

# Cryptographically Sound Security Proofs for Basic and Public-Key Kerberos <sup>★</sup>

M. Backes<sup>1</sup>, I. Cervesato<sup>2</sup>, A.D. Jaggard<sup>3</sup>, A. Scedrov<sup>4</sup>, and J.-K. Tsay<sup>4</sup>

<sup>1</sup> Saarland University

`backes@cs.uni-sb.de`

<sup>2</sup> Deductive Solutions

`iliano@deductivesolutions.com`

<sup>3</sup> Tulane University

`adj@math.tulane.edu`

<sup>4</sup> University of Pennsylvania

`{scedrov|jetsay}@math.upenn.edu`

**Abstract.** We present a computational analysis of basic Kerberos and Kerberos with public-key authentication (PKINIT) in which we consider authentication and key secrecy properties. Our proofs rely on the Dolev-Yao style model of Backes, Pfitzmann and Waidner, which allows for mapping results obtained symbolically within this model to cryptographically sound proofs if certain assumptions are met. This is the most complex fragment of an industrial protocol that has yet been verified at the computational level. Considering a recently fixed version of PKINIT, we extend symbolic correctness results we previously attained in the Dolev-Yao model to cryptographically sound results in the computational model.

## 1 Introduction

Cryptographic protocols have traditionally been verified in one of two ways: the first, known as the Dolev-Yao or symbolic approach, abstracts cryptographic concepts into an algebra of symbolic messages [25]; the second, known as the computational or cryptographic approach, retains the concrete view of messages as bitstrings and cryptographic operations as algorithmic mappings between bitstrings, while drawing security definitions from complexity theory [16, 26, 27].

---

<sup>★</sup> Backes was partially supported by the German Research Foundation (DFG) under grant 3194/1-1. Cervesato was partially supported by ONR under Grant N00014-01-1-0795. Jaggard was partially supported by NSF Grants DMS-0239996 and CNS-0429689, and by ONR Grant N00014-05-1-0818. Scedrov was partially supported by OSD/ONR CIP/SW URI “Software Quality and Infrastructure Protection for Diffuse Computing” through ONR Grant N00014-01-1-0795 and OSD/ONR CIP/SW URI “Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing” through ONR Grant N00014-04-1-0725. Additional support from NSF Grants CNS-0429689 and CNS-0524059. Tsay was partially supported by ONR Grant N00014-01-1-0795 and NSF grant CNS-0429689.

While proofs in the computational approach (with its much more comprehensive adversary model) entail stronger security guarantees, verification methods based on the Dolev-Yao abstraction have become efficient and robust enough to tackle large commercial protocols, often even automatically [1, 15, 18, 34].

Kerberos, a widely deployed protocol that allows a user to authenticate herself to multiple end servers based on a single login, constitutes one of the most important examples that have been formally analyzed so far within the Dolev-Yao approach. Kerberos 4, which was then the prevalent version, was verified using the Isabelle theorem prover [15]. The currently predominant version, Kerberos 5 [39], has been extensively analyzed using the Dolev-Yao approach. This analysis of Kerberos 5 showed that: a detailed specification of the core protocol enjoys the expected authentication and secrecy properties except for some relatively innocuous anomalies [18]; “cross-realm” authentication in Kerberos is correct when compared against its specification but has weaknesses in practice [21]; and discovered a serious attack against the then-current specification of the public-key extension (PKINIT) of Kerberos [20]. The discovery of the attack on PKINIT led to an immediate correction of the specification and a security bulletin and patch for Microsoft Windows [36].

The proofs for both Kerberos 5 as well as the fixes to PKINIT are restricted to the Dolev-Yao approach, and currently there does not exist a theorem which allows for carrying the results of existing proofs of Kerberos over to the cryptographic domain with its much more comprehensive adversary. Thus, despite the extensive research dedicated to the Kerberos protocol, and despite its tremendous importance in practice, it is still an open question whether an actual implementation of Kerberos based on provably secure cryptographic primitives is secure under cryptographic security definitions. We close this gap (at least partially) by providing the first security proof of the core aspects of the Kerberos protocol in the computational approach. More precisely, we show that core parts of Kerberos 5 are secure against arbitrary active attacks if the Dolev-Yao-based abstraction of the employed cryptography is implemented with actual cryptographic primitives that satisfy the commonly accepted security notions under active attacks, e.g., IND-CCA2 for public-key encryption.

Obviously, establishing a proof in the computational approach presupposes dealing with cryptographic details such as computational restrictions and error probabilities, hence one naturally assumes that our proof heavily relies on complexity theory and is far out of scope of current proof tools. However, our proof is not performed from scratch in the cryptographic setting, but based on the Dolev-Yao style model of Backes, Pfitzmann, and Waidner [8, 12, 13] (called the *BPW model* henceforth), which provides cryptographically faithful symbolic abstractions of cryptographic primitives, i.e., the abstractions can be securely implemented using actual cryptography. Thus our proof itself is symbolic in nature, but refers to primitives from the BPW model. Kerberos is the largest and most complex protocol whose cryptographic security has so far been inferred from a proof in this Dolev-Yao style approach. Earlier proofs in this approach were mainly for small examples of primarily academic interest, e.g., the

Needham-Schroeder-Lowe, the Otway-Rees, and the Yahalom protocols [4, 7, 11]; some similar work has been done on industrial protocols, e.g., [29], although none that are as complex as Kerberos. We furthermore analyze the recently fixed version of PKINIT and derive computational guarantees for it from a symbolic proof based on the BPW model. Finally we also draw some lessons learned in the process, which highlight areas where to focus research in order to simplify the verification of large commercial protocols with computational security guarantees. In particular it would be desirable to devise suitable proof techniques based on the BPW model for splitting large protocols into smaller pieces which can then be analyzed modularly while still retaining the strong link between the Dolev-Yao and the computational approach.

## 1.1 Related Work

Early work on linking Dolev-Yao models and cryptography [2, 3, 28] only considered passive attacks, and therefore cannot make general statements about protocols. A cryptographic justification for a Dolev-Yao model in the sense of simulatability [40], i.e., under active attacks and within arbitrary surrounding interactive protocols, was first given in [12] with extensions in [8, 13]. Based on that Dolev-Yao model, the well-known Needham-Schroeder-Lowe, Otway-Rees, and Yahalom protocols were proved secure in [4, 7, 11]. All these protocols are considerably simpler than Kerberos, which we analyze in this paper, and arguably of much more limited practical interest. Some work has been done on industrial protocols, such as 802.11i [29], although Kerberos is still a much more complex protocol.

Laud [33] has presented a cryptographic underpinning for a Dolev-Yao model of symmetric encryption under active attacks. His work is directly connected with a formal proof tool, but it is specific to certain confidentiality properties and protocol classes. Herzog et al. [30] and Micciancio and Warinschi [35] have also given a cryptographic underpinning under active attacks. Their results are narrower than those in [12] since they are specific for public-key encryption and certain protocol classes, but consider slightly simpler real implementations. Cortier and Warinschi [22] have shown that symbolically secret nonces are also computationally secret, i.e., indistinguishable from a fresh random value given the view of a cryptographic adversary. Backes and Pfitzmann [9] and Canetti and Herzog [19] have established new symbolic criteria for proving a key cryptographically secret. We stress that none of this work is comprehensive enough to infer computational security guarantees of Kerberos based on an existing symbolic proof; either they are missing suitable cryptographic primitives or rely on slightly changed symbolic abstractions, e.g., as in [12].

Finally, there is also work on formulating syntactic calculi for dealing with probability and polynomial-time considerations and encoding them into proof tools, in particular [17, 23, 32, 37]. This is orthogonal to the work of justifying Dolev-Yao models.

## 1.2 Structure of the Paper

We start in Sect. 2 with a review of Kerberos and its public-key extension PKINIT. In Sect. 3, we recall the Dolev-Yao style model of Backes, Pfitzmann, and Waidner (e.g., [6, 8, 13, 14]), and apply it to the specification of Kerberos 5 and Public-key Kerberos (i.e., Kerberos with PKINIT). Section 4 proves security results for these protocols and lift them to the computational level. Finally, Sect. 5 summarizes this effort and outlines areas of future work.

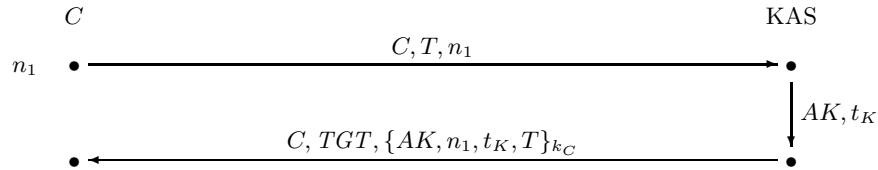
## 2 Kerberos 5 and its Public-Key Extension

The Kerberos protocol [38, 39] allows a legitimate user to log on to her terminal once a day (typically) and then transparently access all the networked resources she needs for the rest of that day. Each time she wants to, e.g., retrieve a file from a remote server, a Kerberos client running on her behalf securely handles the required authentication. The client acts behind the scenes, without any user intervention.

Kerberos comprises three subprotocols: the initial round of authentication, in which the client obtains a credential that might be good for a full day; the second round of authentication, in which she presents her first credential in order to obtain a short-term credential (five-minute lifetime) to use a particular network service; and the client’s interaction with the network service, in which she presents her short-term credential in order to negotiate access to the service.

In the core specification of Kerberos 5 [39], all three subprotocols use symmetric (shared-key) cryptography. Since the initial specification of Kerberos 5, the protocol has been extended by the definition of an alternate first round which uses asymmetric (public-key) cryptography. This new subprotocol, called PKINIT [31], may be used in two modes: “public-key encryption mode” and “Diffie-Hellman (DH) mode.” In recent work [20], we showed that there was an attack against the then-current draft specification of PKINIT when public-key encryption mode was used and then symbolically proved the security of the specification as it was revised in response to our attack. Here we study both basic Kerberos (without PKINIT) and the public-key mode of PKINIT as it was revised to prevent our attack.

**Kerberos Basics** The client process—usually acting for a human user—interacts with three other types of principals when using Kerberos 5 (with or without PKINIT). The client’s goal is to be able to authenticate herself to various application servers (e.g., email, file, and print servers). This is done by obtaining a “ticket-granting ticket” (TGT) from a “Kerberos Authentication Server” (KAS) and then presenting this to a “Ticket-Granting Server” (TGS) in order to obtain a “service ticket” (ST), the credential that the client uses to authenticate herself to the application server. A TGT might be valid for a day, and may be used to obtain several STs for many different application servers from the TGS, while a single ST is valid for a few minutes (although it may be used repeatedly) and is



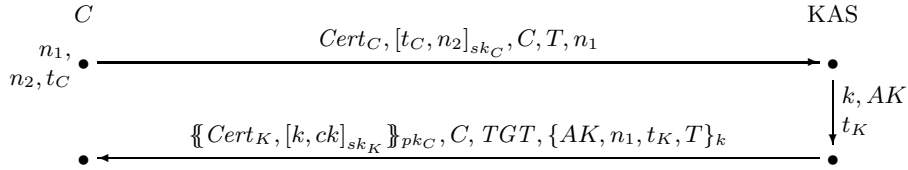
**Fig. 1.** Message Flow in the Traditional AS Exchange, where  $TGT = \{AK, C, t_K\}_{k_T}$ .

used for a single application server. The KAS and the TGS are together known as the “Key Distribution Center” (KDC).

The client’s interactions with the KAS, TGS, and application servers are called the Authentication Service (AS), Ticket-Granting (TG), and Client-Server (CS) exchanges, respectively. We will describe the AS exchange separately for basic Kerberos and PKINIT; as PKINIT does not modify the other subprotocols, we only need to describe them once.

**The Traditional AS Exchange** The abstract structure of the AS exchange is given in Fig. 1. A client  $C$  generates a fresh nonce  $n_1$  and sends it, together with her own name and the name  $T$  of the TGS for whom she desires a TGT, to the KAS  $K$ . This message is called the AS\_REQ message [39]. The KAS responds by generating a fresh authentication key  $AK$  for use between the client and the TGS and sending an AS\_REP message to the client. Within this message,  $AK$  is sent back to the client in the encrypted message component  $\{AK, n_1, t_K, T\}_{k_C}$ ; this also contains the nonce from the AS\_REQ, the KAS’s local time  $t_K$ , and the name of the TGS for whom the TGT was generated. (The  $AK$  and  $t_K$  to the right of the figure illustrate that these values are new between the two messages.) This component is encrypted under a long-term key  $k_C$  shared between  $C$  and the KAS; this key is usually derived from the user’s password. This is the only time that this key is used in a standard Kerberos run because later exchanges use freshly generated keys.  $AK$  is also included in the ticket-granting ticket sent alongside the message encrypted for the client. The TGT consists of  $AK, C, t_K$ , where  $t_K$  is  $K$ ’s local time, encrypted under a long-term key  $k_T$  shared between the KAS and the TGS named in the request. The computational model we use here does not support timestamps, so we will treat these as nonces; this does not compromise our analysis as the timestamps that we include here are used like nonces. Once the client has received this reply, she may undertake the Ticket-Granting exchange.

It should be noted that the actual AS exchange, as well as the other exchanges in Kerberos, is more complex than the abstract view given here. We refer the reader to [39] for the complete specification of Kerberos 5, [31] for the specification of PKINIT, and [18] for a formalization of Kerberos at an intermediate level of detail.



**Fig. 2.** Message flow in the fixed version of PKINIT, where  $TGT = \{AK, C, t_K\}_{k_T}$ .

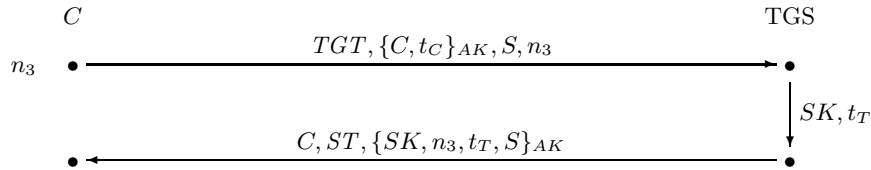
**The AS Exchange in PKINIT** PKINIT [31] is an extension to Kerberos 5 that uses public key cryptography to avoid shared secrets between a client and KAS; it modifies the AS exchange but not other parts of the basic Kerberos 5 protocol. The long-term shared key ( $k_C$ ) in the traditional AS exchange is typically derived from a password, which limits the strength of the authentication to the user’s ability to choose and remember good passwords; PKINIT does not use  $k_C$  and thus avoids this problem. Furthermore, if a public key infrastructure (PKI) is already in place, PKINIT allows network administrators to use it rather than expending additional effort to manage users’ long-term keys as in traditional Kerberos.

In PKINIT, the client  $C$  and the KAS possess independent public/secret key pairs,  $(pk_C, sk_C)$  and  $(pk_K, sk_K)$ , respectively. Certificate sets  $Cert_C$  and  $Cert_K$  issued by a PKI independent from Kerberos are used to testify of the binding between each principal and her purported public key. This simplifies administration as authentication decisions can now be made based on the trust the KDC holds in just a few known certification authorities within the PKI, rather than keys individually shared with each client (local policies can, however, still be installed for user-by-user authentication). Dictionary attacks are defeated as user-chosen passwords are replaced with automatically generated asymmetric keys.

PKINIT resembles the basic AS exchange in that the KAS generates a fresh key  $AK$  for the client and TGS to use, and then the KAS transmits  $AK$  and the TGT to the client. In public-key encryption mode, attacked and fixed in [20] and now analyzed here, the key pairs are used for both signature and encryption. The latter is designed to (indirectly) protect the confidentiality of  $AK$ , while the former ensures its integrity.

Figure 2 illustrates the AS exchange when the fixed version (which defends against the attack of [20]) of PKINIT is used. Here we use  $[m]_{sk}$  for the digital signature of message  $m$  with secret key  $sk$ ,  $\{\{m\}\}_{pk}$  for the encryption of  $m$  with the public key  $pk$ , and  $\{m\}_k$  for the encryption of  $m$  with the symmetric key  $k$ .

The first line of Fig. 2 shows our formalization of the AS\_REQ message that a client  $C$  sends to a KAS  $K$  when using PKINIT. The last part of the message— $C, T, n_1$ —is exactly as in the traditional AS\_REQ. The new data added by PKINIT are the client’s certificates  $Cert_C$  and her signature (with her secret key  $sk_C$ ) over a timestamp  $t_C$  and another nonce  $n_2$ .

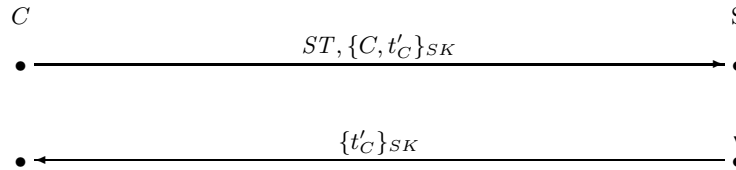


**Fig. 3.** Message flow in the TGS exchange, where  $TGT = \{AK, C, t_K\}_{k_T}$  and  $ST = \{SK, C, t_T\}_{k_S}$ .

The second line in Fig. 2 shows our formalization of  $K$ 's response, which is more complex than in basic Kerberos. The last part of the message— $C, TGT, \{AK, n_1, t_K, T\}_k$ —is very similar to  $K$ 's reply in basic Kerberos; the difference is that the symmetric key  $k$  protecting  $AK$  is now freshly generated by  $K$  and is not a long-term shared key. Because  $k$  is freshly generated for the reply, it must be communicated to  $C$  before she can learn  $AK$ . PKINIT does this by adding the message  $\{\{Cert_K, [k, ck]_{sk_K}\}_{pk_C}\}_{pk_C}$ . This contains  $K$ 's certificates and his signature, using his secret key  $sk_K$ , over  $k$  and a keyed hash  $ck$  ('checksum' in the language of [39]) taken over the entire request from  $C$  using the key  $k$ ; all of this is encrypted under  $C$ 's public key  $pk_C$ . The keyed hash  $ck$  binds this response to the client's request and was added in response to the attack we discovered and reported in [20].

**The Later Exchanges** After the client  $C$  has obtained the key  $AK$  and the TGT, either through the basic AS exchange or the PKINIT AS exchange, she then initiates the TGS exchange, shown in Fig. 3. The first line shows our formalization of the client's request, called a TGS\_REQ message; it contains the TGT (which is opaque to the client), an *authenticator*  $\{C, t_C\}_{AK}$ , the name of the server  $S$  for which  $C$  desires a service ticket, and  $C$ 's local time. Once the TGS receives this message, he decrypts the TGT to learn  $AK$  and uses this to decrypt the authenticator. Assuming his local policies for granting a service ticket are satisfied (while we do not model these here, they might include whether the request is sufficiently fresh), the TGS produces a fresh key  $SK$  for  $C$  and  $S$  to share and sends this back to the client in a TGS\_REP message. The form of this message is essentially the same as the AS\_REP message from the KAS to  $C$ : it contains a ticket (now the service ticket, or  $ST, \{SK, C, t_T\}_{k_S}$  instead of the TGT) encrypted for the next server (now  $S$  instead of  $T$ ) and encrypted data for  $C$  (now encrypted under  $AK$  instead of  $k_C$ ).

Finally, after using the AS exchange to obtain the key  $SK$  and the  $ST$ , the client may use the CS exchange to authenticate herself to the end server. Figure 4 shows this exchange, including the optional reply from the server that authenticates this server to the client. The client  $C$  starts by sending a message (AP\_REQ) that is similar to the TGS\_REQ message of the previous round: it contains the (service) ticket and an authenticator ( $\{C, t'_C\}_{SK}$ ) that is encrypted under the key contained in the  $ST$ . The server  $S$  simply responds with



**Fig. 4.** Message flow in the CS exchange, where  $ST = \{SK, C, t_T\}_{k_S}$ .

an AP\_REP message  $\{t'_C\}_{SK}$  containing the timestamp from the authenticator encrypted under the key from the ST.

**Attack on PKINIT** The attack that we found against the then-current specification of PKINIT was reported in [20]. This attack was possible because, at the time, the reply from the KAS to the client contained  $[k, n_2]_{sk_K}$  in place of  $[k, ck]_{sk_K}$ . In particular, the KAS did not sign any data that depended upon the client's name. This allowed an attacker to copy a message from  $C$  to the KAS, use this data in her own request to the KAS, read the reply from the KAS, and then send this reply to  $C$  as though it was generated by the KAS for  $C$  (instead of for the attacker). The effect of this attack was that the attacker could impersonate the later servers (TGS and application servers) to the client, or she could let the client continue the authentication process while the attacker gains knowledge of all new keys shared by the client and various servers. In the latter variation, the client would be authenticated as the attacker and not as  $C$ .

**Security Properties** We now summarize the security properties that we prove here at the symbolic level for both basic Kerberos and Kerberos with PKINIT; the implications on the computational level are discussed in the subsequent sections. We have proved similar properties in symbolic terms using a formalization in MSR for basic Kerberos [18] and for the AS exchange when PKINIT is used [20]. The first property we prove here concerns the secrecy of keys, a notion that is captured formally as Def. 1 in Sect. 4. This property may be summarized as follows.

*Property 1 (Key secrecy).* For any honest client  $C$  and honest server  $S$ , if the TGS  $T$  generates a symmetric key  $SK$  for  $C$  and  $S$  to use (in the  $CS$ -exchange), then the intruder does not learn the key  $SK$ .

The second property we study here concerns entity authentication, formalized as Def. 2 in Sect. 4. This property may be summarized as follows.

*Property 2 (Authentication properties).*

- i. If a server  $S$  completes a run of Kerberos, apparently with  $C$ , then earlier:  $C$  started the protocol with some KAS to get a ticket-granting ticket and then requested a service ticket from some TGS.

- ii. If a client  $C$  completes a run of Kerberos, apparently with server  $S$ , then  $S$  sent a valid AP\_REP message to  $C$ .

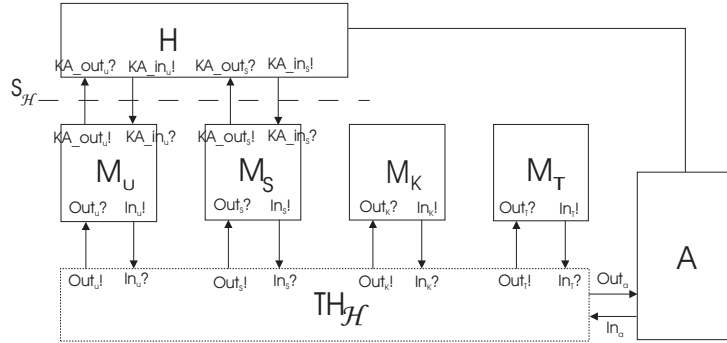
Theorem 1 below shows that these properties hold for our symbolic formalizations of basic and public-key Kerberos in the BPW model. Theorem 2 shows that the authentication property holds as well for cryptographic implementations of these protocols if provably secure primitives are used; the standard cryptographic definition of key secrecy however turns out not to hold for cryptographic implementations of Kerberos, which we further investigate below. Because authentication can be shown to hold for Kerberos with PKINIT, it follows that at the level of cryptographic implementation, the fixed specification of PKINIT does indeed defend against the attack reported in [20].

### 3 The BPW Model

#### 3.1 Review of the BPW Model

The BPW model introduced in [14] offers a deterministic Dolev-Yao style formalism of cryptographic protocols with commands for a vast range of cryptographic operations such as public-key, symmetric encryption/decryption, generation and verification of digital signatures as well as message authentication codes, and nonce generation. Every protocol participant is assigned a machine (an I/O automaton), which is connected to the machines of other protocol participants and which executes the protocol for its user by interacting with the other machines (see Fig. 5). In this reactive scenario, semantics is based on state, i.e., of who already knows which terms. The state is here represented by an abstract “database” and handles to its entries: Each entry (denoted  $D[j]$ ) of the database has a type (e.g., “signature”) and pointers to its arguments (e.g., “private key” and “message”). This corresponds to the way Dolev-Yao terms are represented. Furthermore, each entry in the abstract database also comes with handles to participants who have access to that entry. These handles determine the state. The BPW model does not allow cheating: only if a participant has a handle to the entry  $D[j]$  itself or to the right entries that could produce a handle to  $D[j]$  can the participant learn the term stored in  $D[j]$ . For instance, if the BPW model receives a command, e.g., from a user machine, to encrypt a message  $m$  with key  $k$ , then it makes a new abstract database entry for the ciphertext with a handle to the participant that sent the command and pointers to the message and the key as arguments; only if a participant has handles to the ciphertext and also to the key can the participant ask for decryption. Furthermore, if the BPW model receives the same encryption command a second time then it will generate a new (different) entry for the ciphertext. This meets the fact that secure encryption schemes are necessarily probabilistic. Entries are made known to other participants by a send command, which adds handles to the entry.

The BPW model is based on a detailed model of asynchronous reactive systems introduced in [40] and is represented as a deterministic machine  $\text{TH}_{\mathcal{H}}$  (also an I/O automaton), called *trusted host*, where  $\mathcal{H} \subset \{1, \dots, n\}$  denotes the set of



**Fig. 5.** Overview of the Kerberos symbolic system

honest participants out of all  $m$  participants. This machine executes the commands from the user machines, in particular including the commands for cryptographic operations. A *system* consists of several possible *structures*. A structure consists of a set  $\hat{M}$  of connected correct user machines and a subset  $S$  of the free ports, i.e.,  $S$  is the user interface of honest users. In order to analyze the security of a structure  $(\hat{M}, S)$ , an arbitrary probabilistic polynomial-time *user* machine **H** is connected to the user interface  $S$  and a polynomial-time *adversary* machine **A** is connected to all the other ports and **H**. This completes a structure into a *configuration* of the system (see Fig. 5). The machine **H** represents all users. A configuration is a runnable system, i.e., for each security parameter  $k$ , which determines the input lengths (including the key length), one gets a well-defined probability space of *runs*. To guarantee that the system is polynomially bounded in the security parameter, the BPW model maintains length functions on the entries of the abstract database. The *view* of **H** in a run is the restriction to all inputs and outputs that **H** sees at the ports it connects to, together with its internal states. Formally one defines the view  $view_{conf}(\mathbf{H})$  of **H** for a configuration  $conf$  to be a family of random variables  $X_k$  where  $k$  denotes the security parameter. For a given security parameter  $k$ ,  $X_k$  maps runs of the configuration to a view of **H**.

Corresponding to the BPW model, there exists a cryptographic implementation of the BPW model and a computational system, in which honest participants also operate via handles on cryptographic objects. However, the objects are now bitstrings representing real cryptographic keys, ciphertexts, etc., acted upon by interactive polynomial-time Turing machines (instead of the symbolic machines and the trusted host). The implementation of the commands now uses provably secure cryptographic primitives according to standard cryptographic definitions (with small additions like type tagging and additional randomization). In [8, 12–14] it was established that the cryptographic implementation of the BPW model is *at least as secure as* the BPW model, meaning that whatever an active adversary can do in the implementation can also be achieved by another adversary in the BPW model, or the underlying cryptography can be broken. More

formally, a system  $Sys_1$  being at least as secure as another system  $Sys_2$  means that for all probabilistic polynomial-time user  $H$ , for all probabilistic polynomial-time adversary  $A_1$  and for every computational structure  $(\hat{M}_1, S) \in Sys_1$ , there exists a polynomial-time adversary  $A_2$  on a corresponding symbolic structure  $(\hat{M}_2, S) \in Sys_2$  such that the view of  $H$  is computationally indistinguishable in both configurations. This captures the cryptographic notion of *reactive simulatability*.

### 3.2 Public-key Kerberos in the BPW Model

We now model the Kerberos protocol in the framework of [14] using the BPW model. We write “ $:=$ ” for deterministic assignment, “ $=$ ” for testing for equality and “ $\leftarrow$ ” for probabilistic assignment.

The descriptions of the symbolic systems of Kerberos 5 and PKINIT are very similar, with the difference that the user machines follow different algorithms for the two protocols. We denote Kerberos with PKINIT by “PK,” and basic Kerberos by “K5.” If we let  $Kerb \in \{PK, K5\}$  then, as described in Sect. 3.1, for each user  $u \in \{1, \dots, n\}$  there is a *protocol machine*  $M_u^{Kerb}$  which executes the protocol for  $u$ . There are also protocol machines for the KAS  $K$  and the TGT  $T$ , denoted by  $M_K^{Kerb}$  and  $M_T^{Kerb}$ . Furthermore, if  $S_1, \dots, S_l$  are the servers in  $T$ ’s ‘realm<sup>5</sup>’, then there are server machines  $M_S^{Kerb}$  for  $S \in \{S_1, \dots, S_l\}$ . Each user machine is connected to the user via ports: A port for outputs to the user and a port for inputs from the user, labeled  $KA\_out_u!$  and  $KA\_in_u?$ , respectively (“KA” for “Key sharing and Authentication”). The ports for the server machines are labeled similarly (see Fig. 5).

The behavior of the protocol machines is described in detail in [5]. In the following, we comment on two algorithms of PKINIT (Fig. 6 and Fig. 7). If, for instance, a protocol machine  $M_u^{PK}$  receives a message  $(new\_prot, PK, K, T)$  at  $KA\_in_u?$  then it will execute Algorithm 1A (Fig. 6) to start a protocol run. We give a description below. The state of the protocol machine  $M_u^{Kerb}$  consists of the bitstring  $u$  and the sets  $Nonce_u$ ,  $Nonce2_u$ ,  $TGTicket$ , and  $Session\_Keys_{S_u}$ , in which  $M_u^{Kerb}$  stores nonces, ticket-granting tickets, and the session keys for server  $S$ , respectively. This is the information a client needs to remember during a protocol run.

Only the machines of honest users  $u \in \{1, \dots, n\}$  and honest servers  $S \in \{S_1, \dots, S_l\}$  will be present in the protocol run, in addition to the machines for  $K$  and  $T$ . The others are subsumed in the adversary. We denote by  $\mathcal{H} \subset \{1, \dots, n, K, T, S_1, \dots, S_l\}$  the honest participants, i.e., for  $v \in \mathcal{H}$  the machine  $M_v^{Kerb}$  is guaranteed to run correctly. And we assume that KAS  $K$  and TGS  $T$  are always honest, i.e.,  $K, T \in \mathcal{H}$ .

Furthermore, given a set  $\mathcal{H}$  of honest participants, with  $\{K, T\} \subset \mathcal{H} \subset \{1, \dots, n, K, T, S_1, \dots, S_l\}$  the user interface of public-key Kerberos will be the set  $S_{\mathcal{H}} := \{KA\_out_u!, KA\_in_u? \mid u \in \mathcal{H} \setminus \{K, T\}\}$ . The symbolic system is the

<sup>5</sup> I.e., administrative domain; we do not consider cross-realm authentication here, although it has been analyzed symbolically in [21]

set  $Sys^{\text{Kerb}, \text{symb}} := \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})\}$ . Note that, since we are working in an asynchronous system, we are replacing protocol timestamps by arbitrary messages that we assume are known to the participants generating the timestamps (e.g. nonces). All algorithms should immediately abort if a command to the BPW model yields an error, e.g., if a decryption request fails.

**Notation** The entries of the database  $D$  are all of the form  $(ind, type, arg, hnd_{u_1}, \dots, hnd_{u_m}, hnd_a, len)$ , where  $\mathcal{H} = \{u_1, \dots, u_m\}$ . We denote by  $\downarrow$  an error element available to all ranges and domains of all functions and algorithms. So, e.g.,  $hnd_a = \downarrow$  means the adversary does not have a handle to the entry. For entries  $x \in D$ , the *index*  $x.ind \in \mathcal{INDS}$  consecutively numbers all entries in  $D$ . The set  $\mathcal{INDS}$  is isomorphic to  $\mathbb{N}$  and is used to distinguish index arguments. We write  $D[i]$  for the selection  $D[ind = i]$ , i.e., it is used as a primary key attribute of the database. The entry  $x.type \in \text{typeset} = \{\text{auth}, \text{cert}, \text{enc}, \text{nonce}, \text{list}, \text{pke}, \text{pkse}, \text{sig}, \text{ske}, \text{skse}, \}$  identifies the type of  $x$ . Here *ske/pke* is a private/public key pair and *skse* is a symmetric key which comes with a ‘public’ key *pkse*. This “public key identifier” *pkse* cannot be used for any cryptographic operation but works as a pointer to *skse* instead (see [7] for a more detailed explanation). The entry  $x.arg = (a_1, \dots, a_j)$  is a possibly empty list of arguments. Many values  $a_i$  are in  $\mathcal{INDS}$ .  $x.hnd_u \in \mathcal{HANDS} \cup \{\downarrow\}$  for  $u \in \mathcal{H} \cup \{a\}$  are handles by which  $u$  knows this entry. We always use a superscript “*hnd*” for handles.  $x.len \in \mathbb{N}_0$  denotes the “length” of the entry; it is computed by applying length functions (mentioned in Sect. 3.1).

Initially,  $D$  is empty.  $\text{TH}_{\mathcal{H}}$  has a counter  $size \in \mathcal{INDS}$  for the current size of  $D$ . For the handle attributes, it has counters  $currhnd_u$  initially 0. First we need to add the symmetric keys shared exclusively by  $K$  and  $T$ ,  $S$  and  $T$ . Public-key Kerberos uses certificates; therefore, in this case all users need to know the public key for certificate authorities and have their own public-key certificates signed by a certificate authority. For simplicity we use only one certificate authority  $CA$ . Therefore, we add to  $D$  an entry for the public key of  $CA$  with handles to all users (i.e., to all user machines). For every user we add an entry for the certificate of that user signed by the certificate authority with a handle to the user (machine). In the case of Kerberos 5, we are adding entries for the key  $k_u$  shared exclusively by  $K$  and  $u$ , for all users  $u$ .

**Example of Algorithms** Due to space constraints we are only going to examine PKINIT (Fig. 2) and explain the steps of its Algorithms 1A and 2 (Fig. 6 and Fig. 7) which are more complex than the algorithms in Kerberos 5. For details on the definition of the used commands see [8, 13, 14]. For readability of the figures, we noted on the right (in curly brackets) to which terms in the more commonly used Dolev-Yao notation the terms in the algorithms correspond ( $\approx$ ).

*Protocol start of PKINIT.* In order to start a new PKINIT protocol, user  $u$  inputs  $(\text{new\_prot}, \text{PK}, K, T)$  at port  $\text{KA\_in}_u?$ . Upon such an input,  $M_u^{\text{PK}}$  runs Algorithm 1A (Fig. 6) which prepares and sends the  $\text{AS\_REQ}$  to  $K$  using the BPW model.  $M_u^{\text{PK}}$  generates symbolic nonces in steps 1A.1 and 1A.2 by sending

**A) Input:**(new\_prot, PK, K, T) at KA\_in<sup>u</sup>? .

1.  $n_{u,1}^{hnd}, t_u^{hnd} \leftarrow \text{gen\_nonce}()$
2.  $n_{u,2}^{hnd} \leftarrow \text{gen\_nonce}()$
3.  $l^{hnd} \leftarrow \text{list}(t_u^{hnd}, n_{u,2}^{hnd})$   $\{l \approx (t_C, n_2)\}$
4.  $s^{hnd} \leftarrow \text{sign}(sk_e^{hnd}, l^{hnd})$   $\{s \approx [t_C, n_2]_{sk_C}\}$
5.  $u^{hnd} \leftarrow \text{store}(u)$
6.  $T^{hnd} \leftarrow \text{store}(T)$
7.  $m_1^{hnd} \leftarrow \text{list}(cert_u^{hnd}, s^{hnd}, u^{hnd}, T^{hnd}, n_{u,1}^{hnd})$   $\{m_1 \approx Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1\}$
8.  $Nonce_u := Nonce_u \cup \{(n_{u,1}^{hnd}, m_1^{hnd}, K)\}$
9. send\_i(K, m\_1^{hnd})

**B) Input:**(continue\_prot, PK, T, S, AK<sup>hnd</sup>) at KS\_in<sup>u</sup>? for  $S \in \{S_1, \dots, S_l\}$

1. **if** ( $\nexists (TGT^{hnd}, AK^{hnd}, T) \in TGTicket_u$ ) **then**
2.     Abort
3. **end if**
4.  $z^{hnd} \leftarrow \text{list}(u^{hnd}, t_u^{hnd})$   $\{z \approx C, t_C\}$
5.  $auth^{hnd} \leftarrow \text{sym\_encrypt}(AK^{hnd}, z^{hnd})$   $\{auth \approx \{C, t_C\}_{AK}\}$
6.  $n_{u,3}^{hnd} \leftarrow \text{gen\_nonce}()$
7.  $Nonce2_u := Nonce2_u \cup \{(n_{u,3}^{hnd}, T, S)\}$
8.  $m_2^{hnd} \leftarrow \text{list}(TGT^{hnd}, auth^{hnd}, S^{hnd}, n_{u,3}^{hnd})$   $\{m_2 \approx TGT, \{C, t_C\}_{AK}, S, n_3\}$
9. send\_i(T, m\_2^{hnd})

**Fig. 6.** Algorithm 1 of Public-key Kerberos: Evaluation of inputs from the user (starting the AS and TG exchanges).

the command `gen_nonce()`. In step 1A.3 the command `list(-,-)` concatenates  $t_u$  and  $n_{u,2}$  into a new list that is signed in step 1A.4 with  $u$ 's private key. Since we are working in an asynchronous system, the timestamp  $t_u$  is approximated by some arbitrary message (e.g., by a nonce). The command `store(-)` in step 1A.5–6 makes entries in the database for the names of  $u$  and  $T$ . Handles for the names  $u$  and  $T$  are returned, which are added to a list in the next step.  $M_u^{\text{PK}}$  stores information in the set  $Nonce_u$ , which it will need later in the protocol to verify the message authentication code sent by  $K$ . In step 1A.8  $Nonce_u$  is updated. Finally, in step 1A.9 the AS\_REQ is sent over an insecure (“i” for insecure) channel.

*Behavior of the KAS  $K$  in PKINIT.* Upon input  $(v, K, i, m^{hnd})$  at port `out_K?` with  $v \in \{1, \dots, n\}$ , the machine  $M_K^{\text{PK}}$  runs Algorithm 2 (Fig. 7) which first checks if the message  $m$  is a valid AS\_REQ and then prepares and sends the corresponding AS\_REP. In order to verify that the input is a possible AS\_REQ, the types of the input message  $m$ 's components are checked in steps 2.1–2.5. The command `retrieve( $x_i^{hnd}$ )` in step 2.3 returns the bitstring of the entry  $D[hnd_u = x_i^{hnd}]$ . Next the machine verifies the received certificate  $x_1$  of  $v$  by checking the signature of the certificate authority  $CA$  (steps 2.6–2.10). Then the machine extracts the public key  $pke_v$  out of  $v$ 's certificate with the command `pk_of.cert(-)`

**Input:**  $(v, K, i, m^{hnd})$  at  $\text{out}_K?$  with  $v \in \{1, \dots, n\}$ .

1.  $x_i^{hnd} \leftarrow \text{list\_proj}(m^{hnd}, i)$  for  $i = 1, \dots, 5$
2.  $\text{type}_i \leftarrow \text{get\_type}(x_i^{hnd})$  for  $i = 1, 2, 5$        $\{x_1 \approx \text{Cert}_C, x_2 \approx [t_C, n_2]_{sk_C}, x_5 \approx n_1\}$
3.  $x_i \leftarrow \text{retrieve}(x_i^{hnd})$  for  $i = 3, 4$        $\{x_3 \approx C, x_4 \approx T\}$
4. **if**  $(\text{type}_1 \neq \text{cert}) \vee (\text{type}_2 \neq \text{sig}) \vee (\text{type}_5 \neq \text{Nonce}) \vee (x_3 \neq v) \vee (x_4 \neq T)$  **then**
5.     Abort
6. **end if**
7.  $v^{hnd} \leftarrow \text{store}(v)$
8.  $b \leftarrow \text{verify\_cert}(pke_{CA}^{hnd}, x_1^{hnd}, v^{hnd})$
9. **if**  $b = \text{false}$  **then**
10.     Abort
11. **end if**
12.  $pke_v^{hnd} \leftarrow \text{pk\_of\_cert}(pke_{CA}^{hnd}, x_1^{hnd})$
13.  $\text{type}_6 \leftarrow \text{get\_type}(pke_v^{hnd})$
14. **if**  $(\text{type}_6 \neq \text{pke})$  **then**
15.     Abort
16. **end if**
17.  $y_1^{hnd} \leftarrow \text{msg\_of\_sig}(x_2^{hnd})$        $\{y_1 \approx t_C, n_2\}$
18.  $b \leftarrow \text{verify}(x_2^{hnd}, pke_v^{hnd}, y_1^{hnd})$        $\{x_2 \approx [t_C, n_2]_{sk_C}\}$
19. **if**  $b = \text{false}$  **then**
20.     Abort
21. **end if**
22.  $y_{1i}^{hnd} \leftarrow \text{list\_proj}(y_1^{hnd}, i)$  for  $i = 1, 2$        $\{y_{11} \approx t_C, y_{12} \approx n_2\}$
23.  $\text{type}_{12} \leftarrow \text{get\_type}(y_{12}^{hnd})$
24. **if**  $(\text{type}_{12} \neq \text{nonce}) \vee ((y_{12}^{hnd}, \cdot) \in \text{Nonce3}_K)$  **then**
25.     Abort
26. **end if**
27.  $\text{Nonce3}_K := \text{Nonce3}_K \cup \{(y_{12}^{hnd}, v)\}$
28.  $k^{hnd} \leftarrow \text{gen\_symenc\_key}()$
29.  $AK^{hnd} \leftarrow \text{gen\_symenc\_key}()$
30.  $\text{auth}^{hnd} \leftarrow \text{auth}(k^{hnd}, m^{hnd})$        $\{\text{auth} \approx ck\}$
31.  $z_1^{hnd} \leftarrow \text{list}(k^{hnd}, \text{auth}^{hnd})$        $\{z_1 \approx k, ck\}$
32.  $s_2^{hnd} \leftarrow \text{sign}(ske_K^{hnd}, z_1^{hnd})$        $\{s_2 \approx [k, ck]_{sk_K}\}$
33.  $z_2^{hnd} \leftarrow \text{list}(\text{cert}_K^{hnd}, s_2^{hnd})$        $\{z_2 \approx \text{Cert}_K, [k, ck]_{sk_K}\}$
34.  $m_{21} \leftarrow \text{encrypt}(pke_K^{hnd}, z_2^{hnd})$        $\{m_{21} \approx \{\{\text{Cert}_K, [k, ck]_{sk_K}\}\}_{pk_C}\}$
35.  $z_3^{hnd} \leftarrow \text{list}(AK^{hnd}, x_3^{hnd}, t_K^{hnd})$        $\{z_3 \approx AK, C, t_K, T\}$
36.  $TGT^{hnd} \leftarrow \text{sym\_encrypt}(sk_{se_{K, x_4}}^{hnd}, z_3^{hnd})$        $\{TGT \approx \{AK, C, t_K\}_{k_T}\}$
37.  $z_4^{hnd} \leftarrow \text{list}(AK^{hnd}, x_5^{hnd}, t_K^{hnd}, x_4^{hnd})$        $\{z_4 \approx AK, n_1, t_K, T\}$
38.  $m_{24} \leftarrow \text{sym\_encrypt}(k^{hnd}, z_4^{hnd})$        $m_{24} \approx \{Ak, n_1, t_K, T\}_k$
39.  $m_2^{hnd} \leftarrow \text{list}(m_{21}^{hnd}, x_3^{hnd}, TGT^{hnd}, m_{24}^{hnd})$        $\{m_2 \approx \{\{\text{Cert}_K, [k, ck]_{sk_K}\}\}_{pk_C}, C, TGT, \{Ak, n_1, t_K, T\}_k\}$
40.  $\text{send\_i}(v, m_2^{hnd})$

**Fig. 7.** Algorithm 2 of Public-key Kerberos: Behavior of the KAS

and uses this public key to verify the signature  $x_2$  received in the AS\_REQ (steps 2.11–2.16). In steps 2.17–2.21 the types of the message components of the signed message  $y_1$  are checked, as well as the freshness of the nonce  $y_{12}$  by comparison to nonces stored in  $Nonce3_K$ . If the nonce is fresh then it will be stored in the set  $Nonce3_K$  in step 2.23 for freshness checks in future protocol runs. Finally, in steps 2.24–2.36  $M_K^{PK}$  generates two symmetric keys  $k$  and  $AK$ , composes the AS\_REP, and sends it to  $v$  over an insecure channel.

## 4 Formal Results

### 4.1 Security in the Symbolic Setting

In order to use the BPW model to prove the computational security of Kerberos, we first formalize the respective security properties and verify them in the BPW model. We first prove that Kerberos keeps the symmetric key, which the TGS  $T$  generated for use between user  $u$  and server  $S$ , symbolically secret from the adversary. In order to prove this, we show that Kerberos also keeps the keys generated by KAS  $K$  for the use between  $u$  and the TGS  $T$  secret. Furthermore, we prove entity authentication of the user  $u$  to a server  $S$  (and subsequently entity authentication of  $S$  to  $u$ ). This form of authentication is weaker than the authentication Kerberos offers, since we do not consider the purpose of timestamps in Kerberos. (Recall that timestamps are currently not included in the BPW model.)

**Secrecy and Authentication Requirements** We now define the notion of key secrecy, which was informally captured already in Property 1 of Sect. 2, as the following formal requirement in the language of the BPW model.

**Definition 1 (Key secrecy requirement).** For  $Kerb \in \{PK, K5\}$  the secrecy requirement  $Req_{Kerb}^{Sec}$  is:

For all  $u \in \mathcal{H} \cap \{1, \dots, n\}$ , and  $S \in \mathcal{H} \cap \{S_1, \dots, S_l\}$ , and  $t_1, t_2, t_3 \in \mathbb{N}$ :

$$\begin{aligned} & (t_1 : KA\_out_S!(ok, Kerb, u, SK^{hnd}) \\ & \vee t_2 : KA\_out_u!(ok, Kerb, S, SK^{hnd}) \\ \Rightarrow & t_3 : D[hnd_u = SK^{hnd}].hnd_u = \downarrow \end{aligned}$$

where  $t : D$  denotes the contents of database  $D$  at time  $t$ . Similarly  $t : p?m$  and  $t : p!m$  denotes that message  $m$  occurs at input (respectively output) port  $p$  at time  $t$ . As above PK refers to Public-key Kerberos and K5 to Kerberos 5. In the next section Thm. 1 will show that the symbolic Kerberos systems specified in Sect. 3.2 satisfy this notion of secrecy, and therefore Kerberos enjoys Property 1.

Next we define the notion of authentication in Property 2 in the language of the BPW model.

**Definition 2 (Authentication requirements).** For  $Kerb \in \{PK, K5\}$ :

- i. The authentication requirement  $Req_{Kerb}^{Auth1}$  is: For all  $v \in \mathcal{H} \cap \{1, \dots, n\}$ , for all  $S \in \mathcal{H} \cap \{S_1, \dots, S_l\}$ , and  $K, T$ :

$$\begin{aligned} & \exists t_3 \in \mathbb{N}. t_3 : KA\_out_S!(ok, Kerb, v, SK^{hnd}) \\ \Rightarrow & \exists t_1, t_2 \in \mathbb{N} \text{ with } t_1 < t_2 < t_3. t_2 : KA\_in_v!(continue\_prot, Kerb, T, S, \cdot) \\ & \wedge t_1 : KA\_in_v!(new\_prot, Kerb, K, T) \end{aligned}$$

- ii. The authentication requirement  $Req_{Kerb}^{Auth2}$  is: For all  $u \in \mathcal{H} \cap \{1, \dots, n\}$ , for all  $S \in \mathcal{H} \cap \{S_1, \dots, S_l\}$ , and  $K, T$ :

$$\begin{aligned} & \exists t_2 \in \mathbb{N}. t_2 : KA\_out_u!(ok, Kerb, S, SK^{hnd}) \\ \Rightarrow & \exists t_1 \in \mathbb{N} \text{ with } t_1 < t_2. t_1 : KA\_in_S!(ok, Kerb, u, SK^{hnd}) \end{aligned}$$

- iii. The overall authentication  $Req_{Kerb}^{Auth}$  for protocol  $Kerb$  is:

$$Req_{Kerb}^{Auth} := Req_{Kerb}^{Auth1} \wedge Req_{Kerb}^{Auth2}$$

Theorem 1 will show that this notion of authentication is satisfied by the symbolic Kerberos system. Therefore Kerberos has Property 2.

When proving that Kerberos has these properties, we will use the notion of a system  $Sys$  perfectly fulfilling a requirement  $Req$ ,  $Sys \models^{perf} Req$ . This means the property  $Req$  holds with probability one over the probability space of runs for a fixed security parameter (as defined in Sect. 3.1). Later we will also need the notion of a system  $Sys$  computationally fulfilling a requirement  $Req$ ,  $Sys \models^{poly} Req$ , i.e., the property holds with negligible error probability for all polynomially bounded users and adversaries (again, over the probability space of all runs for a fixed security parameter). In particular, perfect fulfillment implies computational fulfillment.

In order to prove Thm. 1, we first need to prove a number of auxiliary properties (previously called *invariants* in, e.g., [4, 11]). Although these properties are nearly identical for Kerberos 5 and Public-key Kerberos, their proofs had to be carried out separately. We consider it interesting future work to augment the BPW model with proof techniques that allow for conveniently analyzing security protocols in a more modular manner. In fact, a higher degree of modularity would simplify the proofs for each individual protocol as it could exploit the highly modular structure of Kerberos; moreover, it would also simplify the treatment of the numerous optional behaviors of this protocol.

Some of the key properties needed in the proof of Thm. 1, which formalizes Properties 1 and 2, make authentication and confidentiality statements for the first two rounds of Kerberos. These properties are described in English below; they are formalized and proved in [5].

- i) Authentication of KAS to client and Secrecy of  $AK$ :** If a user  $u$  receives a valid AS\_REP message then this message was indeed generated by  $K$  for  $u$  and an adversary cannot learn the symmetric keys contained in this message.

- ii) **TGS Authentication of the TGT:** If a TGS  $T$  receives a TGT and an authenticator  $\{v, t_v\}_{AK}$  where the key  $AK$  and the username  $v$  are contained in the TGT, then the TGT was generated by  $K$  and the authenticator was created by  $v$ .
- iii) **Authentication of TGS to client and Secrecy of  $SK$ :** If a user  $u$  receives a valid TGS\_REP then it was generated by  $T$  for  $u$  and  $S$  and no adversary can learn the session key  $SK$  contained in this message.
- iv) **Server Authentication of the ST:** If a server  $S$  receives an ST and an authenticator  $\{v, t_v\}_{SK}$  where the key  $SK$  and the name  $v$  are contained in the ST, then the ST was generated by  $T$  and the authenticator was created by  $v$ .

We can now capture the security of Kerberos in the BPW model in the following theorem, which says that Properties 1 and 2 hold symbolically for Kerberos. We show a proof excerpt in the case of Public-key Kerberos (the outline is analogous for Kerberos 5).

**Theorem 1.** (*Security of the Kerberos Protocol based on the BPW Model*)

- Let  $Sys^{K5, \text{symb}}$  be the symbolic Kerberos 5 system defined in Sect. 3.2, and let  $Req_{K5}^{\text{Sec}}$  and  $Req_{K5}^{\text{Auth}}$  be the secrecy and authentication requirements defined above. Then  $Sys^{K5, \text{symb}} \models^{\text{perf}} Req_{K5}^{\text{Sec}} \wedge Req_{K5}^{\text{Auth}}$ .
- Let  $Sys^{\text{PK}, \text{symb}}$  be the symbolic Public-key Kerberos system, and let  $Req_{PK}^{\text{Sec}}$  and  $Req_{PK}^{\text{Auth}}$  be the secrecy and authentication requirements defined above. Then  $Sys^{\text{PK}, \text{symb}} \models^{\text{perf}} Req_{PK}^{\text{Sec}} \wedge Req_{PK}^{\text{Auth}}$ .

*Proof (sketch).* We assume that all parties are honest. If user  $u$  successfully terminates a session run with a server  $S$ , i.e., there was an output (ok, PK,  $S$ ,  $k^{hnd}$ ) at  $KA_{out_u}!$ , then the key  $k$  was stored in the set  $Session\_KeysS_u$ . This implies that the key was generated by  $T$  and sent to  $u$  in a valid TGS\_REP. By auxiliary property iv), an adversary cannot learn  $k$ . The case that  $S$  successfully terminates a session run is analogous. This shows the key secrecy property  $Req_{PK}^{\text{Sec}}$ . As for the authentication property  $Req_{PK}^{\text{Auth}1}$ , if server  $S$  successfully terminates a session with  $u$ , i.e., there was an output (ok, PK,  $u$ ,  $k^{hnd}$ ) at  $KA_{out_S}!$ , then  $S$  must have received a ticket generated by  $T$  (for  $S$  and  $u$ ) and also a matching authenticator generated by user  $u$  (by auxiliary property iv)). But the ticket will only be generated if  $u$  sends the appropriate request to  $T$ , i.e., there was an input (continue\_prot, PK,  $T$ ,  $S$ ,  $AK^{hnd}$ ) at  $KA_{in_u}?$ . The request, on the other hand, contains a TGT that was generated by  $K$  for  $u$  (by auxiliary property ii)), therefore  $u$  must have sent an request to  $K$ . In particular, there had been an input (new\_prot, PK,  $K$ ,  $T$ ) at  $KA_{in_u}?$ . As for the authentication property  $Req_{PK}^{\text{Auth}2}$ , if the user  $u$  successfully terminates a session with server  $S$ , i.e., there was an output (ok, PK,  $S$ ,  $k^{hnd}$ ) at  $KA_{out_u}!$ , then it must have received a message encrypted under  $k$  that does not contain  $u$ 's name. The key  $k$  was contained in a valid TGS\_REP and was therefore generated by  $T$ , by auxiliary property iii). Only  $T$ ,  $u$ , or  $S$  could know the key  $k$ , but only  $S$  uses this key to encrypt and send a message that  $u$  received. On the other hand,  $S$  follows sending such a message immediately by an output (ok, PK,  $u$ ,  $k^{hnd}$ ) at  $KA_{out_S}!$ .  $\square$

This proof shares similarities with the Dolev-Yao style proofs of analogous properties for Kerberos 5 and PKINIT using the MSR framework [18, 20]. The two approaches are similar in the sense that both reconstruct a necessary trace backward from an end state, and in that they rely on some form of induction (based on rank/co-rank functions in MSR). In future work, we plan to draw a formal comparison between these two Dolev-Yao encodings of a protocol, and the proof techniques they support.

## 4.2 Security in the Cryptographic Setting

The results of [14] allow us to take the authentication results in Thm. 1 and derive a corresponding authentication results for a cryptographic implementation of Kerberos. Just as Property 2 holds symbolically for Kerberos, this shows that it holds in a cryptographic implementation as well. In particular, entity authentication between a user and a server in Kerberos holds with overwhelming probability (over the probability space of runs). However, symbolic results on key secrecy can only be carried over to cryptographic implementations if the protocol satisfies certain additional conditions. Kerberos unfortunately does not fulfill these definitions, and it can easily be shown that cryptographic implementations of Kerberos do not fulfill the standard notion of cryptographic key secrecy, see below. This yields the following theorem.

**Theorem 2.** (*Computational security of the Kerberos protocol*)

- Let  $Sys^{K5, \text{comp}}$  denote the computational Kerberos 5 system implemented with provable secure cryptographic primitives. Then  $Sys^{K5, \text{comp}} \models^{\text{poly}} Req_{K5}^{\text{Auth}}$ .
- Let  $Sys^{\text{PK}, \text{comp}}$  denote the computational Public-key Kerberos system implemented with provable secure cryptographic primitives. Then  $Sys^{\text{PK}, \text{comp}} \models^{\text{poly}} Req_{PK}^{\text{Auth}}$ .

*Proof (Sketch for public-key Kerberos).* By Thm. 1, we know that  $Sys^{\text{PK}, \text{id}} \models^{\text{perf}} Req_{PK}^{\text{Auth}}$ . And, as we mentioned earlier, the cryptographic implementation of the BPW model (using provably secure cryptographic primitives) is at least as secure as the BPW model,  $Sys^{\text{cry}, \text{comp}} \geq_{\text{sec}}^{\text{poly}} Sys^{\text{cry}, \text{id}}$ . After checking that the “Commitment Problem” does not occur in the protocol, we can use the Preservation of Integrity Properties Theorem from [6] to automatically obtain Thm. 2.

The Commitment Problem occurs when keys that have been used for cryptographic work are revealed later in the protocol. If the simulator in [14] (with which one can simulate a computational adversary attack on the symbolic system) learns in some abstract way that e.g. a ciphertext was sent, the simulator generates a distinguishable ciphertext without knowing the symmetric key nor the plaintext. If the symmetric key is revealed later in the protocol then the trouble for the simulator will be to generate a suitable symmetric key that decrypts the ciphertext into the correct plaintext. This is typically an impossible task. In order for the simulation with the BPW model to work, one thus needs to check that the Commitment Problem does not occur in the protocol.  $\square$

As far as key secrecy is concerned, it can be proven that the adversary attacking the cryptographic implementation does not learn the secret key string as a whole. However, it does not necessarily rule out that an adversary will be able to distinguish the key from other fresh random keys, as required by the definition of *cryptographic key secrecy*. This definition of secrecy says that an adversary cannot learn any partial information about such a key and is hence considerably stronger than requiring that an adversary cannot obtain the whole key. For Kerberos we can show that the key  $SK$  does not satisfy cryptographic key secrecy after the last round of Kerberos, i.e.,  $SK$  is distinguishable from other fresh random keys. It should also be noted that this key  $SK$  is still indistinguishable from random after the second round but before the start of the third round of Kerberos. We have the following proposition.

**Proposition 1.** *a) Kerberos does not offer cryptographic key secrecy for the key  $SK$  generated by the TGS  $T$  for the use between client  $C$  and server  $S$  after the start of the last round of Kerberos.*

*b) After the TGS exchange and before the start of the CS exchange the key  $SK$  generated by the TGS  $T$  is still cryptographically secret.*

*Proof.* a) To see that Kerberos does not offer cryptographic key secrecy for  $SK$  after the start of the third round, note that the key  $SK$  is used in the protocol for symmetric encryption. As symmetric encryption always provides partial information to an adversary if the adversary also knows the message that was encrypted. An adversary can exploit this to distinguish the key  $SK$  as follows: the adversary first completes a regular Kerberos execution between  $C$  and  $S$  learning the message  $\{C, t'\}_{SK}$  encrypted under the unknown key  $SK$ . The adversary will also learn a bounded time period  $TP$  (of a few seconds) in which the timestamp  $t'$  was generated. Next a bit  $b$  is flipped and the adversary receives a key  $k$ , where  $k = SK$  for  $b = 0$  and  $k$  is a fresh random key for  $b = 1$ . The adversary now attempts to decrypt  $\{C, t'\}_{SK}$  with  $k$  yielding a message  $m$ . If  $m \neq C, t$  for a timestamp  $t$  then the adversary guesses  $b = 1$ . If  $m = C, t$  for a timestamp  $t$  then the adversary checks whether  $t \in TP$  or not. If  $t \notin TP$  then the adversary guesses  $b = 1$  otherwise the adversary guesses  $b = 0$ . The probability of the adversary guessing correctly is then  $1 - \epsilon$ , where  $\epsilon$  is the probability that for random keys  $k$ ,  $SK$  the ciphertext  $\{C, t'\}_{SK}$  decrypted with  $k$  is  $C, t$  with  $t \in TP$ . Clearly,  $\epsilon$  is negligible (since the length of the time period  $TP$  does not depend on the security parameter). Hence,  $SK$  is distinguishable and cryptographic key secrecy does not hold.

b) However, before the third round has been started the key  $SK$  is not only unknown to the adversary but, in particular,  $SK$  has not been used for symmetric encryption yet. We can therefore invoke the key secrecy preservation theorem of [9], which states that a key that is symbolically secret and symbolically unused is also cryptographically secret. This allows us to conclude that  $SK$  is cryptographically secret from the adversary.

For similar reasons, we also have the following proposition

**Proposition 2.** *a) Kerberos does not offer cryptographic key secrecy for the key  $AK$  generated by the KAS  $K$  for the use between client  $C$  and TGS  $T$  after the start of the second round of Kerberos.*

*b) After the AS exchange and before the start of the TGS exchange the key  $AK$  generated by the KAS  $K$  is still cryptographically secret.*

Finally, we note the following

*Remark 1.* Kerberos allows the client or the server to generate a sub-session key [39]. This optional key, which can then be used for the encryption of further communication between the two parties, is cryptographically secret as it can be proven symbolically secret and symbolically unused. This proof can easily be conducted symbolically similar to Thm. 1, and then the key secrecy preservation theorem of [10] can be used to automatically obtain a proof of cryptographic key secrecy for the optional sub-key. Note that this preservation theorem could not be used for proving cryptographic key secrecy for the main key as this key is already used within the key exchange protocol.

## 5 Conclusions and Future Work

In this paper, we have exploited the Dolev-Yao style model of Backes, Pfitzmann, and Waidner [8, 12, 13] to obtain the first computational proof of authentication for the core exchanges of the Kerberos protocol and its extension to public keys (PKINIT). Although the proofs sketched here are conducted symbolically, grounding the analysis on the BPW model automatically lifts the results to the computational level, assuming that all cryptography is implemented using provably secure primitives. Cryptographic key secrecy in the sense of indistinguishability of the exchanged key from a random key could only be established for the optional sub-key exchanged in Kerberos while for the actually exchanged key, cryptographic key secrecy could be proven not to hold.

Potentially promising future work includes the augmentation of the BPW model with specialized proof techniques that allow for conveniently performing modular proofs. Such techniques would provide a simple and elegant way to integrate the numerous optional behaviors supported by Kerberos and nearly all commercial protocols; for example, this would facilitate the analysis of DH mode in PKINIT which is part of our ongoing work. We intend to tackle the invention of such proof techniques that are specifically tailored towards the BPW model in the near future, e.g., by exploiting recent ideas from [24]. Another potential improvement we plan to pursue in the near future is to augment the BPW model with timestamps; this would in particular allow us to establish authentication properties that go beyond entity authentication [18, 20, 21]. A further item on our research agenda is to fully understand the relation between the symbolic correctness proof for Kerberos 5 presented here and the corresponding results achieved in the MSR framework [18, 20].

## References

1. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proc. of Computer-aided Verification (CAV)*. Springer, 2005. URL: [www.avispa-project.org](http://www.avispa-project.org).
2. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. TACS*, pages 82–94, 2001.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, pages 3–22. Springer LNCS 1872, 2000.
4. M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. ESORICS*, pages 89–108. Springer LNCS 3193, 2004.
5. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically sound security proofs for basic and public-key Kerberos. IACR Cryptology ePrint Archive, Report 2006/219, <http://eprint.iacr.org/>, June 2006.
6. M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th STACS*, pages 675–686. Springer LNCS 2607, 2003.
7. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.
8. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. CSFW'04*, pages 204–218, June 2004.
9. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Trans. Dependable Secure Comp.*, 2(2):109–123, April–June 2005.
10. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. 26th IEEE Symposium on Security & Privacy*, pages 171–182, 2005. Extended version in IACR Cryptology ePrint Archive 2004/300.
11. M. Backes and B. Pfitzmann. On the cryptographic key secrecy of the strengthened Yahalom protocol. In *Proceedings of 21st IFIP SEC'06*, To appear.
12. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. CCS'03*, pages 220–230, 2003.
13. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. ESORICS'03*, pages 271–290. Springer LNCS 2808, 2003.
14. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive, Report 2003/015, <http://eprint.iacr.org/>, January 2003.
15. G. Bella and L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In *Proc. ESORICS'98*, pages 361–375. Springer LNCS 1485, 1998.
16. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proc. CRYPTO '93*, pages 232–249. Springer LNCS 773, 1994.
17. B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, 2006.
18. F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. An Analysis of Some Properties of Kerberos 5 Using MSR. In *Proc. CSFW'02*, 2002.
19. R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). In *Proc. 3rd Theory of Cryptography Conference (TCC)*, 2006.

20. I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key Kerberos. In *Proc. WITS'06*, 2006.
21. I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Specifying Kerberos 5 Cross-Realm Authentication. In *Proc. WITS'05*, pages 12–26, 2005.
22. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. ESOP-14*, pages 157–171, 2005.
23. A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. ICALP*, pages 16–29. Springer LNCS 3580, 2005.
24. A. Datta, A. Derek, J. Mitchell, and B. Warinschi. Key exchange protocols: Security definition, proof method, and applications. In *19th IEEE Computer Security Foundations Workshop (CSFW 19)*, Venice, Italy, 2006. IEEE Press.
25. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Trans. Info. Theory*, 2(29):198–208, 1983.
26. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. STOC*, pages 218–229, 1987.
27. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
28. J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. CCS-8*, pages 186–195, 2001.
29. C. He and J. C. Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *Proc. NDS'05*, 2005.
30. J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *Proc. CRYPTO*, pages 548–564. Springer LNCS 2729, 2003.
31. IETF. Public Key Cryptography for Initial Authentication in Kerberos, 1996–2006. Sequence of Internet drafts available from <http://tools.ietf.org/wg/krb-wg/draft-ietf-cat-kerberos-pk-init/>.
32. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. FOCS*, pages 372–381, 2003.
33. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. Symp. Security and Privacy*, pages 71–85, 2004.
34. C. Meadows. Analysis of the internet key exchange protocol using the NRL Protocol Analyzer. In *Proc. IEEE Symp. Security and Privacy*, pages 216–231, 1999.
35. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. TCC*, pages 133–151. Springer LNCS 2951, 2004.
36. Microsoft. Security Bulletin MS05-042. <http://www.microsoft.com/technet/security/bulletin/MS05-042.msp>, Aug. 2005.
37. J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
38. C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, Sept. 1994.
39. C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5), July 2005. <http://www.ietf.org/rfc/rfc4120.txt>.
40. B. Pfizmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. S&P*, pages 184–200, 2001.