Cryptographically Sound Security Proofs for Basic and Public-Key Kerberos *

M. Backes[1], I. Cervesato[2], A.D. Jaggard[3], A. Scedrov[4], and J.-K. Tsay[4]

[1] Saarland University
backes@cs.uni-sb.de
[2] Deductive Solutions
iliano@deductivesolutions.com
[3] Tulane University
adj@math.tulane.edu
[4] University of Pennsylvania
{scedrov|jetsay}@math.upenn.edu

Abstract. We present a computational analysis of core Kerberos with public-key authentication (PKINIT) in which we consider authentication and key secrecy properties. These 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. Our work constitutes the first formal verification of a significant subset of an industrial protocol at the computational level. By 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 [DY83]; 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 definition on complexity theory [GM84,GMW87,BR94].

* 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 [ABB+05,BCJS02,BCJ+ar,BR97,BP98,Mea99].

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 within the Dolev-Yao approach so far. Kerberos 4, which was then the prevalent version, was verified using the Isabelle theorem prover [BR97,BP98]. Butler et al. have extensively analyzed the currently predominant version, Kerberos 5 [NYHR05], within the Dolev-Yao approach [BCJS02,BCJ+ar]: they showed that a detailed specification of the core protocol enjoys the expected authentication and secrecy properties except for some relatively innocuous anomalies [CJSW05]; they examined the support for cross-domain authentication, finding it correct against its specification but very weak in practice [CJS+06]; and they exposed a serious attack against the public key extension of Kerberos (PKINIT) which led to an immediate correction of the specification and was accompanied by security patches from several vendors, including Microsoft.

However, 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 cryptograph is implemented with actual cryptographic primitives that satisfy their 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 [BPW03a,BPW03b,BP04b] (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 constitutes by far the largest protocol whose cryptographic security has so far been inferred from a proof in this Dolev-Yao style approach; earlier proofs

in this approach were only for small examples of purely scientific relevance, e.g., for the Needham-Schroeder-Lowe, the Otway-Rees, and the Yahalom protocols [Bac04,BP04a,BPar]. 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 computational approach. We view this as a research opportunity for the short-term future.

1.1 Related Work

Early work on linking Dolev-Yao models and cryptography [AJ01,AR00,GTZ01,Lau01] only considered passive attacks, and therefore cannot make general statements about protocols. A cryptographic justification for a Dolev-Yao model in the sense of simulatibility [PW01], i.e., under active attacks and within arbitrary surrounding interactive protocols, was first given in [BPW03a] with extensions in [BPW03b,BP04b]. Based on that Dolev-Yao model, the well-known Needham-Schroeder-Lowe, Otway-Rees, and Yahalom protocols were proved secure in [BP04a,Bac04,BPar]. 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 [HM05], although Kerberos is still a much more complex protocol.

Laud [Lau04] 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. [HLM03] and Micciancio and Warinschi [MW04] have also given a cryptographic underpinning under active attacks. Their results are narrower than that in [BPW03a] since they are specific for public-key encryption and certain protocol classes, but consider slightly simpler real implementations. Cortier and Warinschi [CW05] 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 [BP05] and Canetti and Herzog [CH06] 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 [BPW03a].

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 [Bla06,DDM+05,IK03,MMS98,MMST01] . This is orthogonal to the work of justifying Dolev-Yao models, which offer a higher level of

abstractions and thus much simpler proofs where applicable, so that proofs of larger systems can be automated.

1.2 Structure of the Paper

We start in Section 2 with a review of Kerberos and its public-key extension. In Section 3, we recall the Dolev-Yao style model of Backes, Pfitzmann, and Waidner (e.g. [BJ03,BPW03c,BPW03b,BP04b]), 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, Section 5 summarizes this effort and outlines areas of future work.

2 Kerberos 5 and its Public-Key Extension

The Kerberos protocol [NT94,NYHR05] 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 [NYHR05], 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, may be used in two modes: "public-key encryption mode" and "Diffie-Hellman (DH) mode." In recent work [CJS+06], 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. The fix first appeared in revision 27 of the PKINIT specification [IET06]; subsequent drafts have not changed this aspect of PKINIT, and the current draft (revision 34 of PKINIT) has moved to the next stage in the IETF [The] standards process. In the rest of this section, we describe the operation of both basic Kerberos and Kerberos with PKINIT in public-key mode.

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

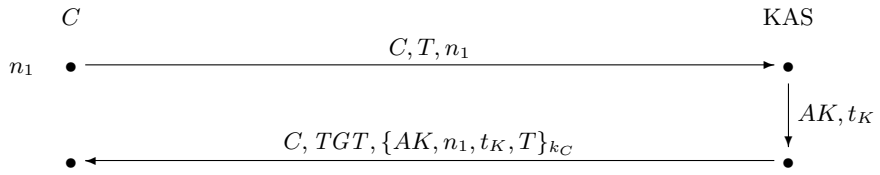

Fig. 1. Message Flow in the Traditional AS Exchange, where $TGT = \{AK, C, t_K\}_{k_T}$.

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 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 Figure 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 [NYHR05]. 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. These encrypted messages are accompanied by the client's

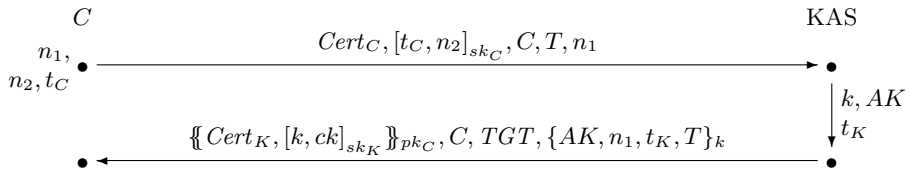

Fig. 2. Message flow in the fixed version of PKINIT, where $TGT = \{AK, C, t_K\}_{k_T}$.

name—and other data that we abstract away—sent in the clear. 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 [NYHR05] for the complete specification of Kerberos 5, [IET06] for the specification of PKINIT, and [BCJS02] for a formalization of Kerberos at an intermediate level of detail.

The AS Exchange in PKINIT PKINIT [IET06] 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. This protocol extension adds complexity to Kerberos as it retains symmetric encryption in the later rounds but relies on asymmetric encryption, digital signatures, and corresponding certificates in the first round.

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.[5]

[5] The login process changes as very few users would be able to remember a random public/secret key pair. In Microsoft Windows, keys and certificate chains are stored in a smartcard that the user swipes in a reader at login time. A passphrase is generally required as an additional security measure [DCB01]. Other possibilities include keeping these credentials on the user's hard drive, again protected by a passphrase.

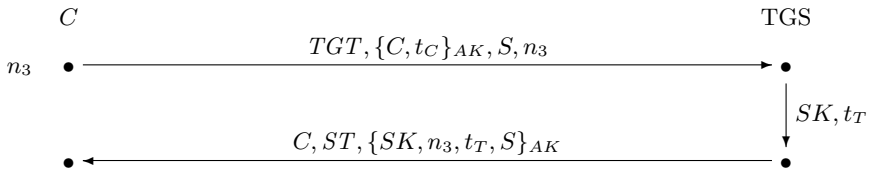

Fig. 3. Message flow in the TGS exchange, where $TGT = \{AK, C, t_K\}_{k_T}$ and $ST = \{SK, C, t_T\}_{k_S}$.

As noted above, PKINIT can operate in two modes. These resemble 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. The modes of PKINIT provide two different ways for the KAS to transmit this key using the asymmetric key pairs rather than a key that is shared between the client and KAS. In DH mode, the key pairs (pk_C, sk_C) and (pk_K, sk_K) are used to provide digital signature support for an authenticated Diffie-Hellman key agreement which is then used to protect the fresh key AK. A variant of this mode allows the reuse of previously generated shared secrets. In public-key encryption mode, 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.

We will not discuss the DH mode any further as our preliminary investigation did not reveal any flaw in it; we are still working on a complete analysis of this mode. Furthermore, it appears not to have yet been included in any of the major operating systems. The only support we are aware of is within the PacketCable system [Cab04], developed by CableLabs, a cable television research consortium.

Figure 2 illustrates the AS exchange when the fixed version (which defends against the attack of [CJS+06]) 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 Figure 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. (The nonces and timestamp at the left of this line indicate that these are generated by C specifically for this request.)

The second line in Figure 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

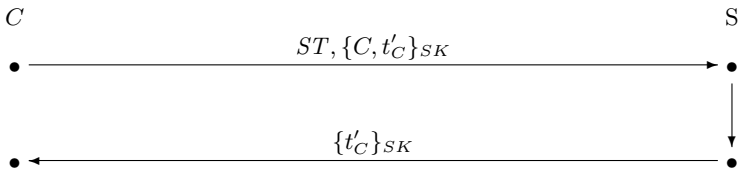

Fig. 4. Message flow in the CS exchange, where $ST = \{SK, C, t_T\}_{k_S}$.

adding the message $\{\!|Cert_K, [k, ck]_{sk_K}|\!\}_{pk_C}$. 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 [NYHR05]) 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 [CJS+06].

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. This exchange is shown in Fig. 3. The first line of this figure 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 $ST = \{SK, C, t_T\}_{k_S}$) 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. As shown in the first line of the figure, C starts by sending a message (AP_REQ) that is similar to the TGS_REQ message of the previous round: in contains the (service) ticket and an authenticator ($\{C, t'_C\}_{SK}$) that is encrypted under the key contained in the service ticket. As shown in the second line of the figure, the server S simply responds with an AP_REP message $\{t'_C\}_{SK}$ containing the timestamp from the authenticator encrypted under the key from the service ticket.

Attack on PKINIT The attack that we found against the then-current specification of PKINIT was reported in [CJS+06]. 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 who was herself a legitimate client to copy a message from another client 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 [BCJS02,BCJ$^+$ar] and for the AS exchange when PKINIT is used [CJS$^+$06]. The first property we prove here concerns the secrecy of keys, a notion that is captured formally as Def. 1 in Sec. 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 Sec. 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; Thm. 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 [CJS$^+$06].

3 The BPW Model

We will now abstractly review the BPW model and then formalize Kerberos using it.

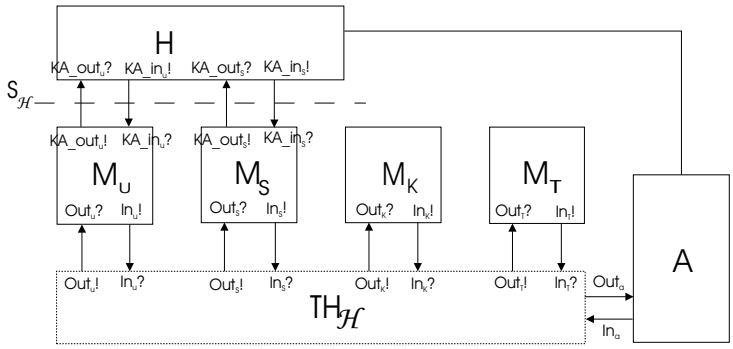

Fig. 5. Overview of the Kerberos symbolic system

3.1 Review of the BPW Model

The BPW model introduced in [BPW03c] 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; and 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 [PW01] and is represented as a deterministic machine $\mathrm{TH}_{\mathcal{H}}$ (also an I/O automaton), called *trusted host*, where $\mathcal{H} \subset \{1, ..., n\}$ denotes the set of 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 $view_{conf}(\mathsf{H})$ 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 exist 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 [BPW03a,BPW03b,BP04b,BPW03c] it was established that the cryptographic implementation of the BPW model is *at least as secure as* the BPW model, (denoted by $\geq$, Fig. 6) 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 $(\hat{M}_1, \mathsf{S}) \in Sys_1$, there exist a polynomial-time adversary A_2 on a corresponding symbolic structure $(\hat{M}_2, \mathsf{S}) \in Sys_2$ such that the view of H is computationally indistinguishable in both configurations(Fig. 6). 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 [BPW03c] 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

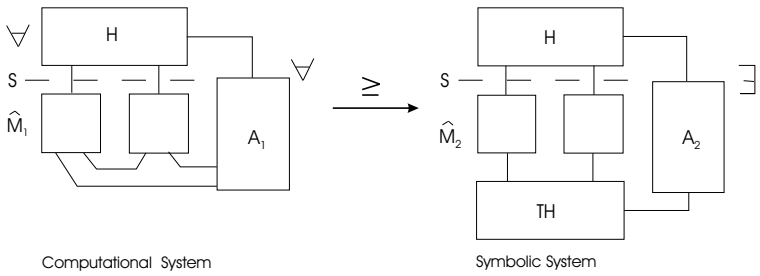

Fig. 6. Simulatability: The views of H must be indistinguishable

for the two protocols. We denote Kerberos with PKINIT by "PK," and basic Kerberos by "K5." If we let Kerb∈{PK, K5} then, as described in Section 3.1, for each user $u \in \{1, ..., n\}$ there is a *protocol machine* $\mathrm{M}_u^{\mathrm{Kerb}}$ which executes the protocol for u. There are also protocol machines for the KAS K and the TGT T, denoted by $\mathrm{M}_K^{\mathrm{Kerb}}$ and $\mathrm{M}_T^{\mathrm{Kerb}}$. Furthermore, if $S_1, ..., S_l$ are the servers in the realm of T then there are server machines $\mathrm{M}_S^{\mathrm{Kerb}}$ for $S \in \{S_1, ..., S_l\}$. 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 $\mathrm{KA\_out}_u!$ and $\mathrm{KA\_in}_u?$, respectively ("KA" for"Key sharing and Authentication"). The ports for the server machines are labeled similarly (see Fig. 5).

The behaviors of the protocol machines is described in Algorithm 1 to 5 (Fig. 7- 16). In the following, we comment on two algorithms of PKINIT (Fig. 7 and Fig. 8) , the rest is displayed in Appendix A. If, for instance, a protocol machine $\mathrm{M}_u^{\mathrm{PK}}$ receives a message (new_prot, PK, K, T) at $\mathrm{KA\_in}_u?$ then it will execute Algorithm 1A (Fig. 7) to start a protocol run. We give a description below. The state of the protocol machine $\mathrm{M}_u^{\mathrm{Kerb}}$ consists of the bitstring u and the sets $Nonce_u$, $Nonce2_u$, $TGTicket$, and $Session\_KeysS_u$, in which $\mathrm{M}_u^{\mathrm{Kerb}}$ 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 $u \in \{1, ..., n\}$ and honest servers $S \in \{S_1, ..., S_l\}$ 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 $\mathcal{H} \subset \{1, ..., n, K, T, S_1, ..., S_l\}$ the honest participants, i.e. for $v \in \mathcal{H}$ the machine $\mathrm{M}_v^{\mathrm{Kerb}}$ 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 $\{K, T\} \subset \mathcal{H} \subset \{1, ..., n, K, T, S_1, ..., S_l\}$ the user interface of public-key Kerberos will be the set $S_{\mathcal{H}} := \{\mathrm{KA\_out}_u!, \mathrm{KA\_in}_u? \mid u \in \mathcal{H} \setminus \{K, T\}\}$. The symbolic system is the set $Sys^{\mathrm{Kerb,\ symb}} := \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})\}$. 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. 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},..., hnd_{u_m}, hnd_a, *len)*, where $\mathcal{H} = \{u_1, ..., u_m\}$. 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 each entry $x \in D$: $x.hnd \in \mathcal{INDS}$, called index, 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 $x.type \in typeset = \{$auth, cert, enc, nonce, list, pke, pkse, sig, ske, 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 [BP04a] for a more detailed explanation) . The entry $x.arg = (a_1, ..., a_j)$ is a possibly empty list of arguments. Many values a_i are in $\mathcal{INDS}$. $x.hnd_u \in \mathcal{HNDS} \cup \{\downarrow\}$ 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 the functions from L.

Initially, D is empty. $\mathrm{TH}_{\mathcal{H}}$ 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). And 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 user 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. 7 and Fig. 8) which are more complex than the algorithms in Kerberos 5. However, with these explanations the remaining algorithms (App. A) should be easily understandable. For details on the definition of the used commands see [BPW03c,BPW03b,BP04b]. 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 (new_prot, PK, K, T) at port $\mathrm{KA\_in}_u$?. Upon such an input, $\mathrm{M}_u^{\mathrm{PK}}$ runs Algorithm 1A (Fig. 7) which prepares and sends the AS_REQ to K using the BPW model. $\mathrm{M}_u^{\mathrm{PK}}$ generates symbolic nonces in steps 1A.1 and 1A.2 by sending 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. The command store(_) in step 1A.5-6 makes entries

A) Input:$(\text{new\_prot}, \text{PK}, K, T)$ at $KA\_\text{in}_u?$.

1. $n_{u,1} t_u^{hnd} \leftarrow \text{gen\_nonce}()$
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. $s^{hnd} \leftarrow \text{sign}(ske_u^{hnd}, l^{hnd})$ $\{s \approx [t_C, n_2]_{sk_C}\}$
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. $\text{send\_i}(K, m_1^{hnd})$

B) Input:$(\text{continue\_prot}, \text{PK}, T, S, AK^{hnd})$ at $\text{KS\_in}_u?$ for $S \in \{S_1, ..., S_l\}$

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. $\text{send\_i}(T, m_2^{hnd})$

Fig. 7. Algorithm 1 of Public-key Kerberos: Evaluation of inputs from the user (starting the AS and TG exchanges).

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^{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 $\text{out}_K?$ with $v \in \{1, .., n\}$, the machine M_K^{PK} runs Algorithm 2 (Fig. 8) 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 $\text{retrieve}(x_i^{hnd})$ 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 $\text{pk\_of\_cert}(\_)$ 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,

Input:(v, K, i, m^{hnd}) at out_K? with $v \in \{1, ..., n\}$.

1. $x_i^{hnd} \leftarrow \text{list\_proj}(m^{hnd}, i)$ for $i = 1, ..., 5$
2. $type_i \leftarrow \text{get\_type}(x_i^{hnd})$ for $i = 1, 2, 5$ $\{x_1 \approx Cert_C, x_2 \approx [t_C, n_2]_{sk_C}, x_5 \approx n_1\}$
3. $x_i \leftarrow \text{retrieve}(x_i^{hnd})$ for $i = 3, 4$ $\{x_3 \approx C, x_4 \approx T\}$
4. **if** $(type_1 \neq \text{cert}) \vee (type_2 \neq \text{sig}) \vee (type_5 \neq \text{Nonce}) \vee (x_3 \neq v) \vee (x_4 \neq T)$ **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** $b = false$ **then**
10. Abort
11. **end if**
12. $pke_v^{hnd} \leftarrow \text{pk\_of\_cert}(pke_{CA}^{hnd}, x_1^{hnd})$
13. $type_6 \leftarrow \text{get\_type}(pke_v^{hnd})$
14. **if** $(type_6 \neq \text{pke})$ **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** $b = false$ **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. $type_{12} \leftarrow \text{get\_type}(y_{12}^{hnd})$
24. **if** $(type_{12} \neq \text{nonce}) \vee ((y_{12}^{hnd}, .) \in Nonce3_K)$ **then**
25. Abort
26. **end if**
27. $Nonce3_K := Nonce3_K \cup \{(y_{12}^{hnd}, v)\}$
28. $k^{hnd} \leftarrow \text{gen\_symenc\_key}()$
29. $AK^{hnd} \leftarrow \text{gen\_symenc\_key}()$
30. $auth^{hnd} \leftarrow \text{auth}(k^{hnd}, m^{hnd})$ $\{auth \approx ck\}$
31. $z_1^{hnd} \leftarrow \text{list}(k^{hnd}, auth^{hnd})$ $\{z_1 \approx k, ck\}$
32. $s_2^{hnd} \leftarrow \text{sign}(ske_K^{hnd}, z_1^{hnd})$ $\{s_2 \approx [k, ck]_{sk_K}\}$
33. $z_2^{hnd} \leftarrow \text{list}(cert_K^{hnd}, s_2^{hnd})$ $\{z_2 \approx Cert_K, [k, ck]_{sk_K}\}$
34. $m_{21} \leftarrow \text{encrypt}(pke_K^{hnd}, z_2^{hnd})$ $\{m_{21} \approx \{\{Cert_K, [k, ck]_{sk_K}\}\}_{pk_C}\}$
35. $z_3^{hnd} \leftarrow \text{list}(AK^{hnd}, x_3^{hnd}, t_K^{hnd})$ $\{z_3 \approx AK, C, t_K, T\}$
36. $TGT^{hnd} \leftarrow \text{sym\_encrypt}(skse_{K,x_4}^{hnd}, z_3^{hnd})$ $\{TGT \approx \{AK, C, t_K\}_{k_T}\}$
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. $m_2^{hnd} \leftarrow \text{list}(m_{21}^{hnd}, x_3^{hnd}, TGT^{hnd}, m_{24}^{hnd})$
$\{m_2 \approx \{\{Cert_K, [k, ck]_{sk_K}\}\}_{pk_C}, C, TGT, \{Ak, n_1, t_K, T\}_k\}$
40. $\text{send\_i}(v, m_2^{hnd})$

Fig. 8. Algorithm 2 of Public-key Kerberos: Behavior of the KAS

in steps 2.24 - 2.36 $\mathrm{M}_K^{\mathrm{PK}}$ generates two symmetric keys k and AK, composes the AS_REP, and sends it to v over an insecure channel.

4 Formal Results

discussed earlier possesses the properties informally outlined in Section 2. We begin by formalizing the respective security properties and verify them properties in the BPW model in Section 4.1. Then, in Section 4.2, we rely on previous work to transfer the authentication aspect of these results to the computational setting, and discuss the notion of computational secrecy.

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. Timestamps are currently not modeled 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 Section 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* $Req^{\mathsf{Sec}}_{Kerb}$ *is:*

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

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

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 Theorem 1 will show that the symbolic Kerberos systems specified in Section 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 $Req^{\mathsf{Auth1}}_{Kerb}$ *is: For all* $v \in \mathcal{H} \cap \{1, ..., n\}$*, for all* $S \in \mathcal{H} \cap \{S_1, ..., S_l\}$*, and* K, T*:*

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

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

$$\begin{aligned} &\exists\, t_2 \in \mathbb{N}.\ \ t_2 : \mathrm{KA\_out}_u!\,(\mathrm{ok}, \mathrm{Kerb}, S, SK^{hnd}) \\ \Rightarrow\ &\exists\, t_1 \in \mathbb{N} \textit{ with } t_1 < t_2.\ t_1 : \mathrm{KA\_in}_S!\,(\mathrm{ok}, \mathrm{Kerb}, u, SK^{hnd}) \end{aligned}$$

iii. The overall authentication $Req^{\mathsf{Auth}}_{Kerb}$ *for protocol Kerb is:*

$$Req^{\mathsf{Auth}}_{Kerb} := Req^{\mathsf{Auth1}}_{Kerb} \wedge Req^{\mathsf{Auth2}}_{Kerb}$$

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*, $Sys \models^{\mathrm{perf}} Req$. This means the property *Req* holds with probability one over the probability space of runs for a fixed security parameter (as defined in Section 3.1). Later we will also need the notion of a system *Sys computationally fulfilling a requirement Req*, $Sys \models^{\mathrm{poly}} Req$, 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 Theorem 1, we need first to prove a number of auxiliary properties (previously called *invariant* in, e.g., [Bac04,BPar]). 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 Theorem 1, which formalizes Properties 1 and 2, make authentication and confidentiality statements for the first two rounds of Kerberos. They are listed intuitively next. These properties are formalized and proved in Appendix B.

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

ii) TGS Authentication of the TGT: If 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 user u receives a valid TGS_REP then it was generated by T for u and S. And an adversary cannot learn the contained session key SK.

iv) Server Authentication of the ST: If server S receives a 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* $Sys^{\mathrm{K5,\ symb}}$ *be the symbolic Kerberos 5 system defined in Section 3.2, and let* Req^{Sec}_{K5} *and* Req^{Auth}_{K5} *be the secrecy and authentication requirements defined above. Then* $Sys^{\mathrm{K5,\ symb}} \models^{\mathrm{perf}} Req^{\mathsf{Sec}}_{K5} \wedge Req^{\mathsf{Auth}}_{K5}$.
- *Let* $Sys^{\mathrm{PK,\ symb}}$ *be the symbolic Public-key Kerberos system, and let* Req^{Sec}_{PK} *and* Req^{Auth}_{PK} *be the secrecy and authentication requirements defined above. Then* $Sys^{\mathrm{PK,\ symb}} \models^{\mathrm{perf}} Req^{\mathsf{Sec}}_{PK} \wedge Req^{\mathsf{Auth}}_{PK}$.

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 i), an adversary cannot learn k. Similar holds for the case that S successfully terminates a session run. This shows the key secrecy property Req^{Sec}_{PK}. As for the authentication property $Req^{\mathsf{Auth1}}_{PK}$, 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 $Req^{\mathsf{Auth2}}_{PK}$, 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$_S$!. □

This proof shares similarities with the Dolev-Yao style proofs of analogous results attained for Kerberos 5 and PKINIT using the MSR framework [BCJS02,BCJ$^+$ar,CJS$^+$06].

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 [BPW03c] 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 hold 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* $Sys^{\mathrm{K5,\ comp}}$ *denote the computational Kerberos 5 system implemented with provable secure cryptographic primitives. Then* $Sys^{\mathrm{K5,\ comp}} \models^{\mathrm{poly}} Req^{\mathsf{Auth}}_{K5}$.
- *Let* $Sys^{\mathrm{PK,\ comp}}$ *denote the computational Public-key Kerberos system implemented with provable secure cryptographic primitives. Then* $Sys^{\mathrm{PK,\ comp}} \models^{\mathrm{poly}} Req^{\mathsf{Auth}}_{PK}$.

Proof (Sketch for public-key Kerberos). By Theorem 1, we know that $Sys^{PK,\ id} \models^{perf} Req^{\mathsf{Auth}}_{PK}$. 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, $Sys^{\mathrm{cry,\ comp}} \geq^{\mathrm{poly}}_{\mathrm{sec}} Sys^{\mathrm{cry,\ id}}$. After checking that the "Commitment Problem" does not occur in the protocol, we can use the Preservation of Integrity Properties Theorem from [BJ03] 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 [BPW03c] (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. □

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 considerable 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 is the key SK generated by the TGS T 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 knowsn the message that was encrypted. We explain in the following how an adversary can exploit this to distinguish the key SK: An 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 ϵ 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, ϵ 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 [BP05], 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, one also has the next 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 is the key AK generated by the KAS K still cryptographic secret.

Optional Sub-Session Key Kerberos may allow the client or the server to generate a sub-session key. This optional key can then be used for the encryption of further communication between the two parties. To send the optional sub-session key to the other party, the generator of this optional key (C or S) includes the key as part of the message which is encrypted using the session key SK. For instance, server S may generate the optional key k and send $\{t', k\}_{SK}$ as the AP_REP. It is easy to see that, due to the key secrecy of SK, an adversary cannot learn the optional key (i.e., in the language of the BPW model, an adversary does not get a handle to this key). Since the optional key is not used in the protocol, we may invoke Thm. IV.1 of [BP05]. This theorem says that unused keys, which the adversary cannot learn, are kept cryptographically secret by the protocol. This approach is illustrated for the Yahalom protocol in [BPar].

Corollary 1. *(Computational security of the optional sub-session) For fixed* $Kerb \in \{PK, K5\}$*, the symbolic Kerberos system* $Sys^{\mathrm{Kerb,\ id}}$ *from section 3.2 keeps the optional sub-session key symbolically secret, and all polynomial-time configurations of the computational public-key Kerberos system* $Sys^{\mathrm{Kerb,\ comp}}$ *keep the optional sub-session key cryptographically secret.*

5 Conclusions and Future Work

In this paper, we have exploited the Dolev-Yao style model of Backes, Pfitzmann, and Waidner [BPW03a,BPW03b,BP04b] 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.

Concerning future work, it seems promising to augment the BPW model with specialized proof techniques that allow for conveniently performing proofs in a modular manner. 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 [DDMW06]. 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 [BCJS02,BCJ+ar,CJSW05,CJS+06]. 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 [BCJS02,BCJ$^+$ar,CJS$^+$06].

References

[ABB$^+$05] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Pierre-Cyrille Heám, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Micha"el Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. 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`.

[AJ01] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[Bac04] Michael Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.

[BCJ$^+$ar] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Christopher Walstad. Formal analysis of Kerberos 5. *Theor. Comp. Sci., Special issue on Automated Reasoning for Security Protocol Analysis*, to appear.

[BCJS02] Fred Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. An Analysis of Some Properties of Kerberos 5 Using MSR. In *Proc. CSFW'02*, 2002.

[BJ03] Michael Backes and Christian Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proceedings of 20th STACS'03*, volume 2607 of LNCS, pages 675–686. Springer-Verlag, Berlin Heidelberg, February 2003.

[Bla06] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, 2006.

[BP98] 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.

[BP04a] Michael Backes and Birgit 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.

[BP04b] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Proc. CSFW'04*, pages 204–218, June 2004.

[BP05] Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Trans. Dependable Secure Comp.*, 2(2):109–123, April–June 2005.

[BPar] Michael Backes and Birgit Pfitzmann. On the cryptographic key secrecy of the stregthened Yahalom protocol. In *Proceedings of 21st IFIP SEC'06*, to appear.

[BPW03a] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

[BPW03b] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. ESORICS'03*, volume 2808, pages 271–290. Springer-Verlag, Berlin Heidelberg, 2003.

[BPW03c] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive, Report 2003/015, `http://eprint.iacr.org/`, January 2003.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1994.

[BR97] G. Bella and E. Riccobene. Formal Analysis of the Kerberos Authentication System. *J. Universal Comp. Sci.*, 3(12):1337–1381, December 1997.

[Cab04] Cable Television Laboratories, Inc. PacketCable Security Specification, 2004. Technical document PKT-SP-SEC-I11-040730.

[CH06] Ran Canetti and Jonathan 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.

[CJS+06] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Chris Walstad. Breaking and fixing public-key Kerberos. In *Proc. WITS'06*, pages 55–70, 2006.

[CJSW05] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Christopher Walstad. Specifying Kerberos 5 Cross-Realm Authentication. In *Proc. WITS'05*, pages 12–26. ACM Digital Lib., 2005.

[CW05] Vèronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.

[DCB01] Jan De Clercq and Micky Balladelli. Windows 2000 authentication. `http://www.windowsitlibrary.com/Content/617/06/6.html`, 2001. Digital Press.

[DDM+05] Anupam Datta, Ante Derek, John Mitchell, Vitalij Shmatikov, and Matthieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2005.

[DDMW06] Anupam Datta, Ante Derek, John Mitchell, and Bogdan Warinschi. Key exchange protocols: Security definition, proof method, and applications. In *19th IEEE Computer Security Foundations Workshop (CSFW 19)*, Venice, Italy, 2006. IEEE Press.

[DY83] Danny Dolev and Andrew Yao. On the security of public-key protocols. *IEEE Trans. Info. Theory*, 2(29):198–208, 1983.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[GTZ01] J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 186–195, 2001.

[HLM03] Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer, 2003.

[HM05] C. He and J. C. Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '05)*, February 2005.

[IET06] 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/`.

[IK03] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.

[Lau01] Peeter Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.

[Lau04] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[Mea99] Catherine Meadows. Analysis of the internet key exchange protocol using the nrl protocol analyzer. In *Proc. IEEE Symp. Security and Privacy*, pages 216–231, 1999.

[MMS98] J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.

[MMST01] John Mitchell, Mark Mitchell, Andre Scedrov, and Vanessa Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.

[MW04] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[NT94] Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, September 1994.

[NYHR05] Clifford Neuman, Tom Yu, Sam Hartman, and Kenneth Raeburn. The Kerberos Network Authentication Service (V5), July 2005. `http://www.ietf.org/rfc/rfc4120`.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.

Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, http://eprint.iacr.org/.
[The] The Internet Engineering Task Force. http://www.ietf.org.

A Additional Algorithms

For completeness, this appendix collects the algorithms omitted from the main body of this paper. The algorithms for Public-key Kerberos are in Fig. 7–12. The algorithms for Kerberos 5 are in Fig. 11–16. Note that the algorithms for the TGS T and a server S (i.e. Algorithms 4 and 5 in Fig. 11 and 12) are identical for Public-key Kerberos and Kerberos 5.

B Additional Proof Sketches

B.1 Conventions

In the following we will use following convention for the algorithms:

Convention 1 *Let* Kerb $\in \{\mathrm{PK}, \mathrm{K5}\}$. *For all* $w \in \{1, ..., n\} \cup \{S_1, ..., S_l\} \cup \{K, T\}$ *the following holds. If* M_w^{Kerb} *enters a command at port* $\mathrm{in}_w!$ *and receives* $\downarrow$ *at port* $\mathrm{out}_w?$ *as the immediate answer of the cryptographic library, then* M_w^{Kerb} *aborts the execution of the current algorithm, except if the command was of the form* list_proj *or* send_i.

B.2 Auxiliary Properties

In the following we will consider the auxiliary properties for Public-key Kerberos. Let ck be a Message Authentication Code of some list/message m with some key k (instead of a keyed Hash Function H_k).

Handles contained in the sets $Nonce_u$ and $Nonce2_u$ are indeed handles of u to handles.

Lemma 1 (Correct Nonce Owner).
For all $u \in \mathcal{H}$, *and* $(x^{hnd}, ...) \in Nonce_u$ *or* $(x^{hnd}, ...) \in Nonce2_u$, *it holds* $D[hnd_u = x^{hnd}] \neq \downarrow$ *and* $D[hnd_u = x^{hnd}].type = nonce$.

Proof. Let $(x^{hnd}, ...) \in Nonce_u$. By construction, this entry has been added to $Nonce_u$ by M_u^{PK} in step 1A.8. x^{hnd}has been generated by the command gen_nonce() at some time t, input at port $\mathrm{in}_u?$ of $\mathrm{TH}_{\mathcal{H}}$. Convention 1 implies $x^{hnd} \neq \downarrow$, as M_u^{PKINIT} would abort otherwise and not add the entry to $Nonce_u$. By definition of gen_nonce() and using Lemma 5.2 of [Bac04] one gets that $D[hnd_u = x^{hnd}] \neq \downarrow$ and $D[hnd_u = x^{hnd}].type =$ nonce holds (the proof of the statement for $Nonce2_u$ is analogous).

If K generated a symmetric key k or AK for v (i.e., on receiving a AS_REQ from v) and w has a handle to k or AK then w must either be v or K. And if T generated a symmetric key SK for v and server S and w has a handle to SK, then w must be either v, T or S.

Input: (v, u, i, m^{hnd}) at out_u?

1. **if** $v = K$ **then** {AS_REP is input}
2. $c_i^{hnd} \leftarrow \text{list\_proj}(m^{hnd}, i)$ for $i = 1, 2, 3, 4$ $\{c_1 \approx \{\{Cert_K, [k, ck]_{sk_K}\}\}_{pk_C}, c_2 \approx C, c_3 \approx TGT, c_4 \approx \{AK, n_1, t_K, T\}_k\}$
3. $c_2 \leftarrow \text{retrieve}(c_2^{hnd})$
4. **if** $(c_2 \neq u)$ **then**
5. Abort
6. **end if**
7. $l_1^{hnd} \leftarrow \text{decrypt}(ske_u^{hnd}, c_1^{hnd})$ $\{l_1 \approx Cert_K, [k, ck]_{sk_K}\}$
8. $l_{1.i}^{hnd} \leftarrow \text{list\_proj}(l_1^{hnd}, i)$ for $i = 1, 2$ $\{l_{1.1} \approx Cert_K, l_{1.2} \approx [k, ck]_{sk_K}\}$
9. $K^{hnd} \leftarrow \text{store}(K)$
10. $b \leftarrow \text{verify\_cert}(pke_{CA}^{hnd}, l_{1.1}^{hnd}, K^{hnd})$
11. **if** $b = false$ **then**
12. Abort
13. **end if**
14. $pke_K^{hnd} \leftarrow \text{pk\_of\_cert}(pke_{CA}^{hnd}, l_{1.1}^{hnd})$
15. $z_{1.2}^{hnd} \leftarrow \text{msg\_of\_sig}(l_{1.2})$ $\{z_{1.2} \approx k, ck\}$
16. $b \leftarrow \text{verify}(l_{1.2}^{hnd}, pke_k^{hnd}, z_{1.2}^{hnd})$
17. **if** $(b = false)$ **then**
18. Abort
19. **end if**
20. $x_i^{hnd} \leftarrow \text{list\_proj}(z_{1.2}, i)$ for $i = 1, 2$ $\{x_1 \approx k, x_2 \approx ck\}$
21. $type_i \leftarrow \text{get\_type}(x_i^{hnd})$ for $i = 1, 2$
22. **if** $(type_1 \neq \text{skse}) \vee (type_2 \neq \text{auth})$ **then**
23. Abort
24. **end if**
25. $l_4^{hnd} \leftarrow \text{sym\_decrypt}(x_1^{hnd}, c_4^{hnd})$ $\{x_1 \approx k, c_4 \approx \{AK, n_1, t_K, T\}_k\}$
26. $y_i^{hnd} \leftarrow \text{list\_proj}(l_4^{hnd}), i$ for $i = 1, 2, 4$ $\{y_1 \approx AK, y_2 \approx n_1, y_4 \approx T\}$
27. $type_3 \leftarrow \text{get\_type}(y_1^{hnd})$
28. $type_4 \leftarrow \text{get\_type}(y_2^{hnd})$
29. $y_4 \leftarrow \text{retrieve}(y_4^{hnd})$
30. **if** $(type_3 \neq \text{skse}) \vee (type_4 \neq \text{nonce}) \vee (y_4 \neq T) \vee (\nexists! \tilde{m}^{hnd} : (y_2^{hnd}, \tilde{m}^{hnd}) \in Nonce_u)$ **then**
31. Abort
32. **end if**
33. $b \leftarrow \text{auth\_test}(x_2^{hnd}, x_1^{hnd}, \tilde{m}^{hnd})$ $\{x_2 \approx ck = H_k(\tilde{m})\}$
34. **if** $(b = false)$ **then**
35. Abort
36. **end if**
37. $TGTicket_u := TGTicket_u \cup \{(c_3^{hnd}, y_1^{hnd}, T)\}$ $\{c_3 \approx TGT, y_1 \approx AK\}$
38. output $(\text{ok}, \text{KAS\_exchange PK}, K, T, y_1^{hnd}, c_3^{hnd})$ at KA_out_u!

Fig. 9. Algorithm 3 of Public-key Kerberos, part 1: Behavior of user in after initialization

39. **else if** $v = T$ **then** {TGS_REP is input}
40. $d_i^{hnd} \leftarrow$ list_proj(m^{hnd}, i) for $i = 1, 2, 3$ $\{d_1 \approx C,\ d_2 \approx ST,\ d_3 \approx \{SK, n_3, t_T S\}_{AK}\}$
41. $d_1 \leftarrow$ retrieve(d_1^{hnd})
42. **if** $(d_1 \neq u)$
    $\vee\ (\nexists!\,(., AK^{hnd}, T) \in TGTicket_u) :$ sym_decrypt$(AK^{hnd}, d_3^{hnd}) \neq\downarrow)$ **then**
43. Abort
44. **end if**
45. $l_2^{hnd} \leftarrow$ sym_decrypt(AK^{hnd}, d_3^{hnd}) $\{l_2 \approx SK, n_3, t_T, S\}$
46. $x_{2.i}^{hnd} \leftarrow$ list_proj(l_2^{hnd}, i) for $i = 1, 2, 4$ $\{x_{2.1} \approx SK,\ x_{2.2} \approx n_3,\ x_{2.4} \approx S\}$
47. $type_5 \leftarrow$ get_type$(x_{2.1}^{hnd})$
48. $type_6 \leftarrow$ get_type$(x_{2.2}^{hnd})$
49. **if** $(type_5 \neq$ skse$) \vee (type_6 \neq$ nonce$) \vee ((x_{2.2}^{hnd}, T, .) \notin Nonce2_u)$ **then**
50. Abort
51. **end if**
52. $x_3^{hnd} \leftarrow$ list$(u^{hnd}, t_u'^{hnd})$ $\{x_3 \approx C, t_C'\}$
53. $m_{3.2}^{hnd} \leftarrow$ sym_encrypt$(x_{2.1}^{hnd}, x_3^{hnd})$ $\{m_{3.2} \approx \{C, t_C'\}_{SK}\}$
54. $m_3^{hnd} \leftarrow$ list$(d_2^{hnd}, m_{3.2}^{hnd})$ $\{m_3 \approx ST, \{C, t_C'\}_{SK}\}$
55. $S \leftarrow retrieve(x_{2.4}^{hnd})$
56. $Session\_KeysS_u := Session\_KeysS_u \cup \{(S, x_{2.1}^{hnd})\}$ $\{x_{2.1} \approx SK\}$
57. send_i(S, m_3^{hnd})
58. **else if** $v = S \in \{S_1, ..., S_l\}$ **then** {AP_REP is input}
59. **if** $(\nexists!(S, SK^{hnd}) \in Session\_KeysS_u$: sym_decrypt$(SK^{hnd}, m^{hnd}) \neq\downarrow)$ **then**
60. Abort
61. **end if**
62. $l_3^{hnd} \leftarrow$ sym_decrypt(SK^{hnd}, m^{hnd}) $\{m \approx \{t_C'\}_{SK}\}$
63. $x_{3.1}^{hnd} \leftarrow$ list_proj$(l_3^{hnd}, 1)$ $\{x_{3.1} \approx t_C'\}$
64. $x_{3.1} \leftarrow retrieve(x_{3.1}^{hnd})$
65. **if** $x_{3.1} = u$ **then**
66. Abort
67. **end if**
68. output $(\text{ok}, \text{PK}, S, SK^{hnd})$ at KA_out$_u$!

Fig. 10. Algorithm 3 of Public-key Kerberos, part 2: Behavior of user after initialization

Input: (v, T, i, m^{hnd}) at out_T? with $v \in \{1, ..., n\}$.

1. $x_i^{hnd} \leftarrow \text{list\_proj}(m^{hnd}, i)$ for $i = 1, 2, 3, 4, 5$ $\{x_1 \approx TGT,\ x_2 \approx \{C, t_C\}_{AK},\ x_3 S,\ x_5 \approx n_3\ \}$
2. $y_1^{hnd} \leftarrow \text{sym\_decrypt}(skse_{KT}^{hnd}, x_1^{hnd})$ $\{y_1 \approx AK, C, t_K\}$
3. $y_{1.i}^{hnd} \leftarrow \text{list\_proj}(y_1^{hnd}, i)$ for $i = 1, 2$ $\{y_{1.1} \approx AK,\ y_{1.2} \approx C\}$
4. $type_1 \leftarrow \text{get\_type}(y_{1.1}^{hnd})$
5. $type_2 \leftarrow \text{get\_type}(x_5^{hnd})$
6. $x_i \leftarrow \text{retrieve}(x_i^{hnd})$ for $i = 3, 4$
7. $y_{1.2} \leftarrow \text{retrieve}(y_{1.2}^{hnd})$
8. **if** $(type_1 \neq \text{skse}) \vee (type_2 \neq \text{nonce}) \vee ((x_5^{hnd}, v) \in Nonce_T) \vee (x_4 \notin \{S_1, ..., S_l\}) \vee (y_{1.2} \neq v)$**then**
9. Abort
10. **end if**
11. $Nonce4_T := Nonce4_T \cup \{(x_5^{hnd}, v)\}$
12. $z^{hnd} \leftarrow \text{sym\_decrypt}(y_{1.1}^{hnd}, x_2^{hnd})$ $\{z \approx C, t_C\}$
13. $z_1^{hnd} \leftarrow \text{list\_proj}(z^{hnd}, 1)$ $\{z_1 \approx C\}$
14. $z_1 \leftarrow \text{retrieve}(z_1^{hnd})$
15. **if** $(z_1 \neq v)$ **then**
16. Abort
17. **end if**
18. $SK^{hnd} \leftarrow \text{gen\_symenc\_key}()$
19. $l^{hnd} \leftarrow \text{list}(SK^{hnd}, z_1^{hnd}, t_T^{hnd})$ $\{l \approx SK, n_3, t_T\}$
20. $ST^{hnd} \leftarrow \text{sym\_encrypt}(skse_{TS}^{hnd}, l^{hnd})$ $\{ST \approx \{SK, C\}_{k_S}\}$
21. $\tilde{l}^{hnd} \leftarrow \text{list}(SK^{hnd}, x_5^{hnd}, t_T^{hnd}, x_3^{hnd})$ $\{\tilde{l} \approx SK, n_3, t_T, S\}$
22. $m_{4.3}^{hnd} \leftarrow \text{sym\_encrypt}(y_{1.1}^{hnd}, \tilde{l}^{hnd})$ $\{m_{4.3} \approx \{SK, n_3, t_T, S\}_{AK}\}$
23. $m_4^{hnd} \leftarrow \text{list}(x_3^{hnd}, ST^{hnd}, m_{4.3}^{hnd})$ $\{m_4 \approx C, ST, \{SK, n_3, t_T, S\}_{AK}\}$
24. $\text{send\_i}(v, m_4^{hnd})$

Fig. 11. Algorithm 4: Behavior of TGS

Input: (v, S, i, m^{hnd}) at out_S? with $v \in \{1, ..., n\}$.

1. $m_{5.i}^{hnd} \leftarrow$ list_proj(m^{hnd}, i) for $i = 1, 2$ $\quad \{m_{5.1} \approx ST,\ m_{5.2} \approx \{C, t'_C\}_{SK}\}$
2. $x^{hnd} \leftarrow$ sym_decrypt$(skse_{TS}^{hnd}, m_{5.1}^{hnd})$
3. $x_i^{hnd} \leftarrow$ list_proj(x^{hnd}, i) for $i = 1, 2$ $\quad \{x_1 \approx SK,\ x_2 \approx C\}$
4. $x_2 \leftarrow$ retrieve(x_2^{hnd})
5. $type_1 \leftarrow$ get_type(x_1^{hnd})
6. **if** $(type_1 \neq \text{skse}) \vee (x_2 \neq v)$ **then**
7. Abort
8. **end if**
9. $y^{hnd} \leftarrow$ sym_decrypt$(x_1^{hnd}, m_{5.2}^{hnd})$ $\quad \{y \approx C, t'_C\}$
10. $y_i^{hnd} \leftarrow$ list_proj(y^{hnd}, i) for $i = 1, 2$ $\quad \{y_1 \approx C,\ y_2 \approx t'_C\}$
11. $y_1 \leftarrow$ retrieve(y_1^{hnd})
12. **if** $(y_1 \neq v)$ **then**
13. Abort
14. **end if**
15. $m_6^{hnd} \leftarrow$ sym_encrypt(x_1^{hnd}, y_2^{hnd}) $\quad \{m_6 \approx \{t'_C\}_{SK}\}$
16. send_i(S, m_6^{hnd})
17. output $(\text{ok}, \text{PK}, v, x_1^{hnd})$ at KA_out_S!

Fig. 12. Algorithm 5: Behavior of server

Lemma 2 (Key Secrecy). *For all $u, v \in \mathcal{H}$, honest K, T, and $S \in \{S_1, ..., S_l\}$, and for all $j \leq size$ with $D[j].type = skse$:*

a) If $D[j]$ was created by M_K^{PK} in step 2.24 then (with the notation of Algorithm 2 (Fig. 8))

$$D[j].hnd_w \neq \downarrow \quad \textit{implies } w \in \{v, K\}.$$

b) If $D[j]$ was created by M_K^{PK} in step 2.25 then (with the notation of Algorithm 2 (Fig. 8))

$$D[j].hnd_w \neq \downarrow \quad \textit{implies } w \in \{v, K, T\}.$$

c) If $D[j]$ was created by M_T^{PK} in step 4.18 then (with the notation of Algorithm 4 in Fig. 11)

$$D[j].hnd_w \neq \downarrow \quad \textit{implies } w \in \{v, T, S\}$$

where with the notation of Algorithm 4, $S = x_4$.

Proof. a) Let $j \leq size$, $D[j].type =$ skse such that $D[j]$ was created by M_K^{PK} in step 2.24 at time t. The output in step 2.36 contains $D[j]$ encrypted with v's public key. Since, by assumption, handles to private keys are not allowed to be sent around, only v can decrypt the output and get a handle to $D[j]$ after output in step 2.36. But v never sends $D[j]$ as part of a new list for time $t' > t$ (by Algorithms 1 and 3). One immediately gets $D[j].hnd_a = \downarrow$ for all $t' > t$.

b) Let $j \leq size$, $D[j].type =$ skse such that $D[j]$ was created by M_K^{PK} in step 2.25 at time t. The output in step 2.36 contains $D[j]$ encrypted under the symmetric key created by M_K^{PK} in step 2.24 (see step 2.33, 2.34). By Key Secrecy a), only v or K can decrypt that part of the output in step 2.36. The output in

A) Input:$(\text{new\_prot}, \text{K5}, K, T)$ at KA_in_u? .

1. $n_{u,1}^{hnd} \leftarrow \text{gen\_nonce}()$
2. $u^{hnd} \leftarrow \text{store}(u)$
3. $T^{hnd} \leftarrow \text{store}(T)$
4. $m_1^{hnd} \leftarrow \text{list}(u^{hnd}, T^{hnd}, n_{u,1}^{hnd})$ $\{m_1 \approx C, T, n_1\}$
5. $Nonce_u := Nonce_u \cup \{(n_{u,1}^{hnd}, K)\}$
6. $\text{send\_i}(K, m_1^{hnd})$

B) Input:$(\text{continue\_prot}, \text{K5}, T, S, AK^{hnd})$ at KS_in_u? for $S \in \{S_1, ..., S_l\}$

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. $m_2^{hnd} \leftarrow \text{list}(TGT^{hnd}, auth^{hnd}, S^{hnd}, n_{u,3}^{hnd})$ $\{m_2 \approx TGT, \{C, t_C\}_{AK}, C, S, n_3\}$
9. $\text{send\_i}(T, m_2^{hnd})$

Fig. 13. Algorithm 1 of Kerberos 5: Evaluation of inputs from the user (starting the AS and TG exchanges).

step 2.36 also contains $D[j]$ encrypted under a symmetric key shared exclusively between K and T ($\text{skse}_{K,T}^{hnd}$; see step 2.32). By construction of Algorithm 4 (Fig. 11) and since T is honest, one sees that T does not send out a list containing $D[j]$ for time $t' > t$. Also, by construction of Algorithms 1 and 3 (Fig. 7,9, 10) and since v and K are honest, one sees that v and K do not send out a list containing $D[j]$ for time $t' > t$.

c) Let $j \leq size$, $D[j].type = \text{skse}$ such that $D[j]$ was created by M_T^{PK} in step 4.18 at time t. The output in step 4.24 contains $D[j]$ in a list ST^{hnd} encrypted under a symmetric key shared exclusively between T and S (skse_{TS}^{hnd}; see step 4.20) and also in a list $m_{4.3}^{hnd}$ encrypted under a second symmetric key for which T gets a handle in step 4.3, i.e. after decryption with the symmetric key shared exclusively between T and K (skse_{KT}^{hnd}; see step 4.1), otherwise there would be an abort, by Convention 1. Since, by construction, M_T^{PK} does not use the key skse_{KT}^{hnd} for encryption, M_K^{PK} must have created the ciphertext containing the second symmetric key. Key Secrecy b) implies that only v, T, K have handles to this key. T and K do not use this second key for decryption, therefore only v can get a handle to $D[j]$ through decryption with this second key. Also, only S uses skse_{TS}^{hnd} for decryption (in step 5.2). But, by construction, neither S nor v send out $D[j]$ as part of a newly created list for time $t' > t$.

If user u receives a valid AS_REP message then this message was indeed generated by K for u and an adversary cannot learn the contained symmetric keys.

Input:(v, K, i, m^{hnd}) at out_K? with $v \in \{1, ..., n\}$.

1. $x_i^{hnd} \leftarrow \text{list\_proj}(m^{hnd}, i)$ for $i = 1, 2, 3$ $\{x_1 \approx C, x_2 \approx T, x_3 \approx n_1\}$
2. $type_1 \leftarrow \text{get\_type}(x_3^{hnd})$
3. $x_i \leftarrow \text{retrieve}(x_i^{hnd})$ for $i = 1, 2$
4. **if** $(type_1 \neq \text{nonce}) \vee ((x_3, .) \in Nonce3_K) \vee (x_1 \neq v) \vee (x_2 \neq T)$ **then**
5. Abort
6. **end if**
7. $v^{hnd} \leftarrow \text{store}(v)$
8. $Nonce3_K := Nonce3_K \cup \{(x_3^{hnd}, v)\}$
9. $AK^{hnd} \leftarrow \text{gen\_symenc\_key}()$
10. $z_1^{hnd} \leftarrow \text{list}(AK^{hnd}, v^{hnd}, t_K^{hnd})$ $\{z_1 \approx AK, C, t_K\}$
11. $TGT^{hnd} \leftarrow \text{sym\_encrypt}(skse_{K,x_2}^{hnd}, z_1^{hnd})$ $\{TGT \approx \{AK, C, t_K\}_{k_T}\}$
12. $z_2^{hnd} \leftarrow \text{list}(AK^{hnd}, x_3^{hnd}, t_K^{hnd}, x_2^{hnd})$ $\{z_2 \approx AK, n_1, t_K, T\}$
13. $m_{23} \leftarrow \text{sym\_encrypt}(k_v^{hnd}, z_2^{hnd})$ $\{m_{23} \approx \{AK, n_1, t_K, T\}_{k_C}\}$
14. $m_2^{hnd} \leftarrow \text{list}(v^{hnd}, TGT^{hnd}, m_{23}^{hnd})$ $\{m_2 \approx C, TGT, \{AK, n_1, t_K, T\}_{k_C}\}$
15. $\text{send\_i}(v, m_2^{hnd})$

Fig. 14. Algorithm 2 of Kerberos 5: Behavior of the KAS

Input: (v, u, i, m^{hnd}) at out_u?

1. **if** $v = K$ **then** {AS_REP is input}
2. $c_i^{hnd} \leftarrow \text{list\_proj}(m^{hnd}, i)$ for $i = 1, 2, 3$ $\{c_1 \approx C, c_2 \approx TGT, c_3 \approx \{AK, n_1, t_K, T\}_{k_C}\}$
3. $c_1 \leftarrow \text{retrieve}(c_1^{hnd})$
4. **if** $(c_1 \neq u)$ **then**
5. Abort
6. **end if**
7. $type_i \leftarrow \text{get\_type}(c_i^{hnd})$ for $i = 2, 3$
8. **if** $(type_2 \neq \text{skse}) \vee (type_3 \neq \text{auth})$ **then**
9. Abort
10. **end if**
11. $l_3^{hnd} \leftarrow \text{sym\_decrypt}(skse_{uK}^{hnd}, c_3^{hnd})$ $\{l_3 \approx AK, n_1, t_K, T\}$
12. $y_i^{hnd} \leftarrow \text{list\_proj}(l_3^{hnd}), i$ for $i = 1, 2, 4$ $\{y_1 \approx AK, y_2 \approx n_1, y_4 \approx T\}$
13. $type_4 \leftarrow \text{get\_type}(y_1^{hnd})$
14. $type_5 \leftarrow \text{get\_type}(y_2^{hnd})$
15. $y_4 \leftarrow \text{retrieve}(y_4^{hnd})$
16. **if** $(type_3 \neq \text{skse}) \vee (type_4 \neq \text{nonce}) \vee (y_4 \neq T) \vee (\nexists! (\tilde{n}^{hnd}, K) \notin Nonce_u)$ **then**
17. Abort
18. **end if**
19. $TGTicket_u := TGTicket_u \cup \{(c_2^{hnd}, y_1^{hnd}, T)\}$
20. output $(\text{ok}, \text{KAS\_exchange K5}, K, T, y_1^{hnd}, c_2^{hnd})$ at KA_out_u!

Fig. 15. Algorithm 3 of Kerberos 5, part 1: Behavior of user after initialization

21. **else if** $v = T$ **then** {TGS_REP is input}
22. $d_i^{hnd} \leftarrow$ list_proj(m^{hnd}, i) for $i = 1, 2, 3$ { $d_1 \approx C$, $d_2 \approx ST$, $d_3 \approx \{SK, n_3, t_T, S\}_{AK}$ }
23. $d_1 \leftarrow$ retrieve(d_1^{hnd})
24. **if** $(d_1 \neq u)$
$\vee\ (\nexists!\,(., AK^{hnd}, T) \in TGTicket_u) :$ sym_decrypt$(AK^{hnd}, d_3^{hnd}) \neq\downarrow)$ **then**
25. Abort
26. **end if**
27. $l_2^{hnd} \leftarrow$ sym_decrypt(AK^{hnd}, d_3^{hnd}) $\{l_2 \approx SK, n_3, t_T, S\}$
28. $x_{2.i}^{hnd} \leftarrow$ list_proj(l_2^{hnd}, i) for $i = 1, 2, 4$ $\{x_{2.1} \approx SKey,\ x_{2.2} \approx n_3,\ x_{2.4} \approx S\}$
29. $type_5 \leftarrow$ get_type$(x_{2.1}^{hnd})$
30. $type_6 \leftarrow$ get_type$(x_{2.2}^{hnd})$
31. **if** $(type_5 \neq$ skse$) \vee (type_6 \neq$ nonce$) \vee ((x_{2.2}^{hnd}, T, .) \notin Nonce2_u)$ **then**
32. Abort
33. **end if**
34. $x_3^{hnd} \leftarrow$ list$(u^{hnd}, t_u'^{hnd})$ $\{x_3 \approx C, t_C'\}$
35. $m_{3.2}^{hnd} \leftarrow$ sym_encrypt$(x_{2.1}^{hnd}, x_3^{hnd})$ $\{m_{3.2} \approx \{C, t_u'\}_{SK}\}$
36. $m_3^{hnd} \leftarrow$ list$(d_2^{hnd}, m_{3.2}^{hnd})$ $\{m_3 \approx ST, \{C, t_u'\}_{SK}\}$
37. $S \leftarrow retrieve(x_{2.4}^{hnd})$
38. $Session\_KeysS_u := Session\_KeysS_u \cup \{(S, x_{2.1}^{hnd})\}$
39. send_i(S, m_3^{hnd})
40. **else if** $v = S \in \{S_1, ..., S_l\}$ **then** {AP_REP is input}
41. **if** $(\nexists!(S, SK^{hnd}) \in Session\_KeysS_u$: sym_decrypt$(SK^{hnd}, m^{hnd}) \neq\downarrow)$ **then**
42. Abort
43. **end if**
44. $l_3^{hnd} \leftarrow$ sym_decrypt(SK^{hnd}, m^{hnd}) $\{m \approx \{t_C'\}_{SK}\}$
45. $x_{3.1}^{hnd} \leftarrow$ list_proj$(l_3^{hnd}, 1)$ $\{x_{3.1} \approx t_C'\}$
46. $x_{3.1} \leftarrow retrieve(x_{3.1}^{hnd})$
47. **if** $x_{3.1} = u$ **then**
48. Abort
49. **end if**
50. output $(\text{ok}, \text{K5}, S, SK^{hnd})$ at KA_out$_u$!

Fig. 16. Algorithm 3 of Kerberos 5, part 2: Behavior of user after initialization

Lemma 3 (Authentication of KAS to client and Secrecy of AK). *For all $u, v \in \mathcal{H}$, honest KAS K and TGS T, and for all $j \leq size$ with $D[j].type = list$ and $j^{hnd} := D[j].hnd_u \neq \downarrow$:*
If $l_i := D[j].arg[i]$ for $i = 1, 4$
with $D[l_1].type = enc$ and $D[l_4].type = symenc$,
$x_1 := D[l_1].arg[2]$ *with* $D[x_1].type = list$, $\{\approx cert_K, [k, ck]_{sk_K}\}$
$x_{1.1} := D[x_1].arg[1]$ *with* $D[x_{1.1}].type = cert$,
$x_{1.2} := D[x_1].arg[2]$ *with* $D[x_{1.2}].type = sig$, $\{\approx [k, ck]_{sk_K}\}$
$y_{1.1} := D[x_{1.1}].arg[2]$ *with* $D[x_{1.1}].type = pke$,
$y_{1.2} := D[x_{1.2}].arg[2]$ *with* $D[y_{1.2}].type = list$, $\{\approx k, ck\}$
$s_1 := D[y_{1.2}].arg[1]$ *with* $D[s_1].type = skse$,
$r_1 := D[y_{1.2}].arg[2]$ *with* $D[r_1].type = auth$,
$q_1 := D[r_1].arg[1]$ *with* $D[q_1].type = list$, $\{\approx m_1\}$
$p_1 := D[r_1].arg[2]$ *with* $D[p_1].type = pkse$,
$x_4 := D[l_4].arg[1]$ *with* $D[x_4].type = list$, $\{\approx AK, n_1, t_k, T\}$
$y_4 := D[x_4].arg[2]$ *with* $D[y_4].type = nonce$, $\{\approx n_1\}$
and if furthermore

a) $pke := D[l_1].arg[1]$ *and* $D[pke].hnd_u \neq \downarrow$
b) $D[x_{1.2}].arg[1] = D[x_{1.1}].arg[2]$
c) $p_1 + 1 = D[l_4].arg[2] = y_{1.1} + 1$
d) $(D[y_4].hnd_u,\ D[q_1].hnd_u,\ K) \in Nonce_u$

then $D[l_1]$ was created by M_K^{PK} in step 2.30 and $D[l_4]$ was created by M_K^{PK} in step 2.34 and both their indices were arguments of a list created by M_K^{PK} in step 2.35.
Furthermore, Key Secrecy implies that $D[s_1].hnd_a = \downarrow$, and therefore also $D[x_4.arg[1]].hnd_a = \downarrow$.

Proof. By the definition of the commands sign and verify together with b) one gets that the list $(y_{1.1}, y_{1.2})(\approx \{k, ck\})$ was sent by $\mathrm{M}_K^{\mathrm{PK}}$, in particular, by construction of Algorithm 2, symmetric key $y_{1.1}$ was generated by $\mathrm{M}_K^{\mathrm{PK}}$ in step 2.24 . Key Secrecy gives that the encryption of list $(., y_4, .)$, for the nonce y_4, under key $y_{1.1}$ was generated by $\mathrm{M}_K^{\mathrm{PK}}$ in step 2.34. Therefore there must have been an input (u, K, i, m^{hnd}) at out_K? where the list m contains y_4 (see step 2.2-2.5 and 2.33-2.34). Otherwise the algorithm would stop by Convention 1. Since $(D[y_4].hnd_u,\ D[q_1].hnd_u,\ K) \in Nonce_u$, Correct Nonce Owner implies that it was stored there by $\mathrm{M}_u^{\mathrm{PK}}$ while running Algorithm 1A, in particular $\mathrm{M}_u^{\mathrm{PK}}$ must have sent list of the form q_1 to K. And since $y_{1.2}$ ($\approx ck =$ message authentication code of message m_1 with key k) was sent by $\mathrm{M}_K^{\mathrm{PK}}$ as part of a message, by definition of auth and by Key Secrecy, $\mathrm{M}_K^{\mathrm{PK}}$ must have created this ck in step 2.26 which only occurs on response to receiving the request q_1 from u (step 2.1 - 2.21), otherwise there would have been an abort, by Convention 1.

If 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.

Lemma 4 (TGS Authentication of the TGT). *For all honest KAS K and TGS T and for all $j \leq size$ with $D[j].type = list$ and $j^{hnd} := D[j].hnd_T \neq \downarrow$:*
$l_1 := D[j].arg[1]$ *with* $D[l_1].type = symenc$, $\{\approx TGT\}$
$l_2 := D[j].arg[2]$ *with* $D[l_2].type = symenc$, $\{\approx \{u, t_u\}_{AK}\}$
$x_1 := D[l_1].arg[1]$ *with* $D[x_1].type = skse$, $\{\approx k_{KT}\}$
$x_2 := D[l_1].arg[2]$ *with* $D[x_2].type = list$,
$x_{2.1} := D[x_2].arg[1]$ *with* $D[x_{1.1}].type = skse$, $\{\approx AK\}$
$x_{2.2} := D[x_2].arg[2]$ *with* $x_{2.2} \in \mathcal{H}$
$y_1 := D[l_2].arg[1]$ *with* $D[x_1].type = skse$, $\{\approx AK\}$
$y_2 := D[l_2].arg[2]$ *with* $D[y_2].type = list$,
$y_{2.1} := D[y_2].arg[1]$ *with* $y_{2.1} \in \mathcal{H}$
and if furthermore

a) $D[x_1] = skse_{KT}$
b) $D[x_{2.1}] = D[y_1]$
c) $x_{2.2} = y_2$

then entry $D[l_1]$ was generated by M_K^{PK} in step 2.32 at a time t and entry $D[l_2]$ was generated by M_u^{PK} in step 1B.4 at a time $t' > t$ for $u = y_2$.

Proof. By assumption, only $\mathrm{M}_K^{\mathrm{PK}}$ and $\mathrm{M}_T^{\mathrm{PK}}$ have handles to the long-term key $skse_{KT}$. But since, by construction of Algorithm 4 (Fig. 11), $\mathrm{M}_T^{\mathrm{PK}}$ does not use this key for encryption, $\mathrm{M}_K^{\mathrm{PK}}$ must have generated $D[l_1]$ in step 2.32. The entry $D[l_1]$ contains a symmetric key $x_{2.1}$ and the name of a user $x_{2.2} = y_2$. Part i) implies that y_2 must have generated $D[l_2]$.

If user u receives a valid TGS_REP then it was generated by T for u and S. And an adversary cannot learn the contained session key SK.

Lemma 5 (Authentication of TGS to client and Secrecy of SK). *For all $u \in \mathcal{H}$ and for all $j \leq size$ with $D[j].type = symenc$ and $j^{hnd} := D[j].hnd_u \neq \downarrow$:*
$x_1 := D[j].arg[1]$ *with* $D[x_1].type = skse$,
$x_2 := D[j].arg[2]$ *with* $D[x_2].type = list$,
$x_{2.1} := D[x_2].arg[1]$ *with* $D[x_{2.1}].type = skse$,
$x_{2.2} := D[x_2].arg[2]$ *with* $D[x_{2.2}].type = nonce$,
$x_{2.3} := D[x_2].arg[3]$,
and if furthermore

a) $(., x_1^{hnd}, T) \in TGTicket_u$
b) $(x_{2.2}^{hnd}, T, x_{2.3}) \in Nonce2_u$

then $D[j]$ was created by M_T^{PK} in step 4.20.
Furthermore, Key Secrecy implies $D[x_1].hnd_a = \downarrow$ (since not more than one honest can have a handle to x_1 besides K and T) .

Proof. a) guarantees that $\mathrm{M}_u^{\mathrm{PK}}$ has a handle to the symmetric key x_1 needed to decrypt the message in $D[j]$. a) also ensures that x_1 is generated by $\mathrm{M}_K^{\mathrm{PK}}$, and therefore Key Secrecy implies that an adversary cannot have a handle to x_1, since at most one honest user u can have a handle to x_1 besides K and T. But $\mathrm{M}_u^{\mathrm{PK}}$ does not generate a list of the form of $D[j]$, neither does K. So T must have generated $D[j]$ in step 4.20. By construction of Algorithm 4, u must have sent an corresponding request to T (steps 4.1-4.9),otherwise, by Convention 1, Algorithm 4 would abort.

If server S receives a 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.

Lemma 6 (Server Authentication of the ST). *For all $u, K, T, S \in \mathcal{H}$ and for all $j \leq size$ with $D[j].type =$list and $j^{hnd} := D[j].hnd_S \neq \downarrow$:*
$l_1 := D[j].arg[1]$ with $D[l_1].type =$symmenc,
$l_2 := D[j].arg[2]$ with $D[l_2].type =$symmenc,
$x_1 := D[l_1].arg[1]$ with $D[x_1].type =$skse,
$x_2 := D[l_1].arg[1]$ with $D[x_2].type =$list,
$x_{2.1} := D[x_2].arg[1]$ with $D[x_{2.1}].type =$skse,
$x_{2.2} := D[x_2].arg[2]$ with $x_{2.2} \in \mathcal{H}$,
$y_1 := D[l_2].arg[1]$ with $D[y_1].type =$skse,
$y_2 := D[l_2].arg[2]$ with $D[y_2].type =$list,
$y_{2.1}; = D[y_2].arg[1]$ with $y_{2.1} \in \mathcal{H}$ and if furthermore

a) $D[x_1] = k_S$
b) $D[x_{2.1}] = D[y_1]$
c) $x_{2.2} = y_{2.1}$

then $D[l_1]$ was generated by M_T^{PK} in step 4.20 at time t and $D[l_2]$ was generated by M_u^{PK} in step 3.53 at time $t' > t$, where $x_{2.2} = u$.

Proof. By assumption, only $\mathrm{M}_T^{\mathrm{PK}}$ and $\mathrm{M}_S^{\mathrm{PK}}$ have handles to the key $k_S = D[x_1]$. But since, by construction, $\mathrm{M}_S^{\mathrm{PK}}$ does not use the key k_S for encryption, $D[l_1]$ must have been generated by $\mathrm{M}_T^{\mathrm{PK}}$ in step 4.20. This step only occurs after step 4.18, in which $\mathrm{M}_T^{\mathrm{PK}}$ generated a symmetric key, and after receiving valid TGS_REQ from u (steps 4.1 - 4.9), otherwise there would be an abort, by Convention 1. The key generated in step 4.18 must be the key contained in $D[l_1]$, i.e. the key is $D[x_{2.1}]$. By Key Secrecy, only u, T, S can have handles to that key. $D[l_2]$ is a list encrypted under that key. And since, by construction, $\mathrm{M}_T^{\mathrm{PK}}$ and $\mathrm{M}_S^{\mathrm{PK}}$ do not generate a list of the form $D[l_2]$, $\mathrm{M}_u^{\mathrm{PK}}$ must have created it.

B.3 Detailed Proof of Theorem 1

Now we present the proof of Theorem 1:

Proof (of Theorem 1). First we prove the Secrecy Property: Say there was an output $(\mathrm{ok}, \mathrm{PK}, S, SK^{hnd})$ at $\mathrm{KA\_out}_u$!. Examining Algorithm 3 (Fig. 9, 10)

we see that a handle to this key was contained in the set $Session\_KeysS_u$ (see steps 3.59, 3.60). Also by Algorithm 3, one sees that elements in $Session\_KeysS_u$ contain handle to keys which $\mathrm{M}_u^{\mathrm{PK}}$ gets a handle in step 3.46. With the notation of Algorithm 3, any element in $Session\_KeysS_u$ was contained in a list l_2 to which $\mathrm{M}_u^{\mathrm{PK}}$ gets a handle in step 3.45. Here $l_2.hnd_u \neq\downarrow$ since otherwise the algorithm would abort by Convention 1. Algorithm 3 implies that l_2 was encrypted with a symmetric key AK which was part of a list contained in the set $TGTicket_u$ (see steps 3.42, 3.43). Elements stored in $TGTicket_u$ in step 3.37 were received by $\mathrm{M}_u^{\mathrm{PK}}$ as part of a message m. Steps 3.1-3.35 ensure that m was a valid TGS_REP. In particular, Authentication of TGS to client and Secrecy of SK implies that the key SK was generated by $\mathrm{M}_T^{\mathrm{PK}}$ and an adversary cannot get a handle to the key SK.

Now say there was an output $(\mathrm{ok}, \mathrm{PK}, u, x_1^{hnd})$ at $\mathrm{KA\_out}_S!$. By Algorithm 5, the handle to x_1 was contained in a list x to which $\mathrm{M}_S^{\mathrm{PK}}$ got handle in step 5.2. Here $x.hnd_S \neq\downarrow$ since otherwise the algorithm would abort by Convention 1. Steps 5.6, 5.7 ensure that x_1 is really a symmetric key. Since also all other steps of Algorithm 5 must have been executed by $\mathrm{M}_S^{\mathrm{PK}}$ without abort before the output $(\mathrm{ok}, \mathrm{PK}, u, k^{hnd})$, and steps 5.1 - 5. 14 ensure that the server received a valid AP_REQ (including a ST and a corresponding authenticator), TGS Authentication of the TGT implies that the key was generated by $\mathrm{M}_T^{\mathrm{PK}}$ and the authenticator was created by $\mathrm{M}_u^{\mathrm{PK}}$. Key Secrecy implies that only u, T, S can have handles to SK. This proves the Secrecy Property.

Next we prove the Authentication Property: i) Say there was an output (ok, PK, v, x_1^{hnd}) at $\mathrm{KA\_out}_S!$ at a time $t_3 \in \mathbb{N}$. This only happens in step 5.17. By construction, there must have been an input (v, S, i, m^{hnd}) at $\mathrm{out}_S?$ at a past time. By step 5.3, m^{hnd} contained a list x encrypted under the symmetric key $skse_{TS}$. Here $x.hnd_u \neq\downarrow$ otherwise the algorithm would abort by Convention 1. By steps 5.4 - 5.7 one sees that the list contained the handle to a symmetric key x_1 and the name v. By assumption, only $\mathrm{M}_S^{\mathrm{PK}}$ and $\mathrm{M}_T^{\mathrm{PK}}$ have keys to the key $skse_{TS}$ but $\mathrm{M}_S^{\mathrm{PK}}$ does not use this key for encryption (see Algorithm 5). In the notation of Algorithm 4, $\mathrm{M}_T^{\mathrm{PK}}$ uses $skse_{TS}$ in step 4.20 to encrypt a list l, which contains a handle to a symmetric key SK that was freshly generated in step 4.18 and also the user name z_1. The same name was also contained in a list z encrypted under a symmetric key $y_{1.1}$ (see steps 4.12 - 4.16). So if the symmetric key x_1 from above is SK, then $v = z_1$, since the symmetric key SK was freshly generated in step 4.18. The symmetric key $y_{1.1}$ was contained in a list encrypted under the secret key $skse_{KT}$ shared exclusively by K and T, as assumed. By construction, this encryption must have been done by $\mathrm{M}_K^{\mathrm{PK}}$ since $\mathrm{M}_T^{\mathrm{PK}}$ (in Algorithm 4) does not use $skse_{KT}$ for encryption. This implies the symmetric key $y_{1.1}$ was generated by $\mathrm{M}_K^{\mathrm{PK}}$ in step 2.25. And Key Secrecy therefore implies that only $\mathrm{M}_K^{\mathrm{PK}}$ and $\mathrm{M}_u^{\mathrm{PK}}$ for one $u \in \mathcal{H}$ have handles to $y_{1.1}$. By construction, only u uses the key $y_{1.1}$ to encrypt its own name in step 1B.4. So one gets $u = v$. Furthermore, step 1B.4 will only be executed if there was an input $(\mathrm{continue\_prot}\ \mathrm{PK}, T, S, y_{1.1})$ at $\mathrm{KA\_in}_v!$ at time $t_2 < t_3$. On the other hand, there must have been an output in step 3.38 that contained the same symmetric key as in the input, namely the

key $y_{1.1}$. Otherwise there will be an abort in step 1B.2, i.e., the constructions of Algorithm 3 and Algorithm 1B and the definition of $TGTicket$ in step 3.37 imply that there was an output $(\text{ok}, \text{KAS\_exchange PK}, K, T, (.,.,.,y_{1.1}))$ at KA_out_v!. Since the execution of Algorithm 3 did not produce an error, Authentication of KAS to client and Secrecy of AK immediately implies that M_K^{PK} must have generated a AS_REP for v using Algorithm 2. Finally, by construction of Algorithm 2 and by definition of the command verify, one gets that v must have given the input $(\text{new\_prot}, \text{PK}, K, T))$ at KA_in_v! at a time $t_1 < t_2$.

ii) Now say that there was on output $(\text{ok}, \text{PK}, S, SK^{hnd})$ at KA_out_u! at time t_2. By construction of Algorithm 3, this only happens after u received a handle to a message m that was encrypted under a key contained in the set $Session\_KeysS_u$ (see steps 3.59, 3.60). Elements in the set $Session\_KeysS_u$ are stored there in step 3.56 and consist of pairs $(S, x_{2.1}^{hnd})$, where step 3.47, 3.49, 3.50 ensure that $x_{2.1}$ is indeed a symmetric key. Each such key $x_{2.1}$ was contained in a message m which must be a valid TGS_REP. Otherwise Algorithm 3 (part 2) would abort somewhere between step 3.40 to step 3.50. Authentication of TGS to client and Secrecy of SK now implies that each $x_{2.1}$, i.e. in particular also the key SK, must have been generated by M_T^{PK} in step 4.18. Key Secrecy implies that only T, u, S can have handles to SK but M_T^{PK} does not use this key for encryption. Although M_u^{PK} does use this key for encryption in step 3.53, the list it encrypts includes u's name (see step 3.52). On the other hand, the list M_u^{PK} receives a handle to in step 3.60 (after decryption with SK and before the output $(\text{ok}, \text{PK}, S, SK^{hnd})$ at KA_out_u! at time t_2) does not contain u's name (steps 3.63 - 3.65). Therefore, M_S^{PK} must have used the key SK for encryption at a time $t_1 < t_2$. This can only happen in step 5.15 after receiving a valid AP_REQ from v. This encryption must have been sent, so there was no abort in step 5.16, and therefore there must have been an output $(\text{ok}, \text{PK}, v, SK^{hnd})$ at KA_out_S! at some time $t_1 < t_2$.