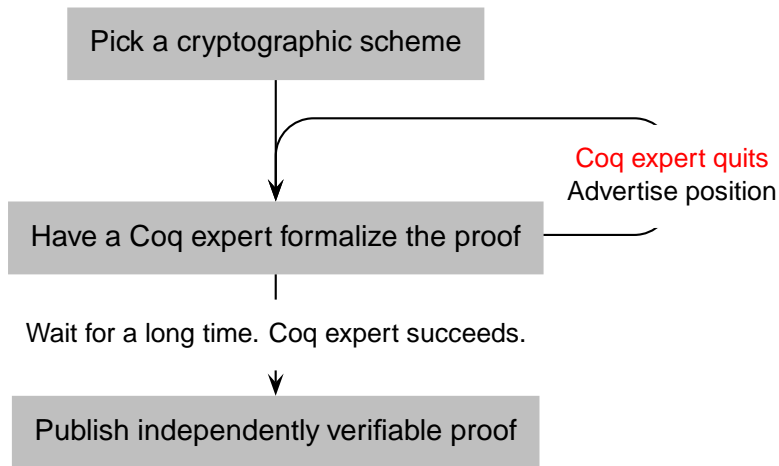


Verifiable security



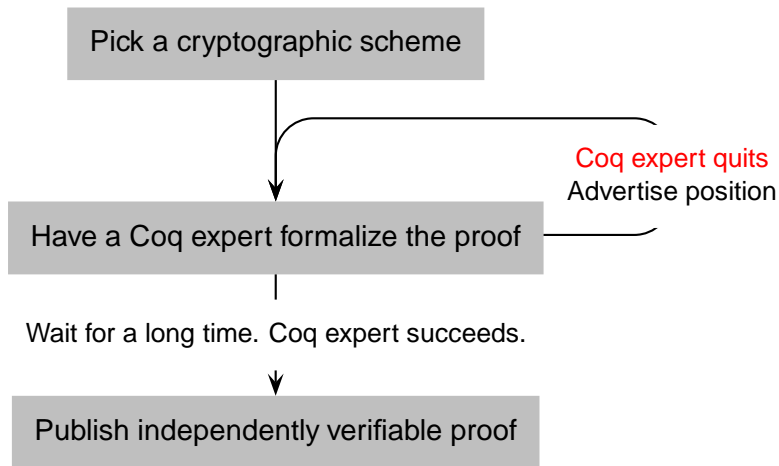
How much time is *enough*?

Verifiable security



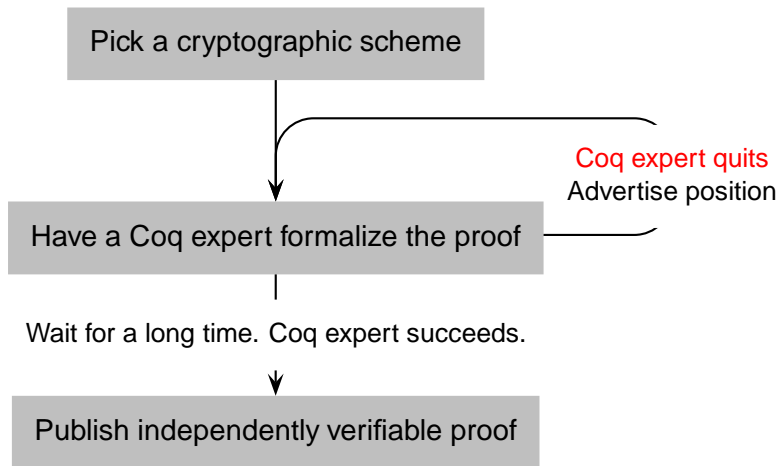
1 month, 3 months, 6 months, 1 year, 2 years?

Verifiable security



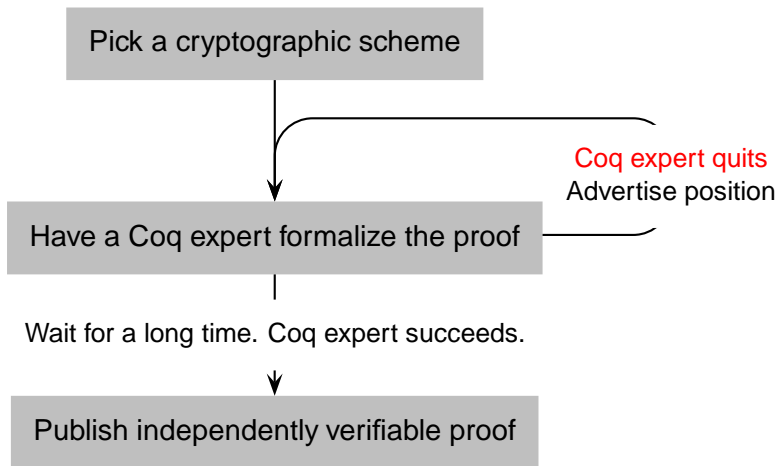
It took **3 monthes** to verify the EF-CMA proof of FDH

Verifiable security



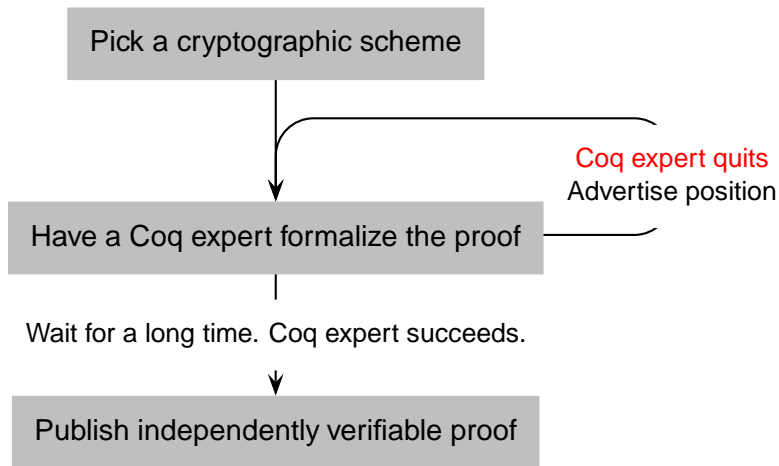
Ok, let's say **4 monthes** to be on the safe side

Verifiable security



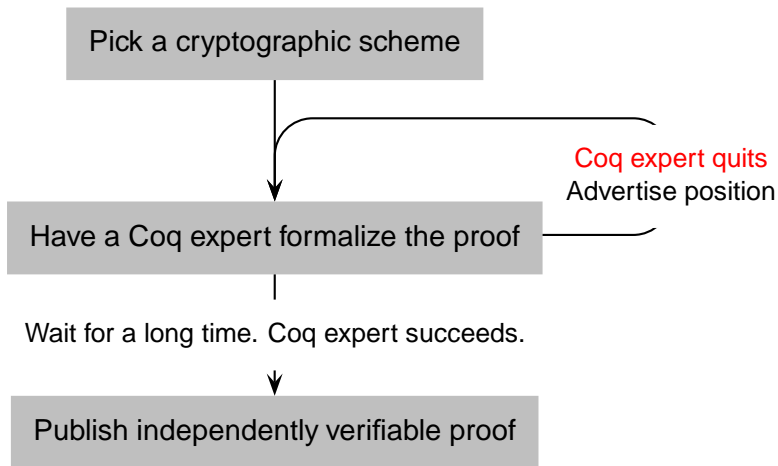
It took **6 monthes** to verify the IND-CCA proof of OAEP

Verifiable security



Can't we do better?

Verifiable security



Think so!

What's decidable about cryptographic proofs?

Can we automatically decide whether a relational Hoare tuple $\models G_1 \sim G_2 : P \Rightarrow Q$ is valid?

A strategy

- Generate verification conditions
- Send proof obligations to an automatic prover

Issues:

- Probabilistic assignment
- Relational judgment
- Proof objects
 - could be handled by certified or certifying SMT solvers
 - in the meantime, one can adopt a believing approach

Verification condition generation

Work in progress

One can generate proof obligations for judgments

$$\models G_1 \sim G_2 : P \Rightarrow Q$$

Intuitively use a one-sided rule for all cases, except:

- procedure calls (procedures have relational specs!):
 - use two-sided rules (needs call graphs to be similar)
 - inlining if the call relation is well-founded
- random assignment:
 - make programs in single static random assignments
 - perform eager samplings (for absolutely terminating programs)
 - deal with random sampling by existence of bijection

Preliminary experiments are encouraging!

Computational Indistinguishability Logic

Joint work with Marion Daubignard, Bruce Kapron and Yassine Lakhnech

- Foundational motivation: capture the essence of cryptographic proofs at a high level of abstraction
 - in a language-independent setting
 - with sufficient generality to cover e.g. (Pointcheval 04)
 - through a small set of rules
 - using established tools
 - bisimulations, contexts, determinization
- Practical benefits for CertiCrypt:
 - Sketches: generate proof obligations from overview of proof
 - Generic lemmas (cf Hashed ElGamal)
- Formalization ongoing (Pierre Corbineau, Christine Paulin)
- Proof of OAEP developed both in CertiCrypt and CIL

Proof sketches

Generate from proof sketch a verified statement of the form:

$$\Gamma_{i_1} \wedge \dots \wedge \Gamma_{i_k} \implies \Pr_{\mathbf{Game}G_0}[E_0] \leq p \left(\epsilon_1, \dots, \epsilon_\ell, \Pr_{\mathbf{Game}G_{i_1}}[\mathbf{bad}_{i_1}], \dots, \Pr_{\mathbf{Game}G_{i_m}}[\mathbf{bad}_{i_m}] \right)$$

where:

- $\Gamma_{i_1} \dots \Gamma_{i_k}$ assert the validity of unproved transitions
- p is a polynomial
- $\epsilon_1 \dots \epsilon_\ell$ are probabilities drawn from security hypotheses
- $\Pr_{\mathbf{Game}G_{i_1}}[\mathbf{bad}_{i_1}], \dots, \Pr_{\mathbf{Game}G_{i_m}}[\mathbf{bad}_{i_m}]$ are probabilities of failure events

Conclusion

- Increasing abstraction and automation will hopefully make verifiable security a reasonable and profitable alternative for cryptographers
- Will cryptographers adopt verifiable security tools?

Further work

CertiCrypt is also a good starting point to formalize

- automatic proof methods
- type systems for computational information flow
- computational soundness theorems
- randomized algorithms

To learn more about CertiCrypt

- *Formal certification of ElGamal encryption*
Formal Aspects in Security and Trust, FAST 2008
- *Formal certification of code-based cryptographic proofs*
Principles of Programming Languages, POPL 2009
- *Formally certifying the security of digital signature schemes*
Security & Privacy, S&P 2009
- *A Machine-Checked Formalization of Sigma-Protocols*
Computer Security Foundations, CSF 2010
- *Programming Language Techniques for Cryptographic Proofs*
Interactive Theorem Proving, ITP 2010
- *Beyond Provable Security: Verifiable IND-CCA Security of OAEP* submitted