Breaking and Fixing Public-Key Kerberos

Iliano Cervesato* Aaron D. Jaggard† Andre Scedrov‡
Joe-Kay Tsay** Christopher Walstad††

*Carnegie Mellon University, iliano@cmu.edu
†Tulane University of Louisiana
‡University of Pennsylvania
**University of Pennsylvania
††University of Pennsylvania

Breaking and Fixing Public-Key Kerberos [1]

Iliano Cervesato [a], Aaron D. Jaggard [b], Andre Scedrov [c],
Joe-Kai Tsay [c], Christopher Walstad [c]

[a] *Carnegie Mellon University, Qatar*
[b] *Tulane University*
[c] *University of Pennsylvania*

Abstract

We report on a man-in-the-middle attack on PKINIT, the public key extension of the widely deployed Kerberos 5 authentication protocol. This flaw allows an attacker to impersonate Kerberos administrative principals (KDC) and end-servers to a client, hence breaching the authentication guarantees of Kerberos. It also gives the attacker the keys that the KDC would normally generate to encrypt the service requests of this client, hence defeating confidentiality as well. The discovery of this attack caused the IETF to change the specification of PKINIT and Microsoft to release a security update for some Windows operating systems. We discovered this attack as part of an ongoing formal analysis of the Kerberos protocol suite, and we have formally verified several possible fixes to PKINIT—including the one adopted by the IETF—that prevent our attack as well as other authentication and secrecy properties of Kerberos with PKINIT.

Key words: Computer Security, Authentication Protocols, Kerberos, PKINIT, Man-in-the-middle Attack, Protocol Verification.

Email addresses: `iliano@cmu.edu` (Iliano Cervesato), `adj@math.tulane.edu` (Aaron D. Jaggard), `scedrov@math.upenn.edu` (Andre Scedrov), `jetsay@math.upenn.edu` (Joe-Kai Tsay), `cwalstad@seas.upenn.edu` (Christopher Walstad).

[1] Cervesato was partially supported by the Qatar Foundation under grant number 930107, with early aspects of this work supported by ONR under grants N00014-99-1-0150 and N00014-01-1-0795. Jaggard was partially supported by NSF Grants DMS-0239996 and CNS-0429689 and by ONR Grant N00014-05-1-0818; parts of this work were done while he was a Visiting Scholar in the Mathematics Department at the University of Pennsylvania. Scedrov, Tsay, and Walstad were 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

Preprint submitted to Elsevier *21 May 2007*

1 Introduction

Kerberos [1] is a successful, widely deployed single sign-on protocol that is designed to authenticate clients to multiple networked services, *e.g.*, remote hosts, file servers, or print spoolers. Kerberos 5, the most recent version, is available for all major operating systems: Microsoft has included it in its Windows operating system, it is available for Linux under the name Heimdal, and commercial Unix variants as well as Apple's OS X use code from the MIT implementation of Kerberos 5. Furthermore, it is being used as a building block for higher-level protocols [2]. Introduced in the early 1990s [3], Kerberos 5 continues to evolve as new functionalities are added to the basic protocol. One of these extensions, known as PKINIT, modifies the basic protocol to allow public-key authentication and in the process adds considerable complexity to the protocol. Here we report a protocol-level attack on PKINIT and discuss the constructive process of fixing it. We have verified a few defenses against our attack, including one we suggested, a different one proposed in the IETF Kerberos working group (and included the final PKINIT specification), and a generalization of these two approaches.

A Kerberos session generally starts with a user logging onto a system. This triggers the creation of a client process that will transparently handle all her authentication requests. The initial authentication between the client and the Kerberos administrative principals (altogether known as the KDC, for Key Distribution Center) is traditionally based on a shared key derived from a password chosen by the user. PKINIT is intended to add flexibility, security and administrative convenience by replacing this static shared secret with two pairs of public/private keys, one assigned to the KDC and one belonging to the user. PKINIT is supported by Kerberized versions of Microsoft Windows, typically for use with smartcard authentication, including Windows 2000 Professional and Server, Windows XP, Windows Server 2003 and now Windows Vista [4]; it has also been included in Heimdal since 2002 [5]. The MIT reference implementation is being extended with PKINIT.

The flaw we have uncovered in PKINIT allows an attacker to impersonate the KDC, and therefore all the Kerberized services, to a user, hence defeating authentication of the server to the client. The attacker also obtains all the keys that the KDC would normally generate for the client to encrypt her service requests, hence compromising confidentiality as well. This is a protocol-level attack and was a flaw in the then-current specification, not just a particular implementation. In contrast to recently reported attacks on Kerberos 4 [6], our

---

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.

attack does not use an oracle, but is efficiently mounted in constant time by simply decrypting a message with one key, changing one important value, and re-encrypting it with the victim's public key. The consequences of this attack are quite serious. For example, the attacker could monitor communication between an honest client and a Kerberized network file server. This would allow the attacker to read the files that the client believes are being securely transferred to the file server.

Our attack is possible because the two messages constituting PKINIT were insufficiently bound to each other.[2] More precisely, the reply (the second message of this exchange) can easily be modified so that it appears to correspond to a request (the first message) sent by a client different from the one the reply is generated for. Assumptions required for this attack are that the attacker is a legal user, that he can intercept other clients' requests, and that PKINIT is used in "public-key encryption mode". The alternative "Diffie-Hellman (DH) mode" does not appear vulnerable to this attack; we are in the process of proving its full security.

We discovered this attack as part of an ongoing formal analysis of the Kerberos 5 protocol suite. Our earlier work on Kerberos successfully employed formal methods for the verification of the authentication properties of basic intra-realm Kerberos 5 [7,8] and of cross-realm authentication [8,9]. Although our work is carried out by hand, a variety of automated approaches exist for symbolic proofs [10–17] and have also been applied to deployed protocols (*e.g.*, [18–20]). In a recent collaboration with M. Backes, we have started extending our results from the abstract Dolev-Yao model examined here to the more concrete computational model [21]. Interestingly, the results described in more detail here served as a blueprint for the much more fine-grained proofs in [21]. Furthermore, we have started exploring automated security proofs of Kerberos [22] using the tool CryptoVerif [23,24], which works directly within the computational model.

After discovering the attack on PKINIT, we worked in close collaboration with the IETF Kerberos Working Group, in particular with the authors of the PKINIT specification documents, to correct the problem. Our contribution in this regard has been a formal analysis of a general countermeasure to this attack, as well as the particular instance proposed by the Working Group that has been adopted in the PKINIT specification [25]. Our attack led to an August 2005 Microsoft Security Bulletin and patch [4]. It was also recorded as a CERT advisory [26]. This paper extends our preliminary report [27] with additional authentication properties, full proofs of our results, and a more complete discussion of our techniques, formalization, and the protocol itself.

---

[2] The possibility of an 'identity misbinding' attack was independently hypothesized by Ran Canetti, whom we consulted on some details of the specification

Below, in Section 2 we recall the structure of Kerberos and give a detailed description of PKINIT. In Section 3 we provide an account of the attack we uncovered and outline its consequences. In Section 4 we discuss various approaches to prevent the attack, including the one adopted by the IETF Kerberos working group in response to our work. In Section 5 we review our representation language, MSR, and use it to formalize the fixed version of PKINIT; we give some of our formal results—that the fixed version does prevent our attack and that both the broken and fixed versions have other authentication and secrecy properties—in Section 6. Section 7 provides some concluding remarks.

2 Kerberos 5 and its Public-Key Extension

The Kerberos protocol [1] allows a legitimate user to log on to her terminal once a day (typically) and then transparently access all the networked resources she needs in her organization for the rest of that day. Each time she wants to retrieve a file from a remote server, for example, Kerberos securely handles the required authentication behind the scene, without any user intervention.

We now review how Kerberos provides secure authentication based on a single logon. As we do this, we will be particularly interested in the initial exchange, which happens when the user first logs onto the system. Figure 1 gives an overview of the message flow for the entire Kerberos protocol; Figure 4 below refines this to show some of the message details for the basic protocol. We start this section with a detailed review of the first exchange in the protocol, both with and without PKINIT.

2.1 Kerberos Basics

The client process—usually acting on behalf of a human user—interacts with three other types of principals during the three rounds of Kerberos 5 (with or without PKINIT). The client's goal is to be able to authenticate herself to various application servers (*e.g.*, email, file, and print servers). This is done by first obtaining credentials, called the "ticket-granting ticket" (TGT), from a "Kerberos Authentication Server" (KAS) and then by presenting these credentials to a "Ticket-Granting Server" (TGS) in order to obtain a "service ticket" (ST), which is the credential that the client finally presents to the application servers in order to authenticate herself (see Figure 1). 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 might be valid for a few

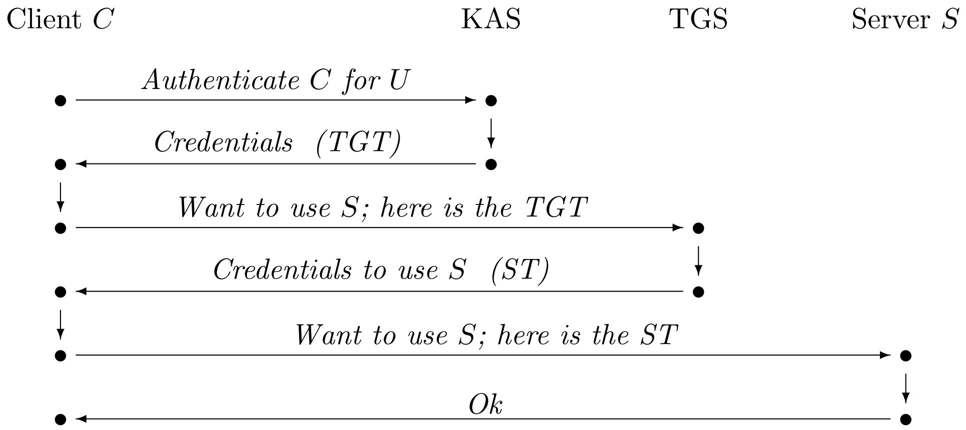
Client C
KAS
TGS
Server S
Authenticate C for U
Credentials (TGT)
Want to use S; here is the TGT
Credentials to use S (ST)
Want to use S; here is the ST
Ok

Fig. 1. An Overview of Kerberos Authentication

minutes (although it, too, may be used repeatedly while it is still valid) and is used for a single application server. The KAS and the TGS are altogether 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. The focus of this work will be the AS exchange, as PKINIT does not alter the remaining parts of Kerberos.

The Traditional Authentication Service Exchange. The abstract structure of the messages in the traditional (non-PKINIT) AS exchange is given in Figure 2. 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. The KAS responds by generating a fresh key AK for use between the client and the TGS. This key is sent back to the client, along with the nonce (n_1) from the request and other data, encrypted under a long-term key k_C shared between C and the KAS; this long-term key is usually derived from the user's password. We write $\{m\}_k$ for the encryption of m with symmetric key k. This is the only time that this long-term key is used in a standard Kerberos run because later exchanges use freshly generated keys. AK is also included in the TGT, sent alongside the message encrypted for the client. The TGT is encrypted under a long-term key k_T shared between the KAS and the TGS named in the request. These encrypted messages are accompanied by the client's 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; the details we omit here do not affect our results and including them would only obscure their exposition. We refer the reader to [1] for the complete specification of

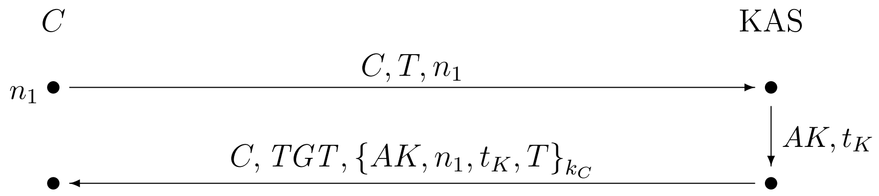

Fig. 2. Message Flow in the Traditional AS Exchange where $TGT = \{AK, C, t_K\}_{k_T}$.

Kerberos 5, and to [7] for a formalization at an intermediate level of detail.

Security Considerations. One weakness of the standard Kerberos protocol is that the key k_C used to encrypt the client's credentials is derived from a password, and passwords are notoriously vulnerable to dictionary attacks [1]. Moreover, since the initial request is entirely plaintext, an active attacker can repeatedly make requests for an honest client's credentials and amass a large quantity of plaintext-ciphertext pairs, the latter component being encrypted with the client's long-term key k_C. While the attacker is unable to use these credentials to authenticate to the system, he is given considerable opportunity to perform an active dictionary attack against the key.

Kerberos can optionally use pre-authentication, a feature that is designed to prevent an attacker from actively requesting and obtaining credentials for an honest user. In brief, pre-authentication works by requiring the client to include a timestamp encrypted with her long-term key (k_C) in the initial request. The authentication server will only return credentials if the decrypted timestamp is sufficiently recent. This method successfully prevents an attacker from actively obtaining ciphertext encrypted with the long-term key; however, it does not prevent passive dictionary attacks, *i.e.*, a passive attacker could eavesdrop on network communications, record credentials as the honest client requests them, and attempt off-line dictionary decryption. Thus, pre-authentication makes it slower for an attacker to perform cryptanalysis against the user's long-term key, but it does not fully prevent the vulnerability. One goal of PKINIT is to eliminate the possibility of this dictionary attack.

2.2 Public-Key Kerberos

PKINIT [25] 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. As we just saw, 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, PKINIT allows network administrators to use an existing public key infrastructure (PKI) rather than expend additional effort to manage users' long-term keys needed for 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. 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 [28]. Other possibilities include keeping these credentials on the user's hard drive, again protected by a passphrase.

The manner in which PKINIT works depends on both the protocol version and the mode invoked. As the PKINIT extension to Kerberos has recently been published as RFC 4556 after a sequence of Internet Drafts [25], we use "PKINIT-n" to refer to the protocol as specified in the n^{th} draft revision and "PKINIT" for the protocol more generally. These various drafts and the RFC can be found at [25]. We discovered the attack described in Section 3 when studying PKINIT-25; our description of the vulnerable protocol is based on PKINIT-26, which does not differ from PKINIT-25 in ways that affect the attack. In response to our work described here, PKINIT-27 included a defense against our attack; we discuss this fix in Section 4. The version of the protocol defined in RFC 4556 does not differ from the parts of PKINIT-27 that we discuss here.

PKINIT can operate in two modes. In *public-key encryption mode*, the key pairs (pk_C, sk_C) and (pk_K, sk_K) are used for both signature and encryption. The latter is designed to (indirectly) protect the confidentiality of AK, while the former ensures its integrity. In *Diffie-Hellman (DH) mode*, the key pairs are used to provide digital signature support for an authenticated Diffie-Hellman key agreement which is used to protect the fresh key AK shared between the client and KAS. A variant of this mode allows the reuse of previously generated shared secrets. We will not discuss the DH mode in detail as our preliminary investigation did not reveal any flaw in it; we are currently working

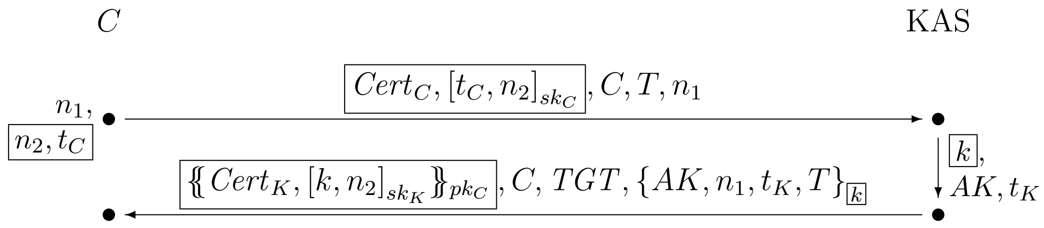
C
KAS
$\boxed{Cert_C, [t_C, n_2]_{sk_C}}, C, T, n_1$
n_1,
$\boxed{n_2, t_C}$
$\boxed{k}$,
AK, t_K
$\boxed{\{\!\!\{Cert_K, [k, n_2]_{sk_K}\}\!\!\}_{pk_C}}, C, TGT, \{AK, n_1, t_K, T\}_{\boxed{k}}$

Fig. 3. Message Flow in $\boxed{\text{PKINIT-26}}$, where $TGT = \{AK, C, t_K\}_{k_T}$.

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 [29], developed by CableLabs, a cable television research consortium.

Figure 3 illustrates the AS exchange in public-key encryption mode as of PKINIT-26. The differences with respect to the traditional AS exchange (see Figure 2) have been highlighted using $\boxed{\text{boxes}}$. In discussing this and other descriptions of the protocol, we write $[m]_{sk}$ for the digital signature of message m with secret key sk. (PKINIT realizes digital signatures by concatenating the message and a keyed hash for it, occasionally with other data in between.) In our analysis of PKINIT in Section 6, we make the standard assumption that digital signatures are unforgeable [30]. The encryption of m with public key pk is denoted $\{\!\!\{m\}\!\!\}_{pk}$. As before, we write $\{m\}_k$ for the encryption of m with symmetric key k.

The first line of Figure 3 describes the relevant parts of the request that a client C sends to a KAS K using PKINIT-26. The last part of the message—C, T, n_1—is exactly as in basic Kerberos 5, containing the client's name, the name of the TGS for which she wants a TGT, and a nonce. The boxed parts added by PKINIT include 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 to the left of this line indicate that these are generated by C specifically for this request, with the box indicating data not included in our abstract formalization of basic Kerberos 5 [7,8].) This effectively implements a form of pre-authentication.

The second line in Figure 3 shows our formalization of K's response, which is more complex than in basic Kerberos. The last part of the message—C, TGT, $\{AK, n_1, t_K, T\}_{\boxed{k}}$—is very similar to K's reply in basic Kerberos; the difference ($\boxed{\text{boxed}}$) is that the symmetric key k protecting AK is now freshly generated by K and not a long-term shared key. The ticket-granting ticket TGT and the message encrypted under k are as in traditional Kerberos. Because k is freshly generated for the reply, it must be communicated to C before she can learn AK. PKINIT does this by adding the message $\{\!\!\{Cert_K, [k, n_2]_{sk_K}\}\!\!\}_{pk_C}$.

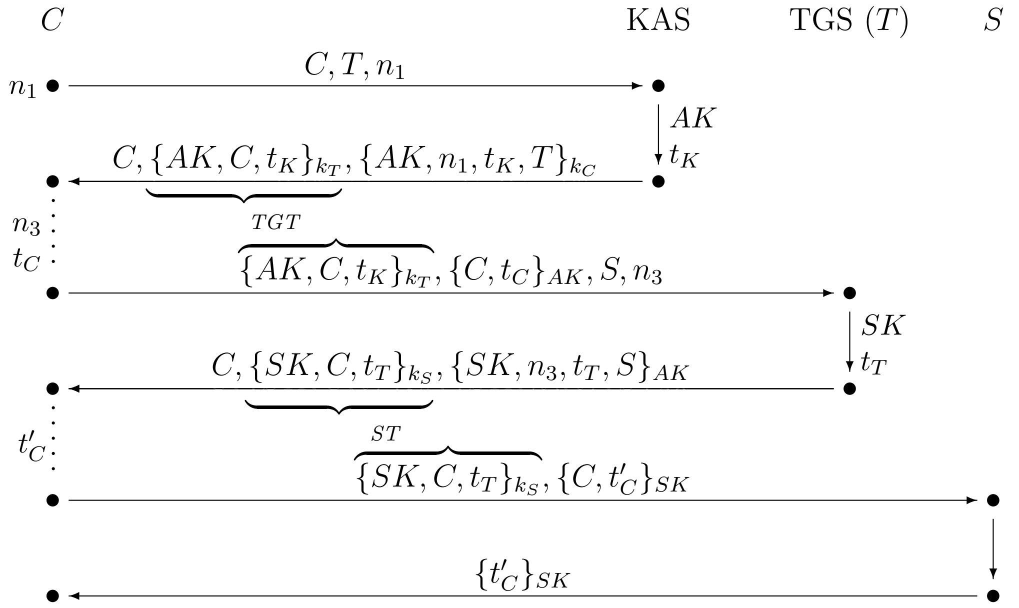
C
KAS
TGS (T)
S
n_1
C, T, n_1
AK
t_K
$C, \{AK, C, t_K\}_{k_T}, \{AK, n_1, t_K, T\}_{k_C}$
TGT
n_3
t_C
$\{AK, C, t_K\}_{k_T}, \{C, t_C\}_{AK}, S, n_3$
SK
t_T
$C, \{SK, C, t_T\}_{k_S}, \{SK, n_3, t_T, S\}_{AK}$
ST
t'_C
$\{SK, C, t_T\}_{k_S}, \{C, t'_C\}_{SK}$
$\{t'_C\}_{SK}$

Fig. 4. Message flow in basic Kerberos

This contains K's certificates and his signature, using his secret key sk_K, over k and the nonce n_2 from C's request; all of this is encrypted under C's public key pk_C.

This abstract description leaves out a number of fields which are of no significance with respect to the reported attack or its fix. We invite the interested reader to consult the specifications [25]. Also, recall that PKINIT leaves the subsequent exchanges of Kerberos unchanged.

2.3 Message Flow in Later Exchanges

For the sake of completeness, we give a brief overview of the message structure in the remaining rounds of Kerberos; Section 3.3 discusses how the attack on PKINIT can be propagated through these later rounds. Figure 4 updates Figure 1 to show the details of the messages in basic Kerberos. PKINIT modifies the first two of these messages as illustrated in Figure 3.

The AS exchange (either traditional or with PKINIT) provides the client with an authentication key AK and a ticket granting ticket $TGT = \{AK, C, t_K\}_{k_T}$.

In the TG exchange, C then requests an ST from T after generating a new nonce n_3 and timestamp; her request includes the TGT (which she cannot read but simply forwards from K), an authenticator $\{C, t_C\}_{AK}$ containing her name and timestamp and encrypted with AK, the name of the server S for which she wants an ST, and the new nonce. T's response has the same structure as K's, but now with an ST in place of the TGT and S taking the role of T in the rest of the message.

Finally, in the CS exchange, C authenticates herself to S by sending the ST and an authenticator $\{C, t'_C\}_{SK}$ containing her name and timestamp, and encrypted under the fresh key SK. S may authenticate himself back to C by encrypting C's timestamp (but not C's name, so that the result differs from the authenticator) with SK and returning this message to C.

3 The Attack

In this section, we report on a dangerous attack against PKINIT in public-key encryption mode. We discovered this attack as we were interpreting the specification documents of this protocol [25] in preparation for its formalization in MSR [9,31,32], the specification language for our analysis. We start with a detailed description of the attacker's actions in the AS exchange, the key to the attack. We then review the conditions required for the attack and close this section with a discussion of how the attacker may propagate the effects of her AS exchange actions throughout the rest of a protocol run.

3.1 Message Flow

Figure 5 shows the AS exchange message flow in the attack. The client C sends a request to the KAS K which is intercepted by the attacker I, who constructs his own request message using the parameters from C's message. All data signed by C are sent unencrypted—indeed $[msg]_{sk}$ can be understood as an abbreviation for the plaintext msg together with a keyed hash of the message—so that I may generate his own signatures over data from C's request. The result is a well-formed request message from I, although constructed using some data originating with C. I's changes to the request message are $\boxed{\text{boxed}}$ above the top-right arrow of Figure 5. (We have omitted an unkeyed checksum taken over unencrypted data from these messages; I can regenerate this as needed to produce a valid request.)

I forwards the fabricated request to the KAS K, who views it as a valid request for credentials if I is himself a legitimate client; there is nothing to indicate that some of the data originated with C. K responds with a reply containing credentials for I (the bottom-right arrow in Figure 5). The TGT has the form $\{AK, I, t_K\}_{k_T}$; note that, because it is encrypted with the key k_T shared between K and the TGS T, it is opaque to C (and I). Another part of the reply is encrypted using the public key of the client for whom the credentials are generated, in this case I. This allows the attacker to decrypt

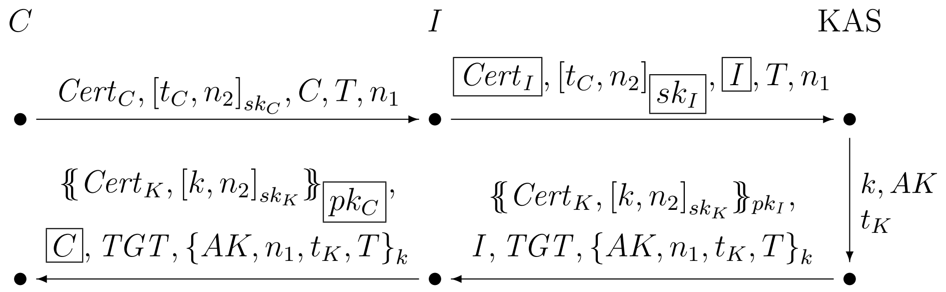
C
I
KAS
$Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1$
$\boxed{Cert_I}, [t_C, n_2]_{\boxed{sk_I}}, \boxed{I}, T, n_1$
k, AK
t_K
$\{\!\{Cert_K, [k, n_2]_{sk_K}\}\!\}_{\boxed{pk_C}},$
$\boxed{C}, TGT, \{AK, n_1, t_K, T\}_k$
$\{\!\{Cert_K, [k, n_2]_{sk_K}\}\!\}_{pk_I},$
$I, TGT, \{AK, n_1, t_K, T\}_k$

Fig. 5. Message Flow in the Man-In-The-Middle Attack on PKINIT-26, where $TGT = \{AK, I, t_K\}_{k_T}$.

this part of the message using his private key, learn the key k, and use this to learn the key AK. An honest client would only use this information to send a request message to the TGS T. Instead, I uses C's public key to re-encrypt the data he decrypted using his private key (having learned pk_C, if necessary, from $Cert_C$ in the original request), replaces his name with C's, and forwards the result to C. To C this message appears to be a valid reply from K generated in response to C's initial request (recall that C cannot read I's name inside the TGT).

At this point, C believes she has authenticated herself to the KAS and that the credentials she has obtained—the key AK and the accompanying TGT—were generated for her. However, the KAS has completed the PKINIT exchange with I and has generated AK and the TGT for I. The attacker knows the key AK (as well as k, which is not used other than to encrypt AK) and can therefore decrypt any message that C would protect with it.

Protocol-level attacks in the same vein of the vulnerability we uncovered have been reported in the literature for other protocols. In 1992, Diffie, van Oorschot, and Wiener noted that a signature-based variant of the Station-to-Station protocol [33] could be defeated by a man-in-the-middle (MITM) attack which bears similarities to what we observed in the first half of our vulnerability; in 2003 Canetti and Krawczyk [34] observed that the "basic authenticated Diffie-Hellman" mode of the Internet Key Exchange protocol (IKE) [35] had this very same vulnerability. In 1996, Lowe [36] found an attack on the Needham-Schroeder public key protocol [37] that manipulates public key encryption essentially in the same way as what happens in the second half of our attack. Because it alters both signatures and asymmetric encryptions, our attack against PKINIT stems from both [36] and [33]. In 1995, Clark and Jacob [38] discovered a similar flaw on Hwang and Chen's corrected SPLICE/AS protocol [39].

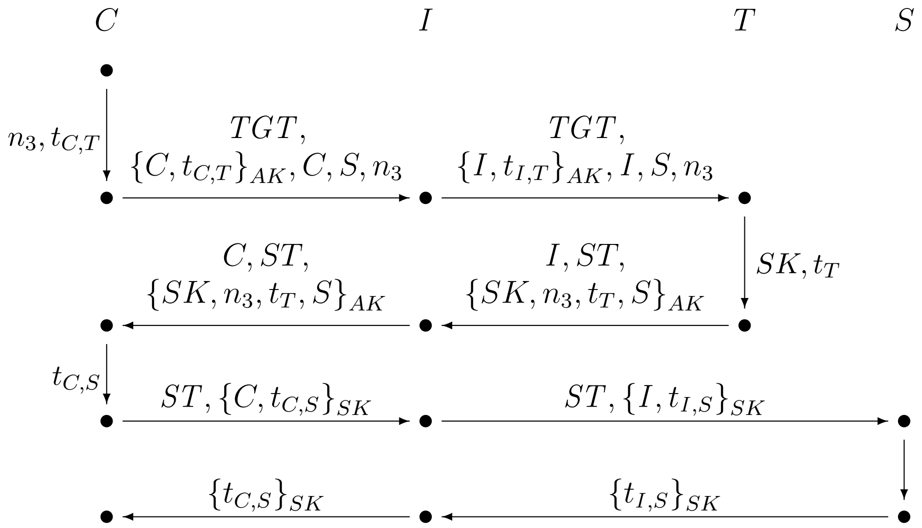
C
I
T
S
$n_3, t_{C,T}$
$TGT,$
$\{C, t_{C,T}\}_{AK}, C, S, n_3$
$TGT,$
$\{I, t_{I,T}\}_{AK}, I, S, n_3$
$C, ST,$
$\{SK, n_3, t_T, S\}_{AK}$
$I, ST,$
$\{SK, n_3, t_T, S\}_{AK}$
SK, t_T
$t_{C,S}$
$ST, \{C, t_{C,S}\}_{SK}$
$ST, \{I, t_{I,S}\}_{SK}$
$\{t_{C,S}\}_{SK}$
$\{t_{I,S}\}_{SK}$

Fig. 6. Message flow in the man-in-the-middle attack on PKINIT-26, after the messages in Figure 5, when the attacker forwards and observes traffic; here $TGT = \{AK, I, t_K\}_{k_T}$ and $ST = \{SK, I, t_T\}_{k_S}$.

3.2 Assumptions

In order for this attack to work, the attacker must be a legal Kerberos client so that the KAS will grant him credentials. In particular, he must possess a public/secret key pair (pk_I, sk_I) and valid certificates $Cert_I$ trusted by the KAS. The attacker must also be able to intercept messages, which is a standard assumption. Finally, PKINIT must be used in public-key encryption mode, which is commonly done as the alternative DH mode does not appear to be readily available, except for domain specific systems [28,29].

3.3 Effects of the Attack

Attacker Observes Traffic. Once the attacker learns AK in the AS exchange, he may either mediate C's interactions with the various servers (essentially logging in as I while leaking data to C so she believes she has logged in) while observing this traffic or simply impersonate the servers in the later exchanges. In the first variant, which is shown in Figure 6, once C has AK and a TGT, she would normally contact the TGS to get an ST for some application server S. This request contains an *authenticator* of the form $\{C, t_{C,T}\}_{AK}$ (*i.e.*, C's name and a timestamp, encrypted with AK). Because I knows AK, he may intercept the request and replace the authenticator with one that refers to himself: $\{I, t_{I,T}\}_{AK}$. The reply from the TGS contains a freshly generated key SK; this is encrypted under AK, for C to read and thus accessible to I, and also included in an ST that is opaque to all but the TGS and application server. I may intercept this message and learn SK, replace an instance of his

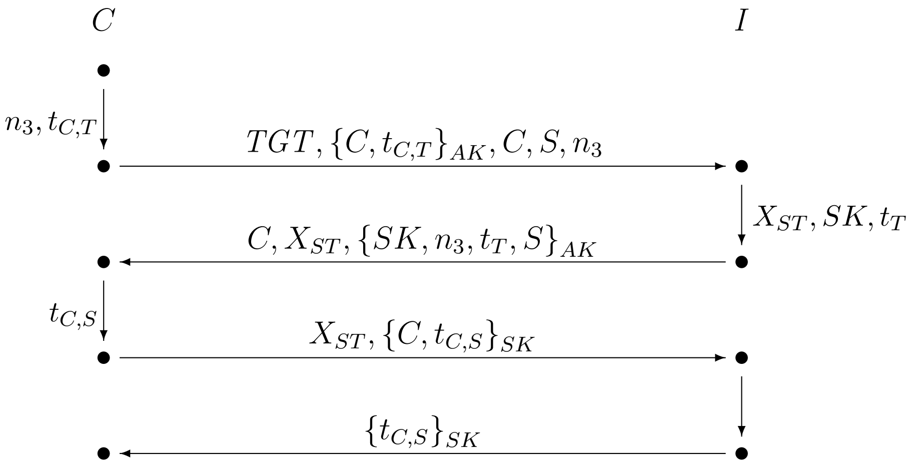
C
I
$n_3, t_{C,T}$
$TGT, \{C, t_{C,T}\}_{AK}, C, S, n_3$
X_{ST}, SK, t_T
$C, X_{ST}, \{SK, n_3, t_T, S\}_{AK}$
$t_{C,S}$
$X_{ST}, \{C, t_{C,S}\}_{SK}$
$\{t_{C,S}\}_{SK}$

Fig. 7. Message Flow in the Man-In-The-Middle Attack on PKINIT-26, after the messages in Figure 5, when the attacker impersonates the TGS and end server; here $TGT = \{AK, I, t_K\}_{k_T}$ while X_{ST} is a garbage message.

name with C's name, and forward the result to C. As I knows SK, he can carry out a similar MITM attack on the CS exchange, replacing the authenticator $\{C, t_{C,S}\}_{SK}$ with the authenticator $\{I, t_{I,S}\}_{SK}$ and then replacing the server's reply $\{t_{I,S}\}_{SK}$ with the reply $\{t_{C,S}\}_{SK}$ that the client is expecting. This exchange ostensibly authenticates C to the application server; however, because the service ticket names I, this server would believe that he is interacting with I, not C.

Attacker Impersonates Servers. Alternatively, the attacker may intercept C's requests in the TG and CS exchanges and impersonate the involved servers rather than forwarding altered messages to them; the message flow for this version of the attack is shown in Figure 7. In the TG exchange, I will ignore the TGT and only decrypt the portion of the request encrypted under AK (which he learned during the initial exchange). The attacker will then generate a bogus service ticket X_{ST}, which the client expects to be opaque, and a fresh key SK encrypted (along with other data n_3, t_T, S) under AK, and send these to C in what appears to be a properly formatted reply from the TGS. In the CS exchange the attacker may again intercept the client's request; in this case, no new keys need to be generated, and the attacker only needs to return the client's timestamp encrypted under SK—which I himself generated in the previous exchange—for C to believe that she has completed this exchange with the application server S. Note that the attacker may take the first approach—mediating the exchange between C and a TGS—in the TG exchange and then the second—impersonating the application server—in the CS exchange. The reverse is not possible because I cannot forge a valid ST for S when impersonating T.

Regardless of which approach the attacker uses to propagate the attack throughout the protocol run, C finishes the CS exchange believing that she has interacted with a server S and that T has generated a fresh key SK known only to C and S. Instead, I knows SK in addition to, or instead of, S (depending on how I propagated the attack). Thus I may learn any data that C attempts to send to S; depending on the type of server involved, such data could be quite sensitive. Note that this attack does not allow I to impersonate C to a TGS or an application server because all involved tickets name I; Section 6.4 discusses a related authentication property. This also means that if C is in communication with an actual server (T or S), that server will view the client as I, not C.

4 Preventing the Attack

The attack outlined in the previous section was possible because the two messages constituting the then-current version of PKINIT were insufficiently bound to each other. More precisely, the attack shows that, although a client can link a received response to a previous request (thanks to the nonces n_1 and n_2, and to the timestamp t_C), she cannot be sure that the KAS generated the key AK and the ticket granting ticket TGT appearing in this response *for her*. Indeed, the only evidence of the principal for whom the KAS generated these credentials appears inside the TGT, which is opaque to her. This suggests one approach to making PKINIT immune to this attack, namely to require the KAS to include the identity of this principal in a component of the response that is integrity-protected and that the client can verify. An obvious mechanism is the submessage signed by the KAS in the reply.

Following a methodology we successfully applied in previous work on Kerberos [7,9], we have constructed a formal model of both PKINIT-26 and various possible fixes to this protocol (including the one adopted in PKINIT-27). Details can be found in Section 6. Property 4.1 informally states that PKINIT-27 and subsequent versions satisfy a security property that we see violated in PKINIT-26, demonstrating that this fix does indeed defend against our attack.

Property 4.1 *In PKINIT-27 (and subsequent versions), whenever a client C processes an AS reply containing server-generated public-key credentials, the KAS previously produced such credentials for C.*

This property informally expresses the contents of Corollary 6.4, which we prove in Section 6.

As we worked on our formal analysis, we solicited feedback from the IETF Kerberos Working Group, and in particular the authors of the PKINIT specifications, about possible fixes we were considering. We also analyzed the fix,

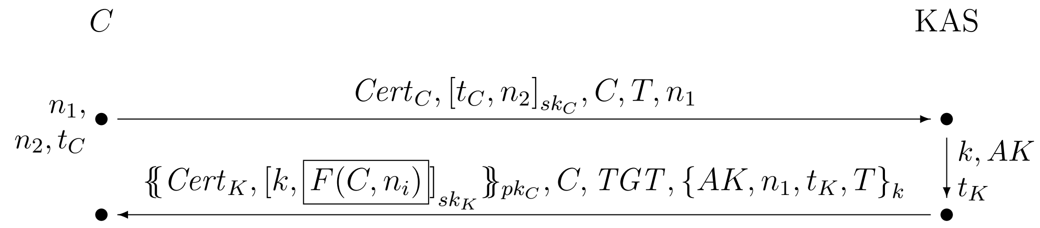
C
KAS
$n_1,$
n_2, t_C
$Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1$
k, AK
t_K
$\{\!\{Cert_K, [k, \boxed{F(C, n_i)}]_{sk_K}\}\!\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k$

Fig. 8. Abstract fix of PKINIT

proposed by the Working Group, that was included in PKINIT-27 and subsequent revisions of this specification [25].

4.1 Abstract Fix

Having traced the origin of the discovered attack to the fact that the client cannot verify that the received credentials (the TGT and the key AK) were generated for her, the problem can be fixed by having the KAS include C's name in the reply in such a way that it cannot be modified *en route* and that C can check it. Following well-established recommendations [40], we initially proposed a simple and minimally intrusive approach to doing so: including C's name in the portion of the reply signed by the KAS (in PKINIT-26, this is $[k, n_2]_{sk_K}$). We then generalized it by observing that the KAS can sign k and any message fragment $F(C, n_i)$ that is suitably built from C's name and at least one of the nonces n_1 and n_2 from C's request for credentials. With this abstract fix in place, the PKINIT exchange in public-key encryption mode is depicted in Figure 8, where we have used a box to highlight the modification with respect to PKINIT-26. Here F represents any construction that injectively involves C and either n_1 or n_2—*i.e.*,$F(C, n_i) = F(C', n'_i)$ implies $C = C'$ and $n_i = n'_i$—and is verifiable by the client. Integrity protection is guaranteed by the fact that it appears inside a component signed by the KAS, and therefore is non-malleable by the attacker (assuming that the KAS's signature keys are secure). Intuitively, this defends against the attack since the client C can now verify that the KAS generated the received credentials for her and not for another principal (such as I in our attack). Indeed, an honest KAS will produce the signature ($[k, F(C, n_i)]_{sk_K}$) only in response to a request from C. The presence of the nonces n_1 or n_2 uniquely identifies which of the (possibly several) requests from C to which this reply corresponds. Note that the fact that we do not need F to mention both n_1 and n_2 entails that the nonce n_2 is superfluous as far as authentication is concerned. We will formally prove that this variant defends against the attack in Section 6.

A simple instance of this general schema consists in taking $F(C, n_i)$ to be (C, n_2), yielding the signed data $[k, C, n_2]_{sk_K}$, which corresponds to simply

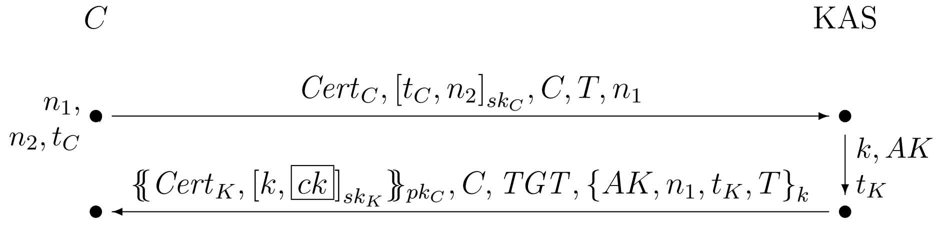
C
KAS
$n_1,$
n_2, t_C
$Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1$
k, AK
t_K
$\{\!\{Cert_K, [k, \boxed{ck}]_{sk_K}\}\!\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k$

Fig. 9. Fix of PKINIT adopted in version 27

including C's name within the signed portion of the PKINIT-26 reply. This version is very similar to the initial target of our formal verification. We showed that indeed it defeats the reported attack and satisfied the formal authentication property violated in PKINIT-26. Only later did we generalize the proof to refer to the abstract construction F. Its correctness will follow as a simple corollary of the validity of our general schema, as we will show in Section 6.

4.2 Solution Adopted in PKINIT-27

When we discussed our initial fix with the authors of the PKINIT document, we received the request to apply our methodology to verify a different solution: rather than simply including C's name in the signed portion of the reply, replace the nonce n_2 there with a keyed hash ("checksum" in Kerberos terminology) taken over the client's entire request. We did so and showed that this approach also defeats our attack. It is on the basis of this finding that we distilled the general fix discussed above, of which both solutions are instances.

The IETF Kerberos Working Group later decided to include the checksum-based approach in PKINIT-27 and its subsequent revisions [25]. The message flow of this version of PKINIT is displayed in Figure 9. Here, ck is a checksum of the client's request keyed with the key k, *i.e.*, ck has the form $H_k(Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1)$ where H is a preimage-resistant MAC function. This means that it is computationally infeasible for the attacker to find a message whose checksum matches that of a given message. Following the specifications in [41], which discusses cryptographic algorithms for use in the Kerberos protocol suite, current candidates for H include hmac-sha1-96-aes128. New strong keyed checksums can be used for ck as they are developed.

5 Formalizing PKINIT in MSR

We have formalized PKINIT in the language MSR [9,31,32] and used this specification to verify the correctness of the proposed fixes. MSR is a flexible framework for specifying complex cryptographic protocols, possibly structured as a collection of coordinated subprotocols. It uses strongly-typed multiset rewriting rules over first-order atomic formulas to express protocol actions and relies on a form of existential quantification to symbolically model the generation of fresh data (*e.g.*, nonces or short-term keys).

5.1 Terms and Types

MSR represents network messages and their components as first-order terms. Thus the TGT $\{AK, C, t_K\}_{k_T}$ sent from K to C is modeled as the term obtained by applying the binary encryption symbol $\{\_\}_\_$ to the constant k_T and the subterm (AK, C, t_K). This subterm is built using atomic terms and two applications of the binary concatenation symbol ("$\_,\_$"). For simplicity, we retain the semi-formal message syntax used earlier. Terms are classified by types, which describe their intended meaning and restrict the set of terms that can be legally constructed. For example, $\{\_\}_\_$ accepts a key (type key) and a message (type msg), producing a term of type msg; using a nonce as the key yields an ill-formed term. Nonces, principal names, *etc.*, often appear within messages; MSR uses the subsort relation to facilitate this. For example, defining nonce to be a subsort of msg (written $\mathsf{nonce} <: \mathsf{msg}$) allows nonces to be treated as messages. Both term constructors and types are definable. This allows us to formalize the specialized principals of Kerberos 5 as subsorts of the generic $\mathsf{principal}$ type: we introduce types client, KAS, TGS and server, with the obvious meanings.

MSR supports more structured type definitions [31]. Dependent types allow capturing the binding between a key and the principals for whom it was created. For example, the fact that a short-term key k is intended to be shared between a particular client C and server S is expressed by declaring it to be of type $\mathsf{shK}\ C\ S$. Because k is a key, $\mathsf{shK}\ C\ S$ is a subsort of key (for all C and S), and since k is short term this type is also a subsort of msg as k needs to be transmitted in a message. We similarly model the long-term keys that a principal A shares with the KAS as objects of type $\mathsf{dbK}\ A$, which is again a subsort of key but *not* of msg; these keys are not intended to be sent over the network, and this typing prohibits this. Dependent types give us elegant means to describe the public-key machinery. If (pk, sk) is the public/secret key pair of principal A, we simply declare pk of type $\mathsf{pubK}\ A$ and sk of type $\mathsf{secK}\ pk$. Secret keys, like the long-term keys, are not intended to be sent over the messages; thus the type $\mathsf{secK}\ pk$ is a subsort of key but not of msg. Although we do not explicitly include public keys in messages here, $\mathsf{pubK}\ A$ is a subsort of msg. The constructors for encryption and digital signature are written $\{\!\{m\}\!\}_{pk}$ and $[m]_{sk}$, respectively, as in the text so far.

$$
\begin{array}{l}
\forall K : \mathsf{KAS} \\
\left[\begin{array}{l}
\qquad \forall C : \mathsf{client} \;\; \forall T : \mathsf{TGS} \;\; \forall n_1, n_2 : \mathsf{nonce} \;\; \forall sk : \mathsf{someSecK} \\
\qquad \forall Cert_C, Cert_K : \mathsf{CertList} \;\; \forall k_T : \mathsf{dbK}\ T \;\; \forall t_C, t_K : \mathsf{time} \\
\qquad \forall pk_C : \mathsf{pubK}\ C \;\; \forall pk_K : \mathsf{pubK}\ K \;\; \forall sk_K : \mathsf{secK}\ pk_K \\[1em]
\begin{array}{lcl}
\mathsf{N}(Cert_C, [t_C, n_2]_{sk}, & & \exists AK : \mathsf{shK}\ C\ T,\ \exists k : \mathsf{shK}\ C\ K \\
 & \overset{\iota_{2.1}}{\Longrightarrow} & \quad \mathsf{N}(\{\!\{Cert_K, [k, \boxed{F(C, n_i))}]_{sk_K}\}\!\}_{pk_C}, \\
\quad C, T, n_1) & & \qquad C, \{AK, C, t_K\}_{k_T}, \{AK, n_1, t_K, T\}_{k})
\end{array} \\[1em]
\text{IF } \ \mathit{VerifySig}([t_C, n_2]_{sk}; (t_C, n_2); C, Cert_C),\ Valid_K(C, T, n_1),\ Clock_K(t_K)
\end{array}\right]
\end{array}
$$

Fig. 10. KAS's Role in the abstract fix version of the PKINIT AS Exchange

Other types used in the formalization of PKINIT include time for timestamps, CertList for lists of digital certificates, and someSecK as an auxiliary type for working with digital signatures. The use of someSecK allows us, *e.g.*, to model a signed message without declaring the public key or its owner that correspond to the signing key; in order to verify the signature we use the predicate *VerifySig* together with a list of certificates (as in Figure 10) instead of a specific public key. We also use the constructor $H_k(m)$ to model the checksum (keyed hash) of message m keyed with symmetric key k.

5.2 States, Rules, and the Formalization of PKINIT

The *state* of a protocol execution is determined by the network messages in transit, the local knowledge of each principal, and other similar data. MSR formalizes individual bits of information in a state by means of *facts* consisting of *predicate name* and one or more terms. For example, the network fact $\mathsf{N}(\{AK, C, t_K\}_{k_T})$ indicates that the term $\{AK, C, t_K\}_{k_T}$, a TGT, is present on the network, and $I(\{AK, C, t_K\}_{k_T})$ that it is known by the attacker.

A protocol consists of actions that transform the state. In MSR, this is modeled by the notion of *rule*: a description of the facts that an action removes from the current state and the facts it replaces them with to produce the next state. For example, Figure 10 describes the actions of the KAS in the abstract fix of PKINIT (see Section 4) where $F(C, n_i)$ stands for a construction that contains C and either n_1 or n_2 (or both). Ignoring for the moment the leading $\forall K$: KAS and the outermost brackets leaves us with a single MSR rule—labeled $\iota_{2.1}$ above the arrow—that we will use to illustrate characteristics of MSR rules in general.

Rules are parametric, as evidenced by the leading string of typed universal

$\forall C : \mathsf{client}$

$$\left[\begin{array}{lcl}
\multicolumn{3}{l}{\forall T : \mathsf{TGS} \quad \forall sk_K : \mathsf{secK}\ C \quad \forall t_C : \mathsf{time} \quad \forall Cert_C : \mathsf{CertList}} \\
 & & \exists n_1, n_2 : \mathsf{nonce} \\
Clock_C(t_C) & \overset{\iota_{1.1}}{\Longrightarrow} & \mathsf{N}(t_C, n_2, Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1), \\
 & & MemASE_C(Cert_C, t_C, T, n_1, n_2, sk_C) \\
\\
\multicolumn{3}{l}{\forall K : \mathsf{KAS} \quad \forall T : \mathsf{TGS} \quad \forall pk_C : \mathsf{pubK}\ C \quad \forall k : \mathsf{shK}\ C\,K \quad \forall n_1, n_2 : \mathsf{nonce}} \\
\multicolumn{3}{l}{\forall X : \mathsf{msg} \quad \forall Cert_K : \mathsf{CertList} \quad \forall AK : \mathsf{shK}\ C\,T \quad \forall t_K : \mathsf{time} \quad \forall sk : \mathsf{someSecK}} \\
\mathsf{N}(\{\!\!\{Cert_K, [k, F(C, n_i)]_{sk}\}\!\!\}_{pk_C}, & & \\
\quad C, X, \{AK, n_1, t_K, T\}_k) & \overset{\iota_{1.2}}{\Longrightarrow} & Auth_C(X, T, AK) \\
MemASE_C(Cert_C, t_C, T, n_1, n_2, sk_C) & & \\
\\
\multicolumn{3}{l}{\text{IF} \quad VerifySig([k, F(C, n_i)]_{sk}; (k, F(C, n_i)); K, Cert_K)}
\end{array}\right]$$

Fig. 11. Client's Role in the abstract fix version of the PKINIT AS Exchange

quantifiers: actual values need to be supplied before applying the rule. The middle portion ($\cdots \Longrightarrow \cdots$) describes the transformation performed by the rule: it replaces states containing a fact of the form $\mathsf{N}(Cert_C, [t_C, n_2]_{skC}, C, T, n_1)$ with states that contain the fact on its right-hand side but which are otherwise identical. The existential marker "$\exists AK : \mathsf{shK}\ C\,T$" requires AK to be replaced with a newly generated symbol of type $\mathsf{shK}\ C\,T$, and similarly for "$\exists k : \mathsf{shK}\ C\,K$"; this is how freshness requirements are modeled in MSR. The last line, starting with the keyword IF, further constrains the applicability of the rule by requiring that certain predicates be present (differently from the left-hand side, they are not removed as a result of applying the rule). Here, we use the predicates *VerifySig* to verify that a digital signature is valid given a list of credentials ($VerifySig(s; m; P, Certs)$ holds if s is the signature, relative to certificates $Certs$, by principal P over the message m). Additionally, we use $Valid_K$ to capture the local policy of K in issuing tickets, and $Clock_K$ to model the local clock of K. While the entities following 'IF' are logically facts, in practice they are often handled procedurally, outside of MSR.

Rule $\iota_{2.1}$ completely describes the behavior of the KAS; in general, multiple rules may be needed, as when modeling the actions of the client in the AS exchange. Coordinated rules describing the behavior of a principal are collected in a *role*. A role is just a sequence of rules, parameterized by the principal executing them (their *owner*)—the "$\forall K : \mathsf{KAS}$" above the brackets in Figure 10. The two-rule role describing the client's actions of the abstract fix version of PKINIT is displayed in Figure 11. Formalizations of the TG and CS exchanges can be found in Appendix A, with detailed explanations in [7,9,42].

Going into more detail, the execution semantics of MSR operates by transforming not states (as introduced in the previous section) but *configurations* of the form $\langle S\rangle_\Sigma^R$, where S is a state, the *signature* Σ contains the symbols being used (with their type), and the *active role set* $R = (\rho_1^{a_1}, \ldots, \rho_n^{a_n})$ records the remaining actions of the currently present roles (ρ_i) together with the principals executing them (a_i).

Basic execution steps are expressed by judgments of the form $\mathcal{P} \rhd C \longrightarrow C'$ where $\mathcal{P}$ is the protocol specification, and C and C' are consecutive configurations. These judgments are defined by the following two rules (which are somewhat simplified from [43]):

$$\frac{}{(\mathcal{P}, \rho) \rhd \langle S\rangle_\Sigma^R \longrightarrow \langle S\rangle_\Sigma^{R,\rho^a}}\ \textbf{inst}$$

$$\frac{}{\mathcal{P} \rhd \langle S, [\theta]\mathit{lhs}\rangle_\Sigma^{R,((\mathit{lhs}\rightarrow\exists\vec{x}.\mathit{rhs}),\rho)^a} \longrightarrow \langle S, [\theta, \vec{c}/\vec{x}]\mathit{rhs}\rangle_{\Sigma,\vec{c}}^{R,\rho^a}}\ \textbf{rw}$$

The rule **inst** prepares a role for execution by inserting it in the active role set: it associates the role with the principal a that will be executing it. The same role can be loaded arbitrarily many times by any principal (subject to some typing limitations), which provides support for the concurrent execution of multiple sessions and therefore also for multi-session attacks. The inference **rw** describes the application of a state transforming rule $r = \mathit{lhs} \Rightarrow \mathit{rhs}$ introduced in the previous section: if an instance $[\theta]\mathit{lhs}$ of its left-hand side appears in the state, it is replaced by the corresponding instantiation of the right-hand side of r after instantiating the existential variables $\vec{x}$ with new constants $\vec{c}$. The rule r is then removed from the active role set of the configuration, and $\vec{c}$ is added to the signature.

In order to present rule application in a compact yet precise way, the notion of an *abstract execution step* is used which denotes a quadruple $C \xrightarrow{r,\iota} C'$. Here, C and C' are consecutive configurations, r identifies the rule from $\mathcal{P}$ being executed, and ι stands for the overall substitution $[\theta, \vec{c}/\vec{x}]$ above. We say that r is *applicable* in C if there exist a substitution ι and a configuration C' such that $C \xrightarrow{r,\iota} C'$ is defined.

A *trace* $\mathcal{T}$ is then a sequence of the form

$$C_0 \overset{r_1,\iota_1}{\longrightarrow} C_1 \overset{r_2,\iota_2}{\longrightarrow} \cdots \overset{r_n,\iota_n}{\longrightarrow} C_{n+1}$$

where $C_0 = \langle S_0\rangle_{\Sigma_0}^{R_0}$ is called the *initial configuration* of $\mathcal{T}$. In the context of Kerberos, the state component S_0 of the initial configuration contains only the predicates used in constraints (*e.g.*, *VerifySig* in Figure 11), and the intruder's knowledge (see next section); in particular, no network message or memory predicate is contained in S_0. The initial signature Σ_0 within C_0 contains the names of all principals together with their types (client, server, KAS, *etc.*), and their keys, *etc.* The initial active role set R_0 within C_0 is empty. Note that we only need to consider finite traces, because execution (in any networked computer system) proceeds by discrete steps and started at a specific moment in time. Likewise, a generic trace will contain only a finite, although *a priori* unbounded, number of applications of inference **inst** for the same role ρ.

5.4 Intruder Model

The intruder model in our analysis of public-key Kerberos is a variant of the classic Dolev-Yao intruder model [37,44]. The attacker in this model can traditionally intercept and originate network traffic, encrypt and decrypt captured messages as long as he knows the correct key, concatenate and split messages at will, generate certain types of messages (*e.g.*, keys and nonces) but not others (*e.g.*, principals), and access public data such as principal names.

This set of intruder capabilities is given a precise specification in MSR. For this purpose, the knowledge of the intruder is modeled as a collection of facts $I(m)$ ("the intruder knows message m") distributed in the state. The intruder himself is represented as the distinguished principal I by means of the declaration $\mathsf{I} : \mathsf{principal}$. Each capability is expressed by means of a one-rule role that can be executed only by I. For example, the specifications of network message interception, message splitting and nonce generation have the following form:

$$\overset{\mathsf{I}}{\left[\begin{array}{l}\forall m : \mathsf{msg}\\[4pt] \mathsf{N}(m) \xRightarrow{INT} I(m)\end{array}\right]} \qquad \overset{\mathsf{I}}{\left[\begin{array}{l}\forall m_1, m_2 : \mathsf{msg}\\[4pt] \begin{array}{l} I(m_1),\\ I(m_2)\end{array} \xRightarrow{CPM} I(m_1, m_2)\end{array}\right]} \qquad \overset{\mathsf{I}}{\left[\begin{array}{ll} \cdot \xRightarrow{NG} & \exists n : \mathsf{nonce}\\ & I(n)\end{array}\right]}$$

Notice that these roles identify I as their owner (above the left brace) rather than a generic principal (introduced by a universal quantifier in the previous section).

Because public-key Kerberos relies on more than shared keys, we extend our earlier intruder formalization with the following rules for public-key encryption and decryption. These allow the intruder to learn any public key of any principal P, or to learn any of his own secret keys (but not those of other principals), and then to encrypt and decrypt if the proper keys are known.

$$\overset{\mathsf{I}}{\left[\begin{array}{c}\forall P : \mathsf{principal}\ \ \forall pk : \mathsf{pubK}\ P\\ .\ \xRightarrow{PK'}\ I(pk)\end{array}\right]} \qquad \overset{\mathsf{I}}{\left[\begin{array}{c}\forall pk : \mathsf{pubK}\ I\ \ \forall sk : \mathsf{secK}\ sk\\ .\ \xRightarrow{SK'}\ I(sk)\end{array}\right]}$$

$$\overset{\mathsf{I}}{\left[\begin{array}{l}\forall P : \mathsf{principal}\ \ \forall pk : \mathsf{pubK}\ P\\ \forall m : \mathsf{msg}\\ I(m), I(pk)\ \xRightarrow{PEC'}\ I(\{\!\!\{m\}\!\!\}_{pk})\end{array}\right]} \qquad \overset{\mathsf{I}}{\left[\begin{array}{l}\forall P : \mathsf{principal}\ \ \forall pk : \mathsf{pubK}\ P\\ \forall sk : \mathsf{secK}\ pk\ \ \forall m : \mathsf{msg}\\ I(\{\!\!\{m\}\!\!\}_{pk}), I(sk)\ \xRightarrow{PDEC'}\ I(m)\end{array}\right]}$$

Signing is handled similarly as shown in rule *SIG* below. Rather than having the intruder verify signatures, an activity an attacker will rarely bother with (although it could easily be modeled in MSR), rule *PEEK* allows him to extract a message from a signature. It should be noted that this specification deemphasizes the PKI infrastructure on which PKINIT relies as it confines the use of certificates to the predicate *VerifySig*. A more explicit treatment is unnecessary in this case.

$$\overset{\mathsf{I}}{\left[\begin{array}{l}\forall P : \mathsf{principal}\ \ \forall pk : \mathsf{pubK}\ P\\ \forall sk : \mathsf{secK}\ pk\ \ \forall m : \mathsf{msg}\\ I(m),\ I(sk)\ \xRightarrow{SIG}\ I([m]_{sk})\end{array}\right]} \qquad \overset{\mathsf{I}}{\left[\begin{array}{c}\forall sk : \mathsf{someSecK}\ \ \forall m : \mathsf{msg}\\ I([m]_{sk})\ \xRightarrow{PEEK}\ I(m)\end{array}\right]}$$

Because secret keys are typed like long-term keys, we add specific rules allowing duplication and deletion in memory of known secret keys (paralleling the rules *DPD* and *DLD* from [42]). These are straightforward, and we omit the specific rules here.

The remaining rules implementing the other traditional Dolev-Yao intruder capabilities are defined similarly. A complete list for basic Kerberos can be found in [42].

6 Formal Analysis of PKINIT

Our formal proofs rely on a double induction aimed at separating the confidentiality and authentication aspects of the analysis of Kerberos 5. Confidentiality and authentication can interact in complex ways, requiring both types

$$\rho_k(m; m_0) = \begin{cases} 0, & m \text{ is an atomic term} \\ 0, & m = \{m_1\}_k, \rho_k(m_1; m_0) = 0, m_1 \neq m_0 \\ 1, & m = \{m_0\}_k \\ \rho_k(m_1; m_0), & m = \{m_1\}_{k'}, k' \neq k \\ \rho_k(m_1; m_0) + 1, & m = \{m_1\}_k, \rho_k(m_1; m_0) > 0 \\ 0, & m = \{\!\!\{m_1\}\!\!\}_k, \rho_k(m_1; m_0) = 0, m_1 \neq m_0 \\ 1, & m = \{\!\!\{m_0\}\!\!\}_k \\ \rho_k(m_1; m_0), & m = \{\!\!\{m_1\}\!\!\}_{k'}, k' \neq k \\ \rho_k(m_1; m_0) + 1, & m = \{\!\!\{m_1\}\!\!\}_k, \rho_k(m_1; m_0) > 0 \\ 0, & m = [m_1]_k, \rho_k(m_1; m_0) = 0, m_1 \neq m_0 \\ 1, & m = [m_0]_k \\ \rho_k(m_1; m_0), & m = [m_1]_{k'}, k' \neq k \\ \rho_k(m_1; m_0) + 1, & m = [m_1]_k, \rho_k(m_1; m_0) > 0 \\ \max\{\rho_k(m_1; m_0), & m = m_1, m_2 \\ \qquad \rho_k(m_2; m_0)\}, & \end{cases}$$

Fig. 12. The definition of $\rho_k(m; m_0)$, the k-rank of m relative to m_0.

of functions in a single proof. (This is not so much the case in the AS exchange, because the security of this first exchange does not rely on properties of earlier rounds, but it is seen clearly in the later rounds as illustrated in [7,42].) Our proofs are supported by two classes of functions, rank and corank, whose definitions we recall in Section 6.1. We prove the correctness of the fixed protocol in Section 6.2, discuss other authentication properties in Sections 6.3 and 6.4, and then state and prove auxiliary lemmas in Section 6.5.

6.1 Rank and Corank

We start by reviewing the definition of rank and corank functions as originally given in [7,8,42] and extended to include the cryptographic primitives that PKINIT adds to basic Kerberos. We start by defining these classes of functions inductively on terms and then extend the definitions to facts.

Rank. Rank captures the amount of work done using a specific key to encrypt a specified message. For a cryptographic key k and a fixed message m_0, we define the *k-rank of a message m relative to m_0* as shown in Figure 12.

If m is an atomic term, then no work has been done using k, and we set the rank to 0. If m is exactly $\{m_0\}_k$, $\{\!\!\{m_0\}\!\!\}_k$, or $[m_0]_k$, then we set the rank to 1.

$$\hat{\rho}_E(m;m_0) = \begin{cases} \infty, & m \text{ is atomic, } m \neq m_0 \\ 0, & m \text{ is atomic, } m = m_0 \\ \hat{\rho}_E(m_1;m_0)+1, & m = \{m_1\}_k,\ k \in E \\ \hat{\rho}_E(m_1;m_0), & m = \{m_1\}_k,\ k \notin E \\ \hat{\rho}_E(m_1;m_0)+1, & m = \{\!\{m_1\}\!\}_k, \exists k' : \mathsf{secK}\ k,\quad k' \in E \\ \hat{\rho}_E(m_1;m_0), & m = \{\!\{m_1\}\!\}_k, \forall k' : \mathsf{secK}\ k,\quad k' \notin E \\ \hat{\rho}_E(m_1;m_0), & m = [m_1]_k \\ \min\{\hat{\rho}_E(m_1;m_0), & m = m_1, m_2, \quad m_1, m_2 \neq \emptyset \\ \qquad \hat{\rho}_E(m_2;m_0)\}, & \end{cases}$$

Fig. 13. Definition of $\hat{\rho}_E(m;m_0)$, the E-corank of m relative to m_0.

Encrypting or signing a message m of positive k-rank will increase the rank by 1 in case one uses the key k, whereas using a key $k' \neq k$ will have no effect on the k-rank of m relative to m_0. The rank of the concatenation of two messages is set to be the larger of the ranks of the constituent messages.

The extension of the rank function to facts is straightforward. For a key k and for m, m_0 of type msg, and terms t, t_i, and P any predicate in the protocol signature, we define the k-rank of a fact $P(t_1, ..., t_j)$ relative to m_0 by $\rho_k(P(t_1, ..., t_j); m_0) = \max_{1 \leq i \leq j} \rho_k(t_i; m_0)$. *E.g.*, for network facts the k-rank relative to m_0 is $\rho_k(N(m); m_0) = \rho_k(m; m_0)$.

Our rank functions are most closely connected to data origin authentication, although they are used in conjunction with corank functions in many proofs [8]. The relationship between rank and authentication follows Theorem 6.1, which was outlined in [8].

Theorem 6.1 *If $\rho_k(F; m_0) = 0$ for every fact F in the initial state of a generic trace $\mathcal{T}$ and no intruder can increase the k-rank relative to m_0 then the existence of a fact F with $\rho_k(F; m_0) > 0$ in some non-initial state of $\mathcal{T}$ implies that some honest principal fired a rule which produced a fact build up from $\{m_0\}_k$.*

As in [42] (although stated there without some of the primitives needed to model PKINIT), if an intruder rule increases k-rank relative to m_0, then the left-hand side of that rule contains $I(k)$. Thus, if we show that $I(k)$ never appears in a trace, we may invoke Theorem 6.1 to help prove authentication.

Corank. Corank captures the minimum effort needed, using keys from a specified set, to extract a (possibly) secret message from a given term. For a set E of keys, none of which is a public key, and a fixed atomic message m_0, we define the *E-corank of message m relative to m_0* as shown in Figure 13. If m is atomic and $m = m_0$, then no work using keys from E is needed to obtain m_0, and we set the corank to 0. If m is atomic and $m \neq m_0$, then no amount of such work can extract m_0. The number of decryptions needed to obtain m_0 from $\{m\}_k$ using keys from E is 1 more than or the same as the number of decryptions needed to obtain m_0 from m, depending on whether k (if the encryption is symmetric) or some $k' : \mathsf{secK}\, k$ (if the encryption is asymmetric) is an element of E or not. The corank of the concatenation of two messages is equal to the smaller of the coranks of the constituent messages.

The extension of the corank function to facts is slightly different from that of the rank function. The definition of corank of a fact depends on the predicate P in which it occurs. For example, for the network predicate $\mathsf{N}()$ we define $\hat{\rho}_E(\mathsf{N}(m); m_0) = \hat{\rho}_E(m; m_0)$, while for the client's memory predicate Auth_C (used in the rules in Figures 11, A.1, and A.3) we define $\hat{\rho}_E(\mathsf{Auth}_C(t_1, t_2, t_3); m_0)$ $= \hat{\rho}_E(t_1; m_0)$. This follows our informal rule [7,8] that the corank of a fact depends only on the arguments of the predicate that might be put onto the network.

Our corank functions are closely connected to confidentiality, which we typically prove by invoking the following theorem from [8].

Theorem 6.2 *If $\hat{\rho}_E(F; m_0) > 0$ for every fact in the initial state of a generic trace $\mathcal{T}$, if no intruder can decrease the E-corank relative to m_0, and if no honest principal creates a fact F with $\hat{\rho}_E(F; m_0) = 0$, then m_0 is secret throughout $\mathcal{T}$.*

6.2 Correctness of the Fix

We now present the theorem that establishes the correctness of the abstract fix to PKINIT introduced in Section 4.1. This, in turn, implies the correctness of PKINIT-27 and subsequent versions of PKINIT (including the final specification)—*i.e.*, that Property 4.1 holds—because these use a special case of the abstract fix. In the following we will assume that $F(C, n_i) = F(C', n_i')$ implies $C = C'$ and $n_i = n_i'$, for any $C, C' : \mathsf{client}$ and $n_i, n_i' : \mathsf{nonce}$ (for $i = 1, 2$).

Theorem 6.3 ***If***

(1) the fact $\mathsf{N}(\{\!\{Cert_K, [k, F(C, n_i)]_{sk_K}\}\!\}_{pk_C}, C, X, \{AK, n_1, t_K, T\}_k)$ appears in a trace, for some $C : \mathsf{client}$, $K : \mathsf{KAS}$, $k : \mathsf{shK}\ C\ K$, $sk_K : \mathsf{someSecK}$, $X : \mathsf{msg}$, $Cert_K : \mathsf{CertList}$, $pk_C : \mathsf{pubK}\ C$, $T : \mathsf{TGS}$, $AK : \mathsf{shK}\ C\ T$, $n_i, n_1 : \mathsf{nonce}$, and $t_K : \mathsf{time}$;

(2) the fact $VerifySig([k, F(C, n_i)]_{sk_K}; (k, F(C, n_i)); K, Cert_K)$ holds; and

(3) *for every* pk_K : $\mathsf{pubK}\,K$ *and* sk : $\mathsf{secK}\,pk_K$*, the fact* $I(sk)$ *does not appear in the trace and no fact in the initial state of the trace contained a fact of positive* sk*-rank relative to* $(k, F(C, n_i))$*,*

then

the KAS K *fired rule* $\iota_{2.1}$*, consuming the fact* $\mathsf{N}(Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1)$ *and creating the fact* $\mathsf{N}(\{\!\{Cert_K, [k, F(C, n_i)]_{sk_K}\}\!\}_{pk_C}, C, \{AK, C, t_K\}_{k_T}, \{AK, n_1, t_K, T\}_k)$*, for some* $Cert_C$: $\mathsf{CertList}$*,* sk_C : secK pk_C*,* n_2 : nonce*,* t_C, t_K : time*,* k_T : dbK T*.*

PROOF. Because $VerifySig([k, F(C, n_i)]_{sk_K}; (k, F(C, n_i)); K, Cert_K)$ holds, by Lemma 6.10 there is some sk : secK pk_K such that $\rho_{sk}([k, F(C, n_i)]_{sk_K}; (k, F(C, n_i))) > 0$ (where pk_K : pubK K). Thus the fact $\mathsf{N}(\{\!\{Cert_C, [k, F(C, n_i)]_{sk_K}\}\!\}_{pk_C}, C, X, \{AK, n_1, t_K, T\}_k)$ has positive sk-rank relative to $(k, F(C, n_i))$; by hypothesis, no such fact existed in the initial state of the trace, so some rule firing during the trace must have increased this rank.

By hypothesis, $I(sk)$ does not appear in the initial state of the trace, so by Lemma 6.14 $I(sk)$ does not appear in any state of the trace. By Lemma 6.11, this means that no rule fired by the intruder can increase sk-rank relative to $(k, F(C, n_i))$; thus, by Theorem 6.1, at some point in this trace an honest principal must have fired a rule that increased this rank. By inspection of the principal rules, the only one that can do this is rule $\iota_{2.1}$; in order for this rule to do so, it must be fired by the KAS K who owns sk, consume a network fact $\mathsf{N}(Cert_{C'}, [t, n_2']_{sk_{C'}}, C', T', n_1')$ for some t : time, n_2', n_1' : nonce, $Cert_{C'}$: $\mathsf{CertList}$, C' : client, and T' : TGS, and produce the fact $\mathsf{N}(\{\!\{Cert_K, [k, F(C', n_i')]_{sk}\}\!\}_{pk_{C'}}, C', \{AK', C', t_K'\}_{k_{T'}}, \{AK', n_1', t_K', T'\}_k)$ for some $Cert_K$: $\mathsf{CertList}$, sk : $\mathsf{someSecK}$, $pk_{C'}$: $\mathsf{pubK}\,C'$, AK' : $\mathsf{shK}\,C'T'$, t_K' : time, $k_{T'}$: $\mathsf{dbK}\,T'$, n_1' : nonce, and $F(C', n_i') = F(C, n_i)$. By assumption, $(C', n_i') = (C, n_i)$, which implies that the request that K processed must match the request described in the hypotheses. $\square$

The following corollary specializes the result in Theorem 6.3 to the particular fix used in PKINIT-27 and to the client's receipt of the network message described in the hypotheses of this theorem. It is the formal statement of Property 4.1, which says that if C processes a reply message (containing the signed checksum of a request that C previously sent), then some KAS K must have sent a reply message intended for C.

Corollary 6.4 ***If***

(1) *some* C : client *fires rule* $\iota_{1.2}$*, consuming the fact* $\mathsf{N}(\{\!\{Cert_K, [k, ck]_{sk_K}\}\!\}_{pk_C}, C, X, \{AK, n_1, t_K, T\}_k)$ *and producing the fact* $Auth_C(X, T, AK)$ *for*

some K : KAS, k : $\mathsf{shK}\ C\ K$, sk_K : $\mathsf{someSecK}$, ck, X : msg, $Cert_K$: $\mathsf{CertList}$, pk_C : $\mathsf{pubK}\ C$, T : TGS, AK : $\mathsf{shK}\ C\ T$, n_1 : nonce, t_K : time, *and*

(2) $ck = H_k(Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1)$, *for some* t_C : time, n_2 : nonce, $Cert_C$: $\mathsf{CertList}$, sk_C : $\mathsf{SecK}\ pk_C$, *and*

(3) *for every* pk_K : $\mathsf{pubK}\ K$ *and* sk : $\mathsf{secK}\ pk_K$, *the fact* $I(sk)$ *does not appear in the trace and no fact in the initial state of the trace contained a fact of positive sk-rank relative to* (k, ck),

then

the KAS K *fired rule* $\iota_{2.1}$, *consuming the fact* $\mathsf{N}(Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1)$ *and creating the fact* $\mathsf{N}(\{\!\{Cert_K, [k, ck]_{sk_K}\}\!\}_{pk_C}, C, \{AK, C, t_K\}_{k_T}, \{AK, n_1, t_K, T\}_k)$, *for some* k_T : $\mathsf{dbK}\ T$, t_K : time.

PROOF. This follows by letting $F(C, n_i) = ck = H_k(Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1)$ in Theorem 6.3; this construction satisfies the assumptions on $F(C, n_i)$ for both $i = 1$ and $i = 2$. C's rule firing implies that the first two hypotheses of the theorem hold; the third hypothesis of the corollary specializes the third hypothesis of the theorem to the case of $ck = F(C, n_i)$, which implies the conclusion for some n'_1, n'_2 : nonce and $Cert'_C$: $\mathsf{CertList}$. C's firing of rule $\iota_{1.2}$ (when specialized to the checksum) implies that the $n'_1 = n_1$, $n'_2 = n_2$, and $Cert'_C = Cert_C$. $\square$

As a further remark, if in the abstract fix of PKINIT from Section 4.1 one chooses $F(C, n_i) = F(C, n_1)$, the proof of Theorem 6.3 shows that authentication of the KAS to the client still holds. In fact, the KAS does not return any information containing n_2. This means that the following holds:

Property 6.5 *The signed nonce* n_2 *in the client's AS request is superfluous for the purpose of authentication in PKINIT.*

This property does not imply that n_2 can simply be omitted from the first message of PKINIT in general, as some signed session identifiers is necessary to correctly support authentication as in Property 6.6 below. Rather, it suggests that it could be simplified by replacing every occurrence of n_2 with n_1.

6.3 Authentication of C to K

While our primary focus here is on the authentication of K to C, because C signs a nonce in her request we may also prove authentication of C to K; note that this holds in both the broken and fixed versions of PKINIT, as the fix does not affect C's request or our reasoning about it. Informally, we state this as the following property

Property 6.6 *If a KAS processes a PKINIT request from a client C, then previously C sent a PKINIT request that contained the signed data in the message received and processed by the KAS.*

Formally, we state this property as the following theorem.

Theorem 6.7 ***If***

(1) some K : KAS *fires rule* $\iota_{2.1}$*, consuming the fact* $\mathsf{N}(Cert_C, [t_C, n_2]_{sk}, C, T, n_1)$ *for some* $Cert_C$: $\mathsf{CertList}$, t_C : time, n_1, n_2 : nonce, sk : $\mathsf{someSecK}$, C : client, *and* T : TGS*; and*

(2) for every pk : $\mathsf{pubK}\, C$ *and* sk : $\mathsf{secK}\, pk$*, the fact* $I(sk)$ *does not appear in the trace and no fact in the initial state of the trace contained a fact of positive* sk*-rank relative to* (t_C, n_2),

then

at some point earlier in the trace, the client C *fired rule* $\iota_{1.1}$*, generating the fact* $\mathsf{N}(t_C, n_2, Cert'_C, [t_C, n_2]_{sk_C}, C, T', n'_1)$ *for some* $Cert'_C$: $\mathsf{CertList}$, T' : TGS*, and* n'_1 : nonce.

PROOF. Having assumed that K fired rule $\iota_{2.1}$, the fact $VerifySig([t_C, n_2]_{sk}; (t_C, n_2); C, Cert_C)$ must hold. By Lemma 6.10, the fact consumed by this rule firing has positive sk-rank relative to (t_C, n_2). By hypothesis, no such fact appeared in the initial state of the trace, so some rule that was fired during the trace increased this rank. Also by hypothesis, $I(sk)$ cannot appear in the trace, so by Lemma 6.11, the intruder cannot increase this rank. By inspection of the principal rules, the only rule that can increase this rank is rule $\iota_{1.1}$ when fired by C to create the fact $\mathsf{N}(t_C, n_2, Cert'_C, [t_C, n_2]_{sk_C}, C, T', n'_1)$ for some $Cert'_C$: $\mathsf{CertList}$, T' : TGS, and n'_1 : nonce. □

6.4 Authentication in the Pre-Fix Protocol

Our attack in Section 3 showed that an intruder could impersonate a KAS to a client. This means that KAS-to-client authentication does not hold when PKINIT-26 is used, in the sense that a client might receive a reply containing her name that appears to come from some KAS but without any KAS having ever sent such a message. However, PKINIT-26 does provide some authentication with respect to the later exchanges as shown by Property 6.8 below, and this property still holds in the fixed protocol (PKINIT-27 and later versions).

Property 6.8 *If TGS T processes a valid request message naming a client C, and the key used to encrypt the TGT which was contained in that request message was secret when the system started, then earlier the KAS K generated a TGT naming C. Furthermore, if C's private key was initially secret (so that C is honest), then C sent a request to T after K sent a corresponding AS reply to C.*

As a consequence, we see that even if the intruder is carrying out the attack by letting C login to T and S as the intruder (so that the intruder can watch the traffic between them), some KAS must have created a TGT for the intruder, formalizing the requirement that the intruder be a legal user of the system. Furthermore, this shows that while C might successfully 'request' a service ticket as the intruder in the attack scenario, it requires the participation of the intruder; in particular, C could not obtain a TGT that names an honest client C' if the relevant keys are secret.

Theorem 6.9 formalizes this property; the relevant MSR roles for the TG and CS exchanges can be found in Appendix A.

Theorem 6.9 ***If***

(1) $T : \mathsf{TGS}$ *fires rule* $\iota_{4.1}$ *consuming the fact* $\mathsf{N}(\{AK, C, t_K\}_{k_T}, \{AK, n_3, t_K, T\}_k, C, S)$ *and creating the fact* $\mathsf{N}(C, \{SK, C\}_{k_S}, \{SK, n_3, S\}_{AK})$ *for some* $AK : \mathsf{shK}\, C T$, $C : \mathsf{client}$, $K : \mathsf{KAS}$, $t_K : \mathsf{time}$, $k_T : \mathsf{dbK}\, T$, $n_3 : \mathsf{nonce}$, $S : \mathsf{server}$, *and*

(2) *the fact* $I(k_T)$ *does not appear in the trace and no fact in the initial state of the trace contained a fact of positive* k_T*-rank relative to* $\{AK, C, t_K\}$, *and*

(3) *the fact* $I(sk_C)$ *is not inferable in the initial state of the trace for every* $sk_C : \mathsf{SecK}\, pk_C$,

then

(i) *the KAS* K *earlier fired rule* $\iota_{2.1}$, *creating a fact* $\mathsf{N}(\{\!\{Cert_K, [k, H_k(Cert_C, [t_C, n_2]_{skC}, C, T, n_1)]_{sk_K}\}\!\}_{pk_C}, C, \{AK, C, t_K\}_{k_T}, \{AK, n_1, t_K, T\}_k)$ *for some* $Cert_K$, $Cert_C : \mathsf{CertList}$, $n_1, n_2 : \mathsf{nonce}$, *and*

(ii) C *fired rule* $\iota_{3.1}$, *creating the fact* $\mathsf{N}(\{AK, C, t_K\}_{k_T}, \{AK, n_3, t_K, T\}_k, C, S)$ *in a state later in the trace than the state at which* K *fired rule* $\iota_{2.1}$ *producing the fact described above.*

PROOF. We start with consideration of k_T-rank relative to (AK, C, t_K). T's hypothesized rule firing consumed a fact for which this rank is positive, but by assumption this rank was 0 for every fact in the initial state of the trace. Thus some rule firing in the trace increased this rank.

Because $I(k_T)$ does not appear in the initial state of the trace, by Lemma 6.13 this fact never appears in the trace. Thus, by Lemma 6.11, the intruder cannot have increased k_T-rank relative to (AK, C, t_K), so some honest principal must have done this. Inspection of the principal rules shows that if k has type $\mathsf{dbK}\,T$ for some $T : \mathsf{TGS}$, then the only honest principals that increase k-rank relative to any message are those of type KAS through the firing of rule $\iota_{2.1}$. Thus some $K : \mathsf{KAS}$ fired rule $\iota_{2.1}$, which must have consumed and produced the facts claimed.

We now show that AK is secret by considering $\{k, k_T\}$-corank relative to AK. K's firing of rule $\iota_{2.1}$ freshly generates AK, so all previous states had infinite $\{k, k_T\}$-corank relative to AK, and the state that results from this rule firing has $\{k, k_T\}$-corank equal to 1 relative to AK. As noted above, $I(k_T)$ never appears in the trace; we must show that $I(k)$ does not either. K's firing of rule $\iota_{2.1}$ also freshly generated k, so all previous states had infinite E-corank relative to k for every set E; furthermore, the resulting state has $\{sk_C\}$-corank equal to 1 relative to k. Because $I(sk_C)$ was not inferable from the initial state of the trace, by Lemma 6.14 this fact never appears in the trace. Thus, by Lemma 6.12, the intruder cannot decrease $\{sk_C\}$-corank relative to k. Inspection of the principal rules show that no honest principal will do this either; note that while C decrypts the message containing k, she does not store this key, much less in a predicate that will allow it to be put onto the network. Therefore no state in the trace has $\{sk_C\}$-corank equal to 0 relative to k, so $I(k)$ never appears in the trace.

Because neither $I(k_T)$ nor $I(k)$ ever appear in the trace, the intruder cannot decrease $\{k, k_T\}$-corank relative to AK. By inspection, we see that the only principal rule which decreases this corank is rule $\iota_{2.1}$ when it is fired to freshly generate AK. As noted above, the resulting state must have $\{k, k_T\}$-corank equal to 1 relative to AK; no principal rule will decrease this corank to 0. Thus no fact of $\{k, k_T\}$-corank equal to 0 relative to AK, and in particular the fact $I(AK)$, appears in the trace.

We may now show that C fired rule $\iota_{3.1}$ in the claimed fashion. T's hypothesized rule firing also consumed a fact of AK-rank equal to 1 relative to C, t_C. Because AK was freshly generated during the trace (as shown above) and $I(AK)$ never appears in the trace, some honest principal must have fired a rule that increased AK-rank relative to C, t_C. By inspection, we see that the only such principal rule is rule $\iota_{3.1}$, which must produce the fact $\mathsf{N}(X, \{C, t_C\}_{AK}, C, S, n_3)$ for some $X : \mathsf{msg}$, $t_C : \mathsf{time}$, $S : \mathsf{server}$, and $n_3 : \mathsf{nonce}$. Because this fact has positive AK-rank relative to C, t_C and AK was freshly generated by K, we know that C's firing of rule $\iota_{3.1}$ occurred after K's firing of rule $\iota_{2.1}$. $\square$

A number of lemmas giving conditions under which various ranks and coranks can be increased and decreased by rule firings can be found in [42]. However, we need to add to them as consider public-key operations where not considered in that work.

Lemma 6.10 *If* $VerifySig(s; m; P, Certs)$ *holds, then for some* $pk : \mathsf{pubK}\ P$ *and* $sk : \mathsf{secK}\ pk$*, the* sk*-rank of* s *relative to* m *is positive.*

PROOF. Our assumptions about $VerifySig$ imply that s is a valid signature of m under one of P's secret keys, *i.e.*, for some $pk : \mathsf{pubK}\ P$ and $sk : \mathsf{secK}\ pk$, $s = [m]_{sk}$. Thus s has positive sk-rank relative to m. □

The following lemmas were proved in [42] for the formalizations of basic Kerberos considered there. These property still holds here once we add asymmetric encryption and digital signatures.

Lemma 6.11 *If an intruder rule* R *can increase* k*-rank relative to a message* m_0*, then the left-hand side of* R *contains* $I(k)$*.*

PROOF. By inspection of the intruder rules. □

Lemma 6.12 *If* m_0 *is not a principal name, time, or key of one of the types* $\mathsf{dbK}\ I$*,* $\mathsf{shK}\ I\ A$ *(for* $A : \mathsf{TGS}$ *or* $A : \mathsf{server}$*),* $\mathsf{shK}\ C\ I$ *for* $C : \mathsf{client}$*, or* $\mathsf{pubK}\ P$ *for* $P : \mathsf{principal}$*, then any intruder rule that decreases* E*-corank relative to* m_0 *either contains* $I(k)$ *in its left hand side for some* $k \in E$ *or freshly generates* m_0*.*

PROOF. By inspection of the intruder rules. □

The following lemma is an analog of Lemma 6 in [42]; the additional intruder rules do not change it.

Lemma 6.13 *For any* $P : \mathsf{principal}$*,* $P \neq I$*, and* $k : \mathsf{dbK}\ P$*, if* $I(k)$ *does not appear in the initial state of the trace, then* $I(k)$ *does not appear in any state of the trace.*

PROOF. By inspection of the intruder rules. Because $\mathsf{dbK}\,P$ is not a subtype of msg, if $P \neq I$ then $I(k)$ appears on the right-hand side of a rule only if it also appears on the left-hand side. □

The next lemma is analogous to the previous one, but it refers to secret keys for asymmetric encryption instead of the keys in the KAS's database.

Lemma 6.14 *For any $P : \mathsf{principal}$, $pubk : \mathsf{pubK}\,P$, $P \neq I$, and $k : \mathsf{secK}\,pubk$, if $I(k)$ does not appear in the initial state of the trace, then $I(k)$ does not appear in any state of the trace.*

PROOF. By inspection of the intruder rules. Because $\mathsf{secK}\ pubk$ is not a subtype of msg, for $k : \mathsf{secK}\,pubk$, the only time $I(k)$ may appear on the right-hand side of an intruder rule but not on the left-hand side is in rule $\textsc{sk}'$, in which case $P = I$. □

7 Conclusions and Future Work

In this paper, we describe our discovery of a man-in-the-middle attack against PKINIT [25], the public key extension to the popular Kerberos 5 authentication protocol [1]. The attack was found on PKINIT-25, but applies to previous versions as well. We found this attack as part of an ongoing formal analysis of Kerberos, which has previously yielded proofs of security for the core Kerberos 5 protocol [7,8] and its use for cross-realm authentication [9]. We have used formal methods approaches to prove that, at an abstract level, several possible defenses against our attack restore security properties of Kerberos 5 that are violated in PKINIT (as shown by the attack); we also proved some security properties that do hold even for the vulnerable form of PKINIT. The fixes we analyzed include the one proposed by the IETF Kerberos Working group, which included it in the specification of PKINIT starting with revision 27 [25]. Our attack was also addressed in a Microsoft security bulletin affecting several versions of Windows [4] and mentioned in a CERT advisory [26].

As a continuation of this research, we have carried over some of the results examined here to the computational model by expressing PKINIT and other aspects of Kerberos in the cryptographic library model of Backes, Pfitzmann and Waidner (BPW model) [45]. The main outcome of this effort was that the fixes examined here were proved to be correct at the cryptographic level [21]. The computationally sound proofs in the BPW model were conducted symbolically by hand [21] and there appears to be a strong relation to the symbolic proof technique used in this work. Gaining a better understanding of how these two methods relate will be subject to future work. If a suitable connection to the BPW framework is discovered, it could contribute to the automation of proof in the cryptographic model [46], which will have the effect of speeding up the analysis work and, therefore, allowing us to tackle larger protocols. We have also started to mechanically verify security properties of PKINIT directly within the computational model [22] using Blanchet's CryptoVerif tool [23,24]. We are also in the process of extending our analysis to the Diffie-Hellman mode of PKINIT: our preliminary observations suggest that it is immune from the attack described in this paper, but we do not yet have definite results on other types of threats.

References

[1] C. Neuman, T. Yu, S. Hartman, K. Raeburn, The Kerberos Network Authentication Service (V5), `http://www.ietf.org/rfc/rfc4120` (July 2005).

[2] M. Thomas, J. Vilhuber, Kerberized Internet Negotiation of Keys (KINK), `http://ietfreport.isoc.org/all-ids/draft-ietf-kink-kink-06.txt` (Dec. 2003).

[3] J. Kohl, C. Neuman, The Kerberos Network Authentication Service (V5), `http://www.ietf.org/rfc/rfc1510` (September 1993).

[4] Microsoft, Security Bulletin MS05-042, `http://www.microsoft.com/technet/security/bulletin/MS05-042.mspx` (Aug. 2005).

[5] M. Strasser, A. Steffen, Kerberos PKINIT Implementation for Unix Clients, Tech. rep., Zurich University of Applied Sciences Winterthur (2002).

[6] T. Yu, S. Hartman, K. Raeburn, The perils of unauthenticated encryption: Kerberos version 4, in: Proc. NDSS'04, 2004.

[7] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, An Analysis of Some Properties of Kerberos 5 Using MSR, in: Proc. CSFW'02, 2002, pp. 175–190.

[8] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, C. Walstad, Formal Analysis of Kerberos 5, Theoretical Computer Science 367 (1–2) (2006) 57–87.

[9] I. Cervesato, A. D. Jaggard, A. Scedrov, C. Walstad, Specifying Kerberos 5 Cross-Realm Authentication, in: Proc. WITS'05, ACM Digital Lib., 2005, pp. 12–26.

[10] M. Abadi, B. Blanchet, Analyzing Security Protocols with Secrecy Types and Logic Programs, Journal of the ACM 52 (1) (2005) 102–146.

[11] M. Anlauff, D. Pavlovic, R. Waldinger, S. Westfold, Proving authentication properties in the Protocol Derivation Assistant, in: P. Degano, R. Küsters, L. Vigano (Eds.), Proceedings of FCS-ARSPA 2006, ACM, 2006, to appear. URL `ftp://ftp.kestrel.edu/pub/papers/pavlovic/FCS-ARSPA06.p%df`

[12] B. Blanchet, An Efficient Cryptographic Protocol Verifier Based on Prolog Rules, in: 14th IEEE Computer Security Foundations Workshop (CSFW-14), IEEE Computer Society, Cape Breton, Nova Scotia, Canada, 2001, pp. 82–96.

[13] B. Blanchet, From Secrecy to Authenticity in Security Protocols, in: M. Hermenegildo, G. Puebla (Eds.), 9th International Static Analysis Symposium (SAS'02), Vol. 2477 of Lecture Notes on Computer Science, Springer Verlag, Madrid, Spain, 2002, pp. 342–359.

[14] S. F. Doghmi, J. D. Guttman, F. J. Thayer, Searching for shapes in cryptographic protocols, in: Proceedings of TACAS 2007, 2007.

[15] L. Viganò, Automated security protocol analysis with the AVISPA tool., Electr. Notes Theor. Comput. Sci. 155 (2006) 61–86.

[16] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, L. Vigneron, The AVISPA tool for the automated validation of internet security protocols and applications., in: K. Etessami, S. K. Rajamani (Eds.), CAV, Vol. 3576 of Lecture Notes in Computer Science, Springer, 2005, pp. 281–285.

[17] C. Cremers, Scyther - semantics and verification of security protocols, Ph.D. dissertation, Eindhoven University of Technology (2006).

[18] R. Kemmerer, C. Meadows, J. Millen, Three systems for cryptographic protocol analysis, J. Cryptology 7 (1994) 79–130.

[19] C. Meadows, Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer., in: Proc. IEEE Symp. Security and Privacy, 1999, pp. 216–231.

[20] J. C. Mitchell, V. Shmatikov, U. Stern, Finite-State Analysis of SSL 3.0, in: Proc. 7^{th} USENIX Security Symp., 1998, pp. 201–216.

[21] M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, Cryptographically Sound Security Proofs for Basic and Public-key Kerberos, in: D. Gollmann, J. Meier, A. Sabelfeld (Eds.), Proc. ESORICS'06, Springer LNCS 4189, 2006, pp. 362–383.

[22] A. D. Jaggard, A. Scedrov, J.-K. Tsay, Computational Sound Mechanized Proof of PKINIT for Kerberos, presented at FCC'07 (no proceedings) (2007).

[23] B. Blanchet, A computationally sound mechanized prover for security protocols, in: IEEE Symposium on Security and Privacy, Oakland, California, 2006, pp. 140–154.

[24] B. Blanchet, Computationally sound mechanized proofs of correspondence assertions, in: 20th IEEE Computer Security Foundations Symposium (CSF'07), IEEE, Venice, Italy, 2007, to appear.

[25] IETF, Public Key Cryptography for Initial Authentication in Kerberos, RFC 4556. Preliminary versions available as a sequence of Internet Drafts at `http://tools.ietf.org/wg/krb-wg/draft-ietf-cat-kerberos-pk-init/` (1996–2006).

[26] CERT, Vulnerability Note 477341, `http://www.kb.cert.org/vuls/id/477341` (2005).

[27] I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, C. Walstad, Breaking and Fixing Public-key Kerberos, in: Proceedings of ASIAN'06, 2006, pp. 164–178.

[28] J. De Clercq, M. Balladelli, Windows 2000 authentication, `http://www.windowsitlibrary.com/Content/617/06/6.html`, digital Press (2001).

[29] Cable Television Laboratories, Inc., PacketCable Security Specification, technical document PKT-SP-SEC-I11-040730 (2004).

[30] S. Goldwasser, S. Micali, R. L. Rivest, A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks, SIAM J. Computing 17 (1988) 281–308.

[31] I. Cervesato, Typed MSR: Syntax and Examples, in: Proc. MMM'01, Springer LNCS 2052, 2001, pp. 159–177.

[32] N. A. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, Multiset Rewriting and the Complexity of Bounded Security Protocols, J. Comp. Security 12 (2) (2004) 247–311.

[33] W. Diffie, P. C. van Oorschot, M. J. Wiener, Authentication and authenticated key exchanges, Designs, Codes and Cryptography 2 (2) (1992) 107–125.

[34] R. Canetti, H. Krawczyk, Security Analysis of IKE's Signature-Based Key-Exchange Protocol, in: Proc. CRYPTO'02, Springer LNCS 2442, 2002, pp. 143–161.

[35] D. Harkins, D. Carrel, The Internet Key Exchange (IKE), `http://www.ietf.org/rfc/rfc2409` (Nov. 1998).

[36] G. Lowe, Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR, in: Proc. TACAS'96, Springer LNCS 1055, 1996, pp. 147–166.

[37] R. Needham, M. Schroeder, Using Encryption for Authentication in Large Networks of Computers, Comm. ACM 21 (12) (1978) 993–999.

[38] J. Clark, J. Jacob, On the security of recent protocols, Information Processing Letters 56 (3) (1995) 151–155.

[39] T. Hwang, Y.-H. Chen, On the Security of SPLICE/AS — The Authentication System in WIDE Internet, Information Processing Letters 53 (2) (1995) 91–101.

[40] M. Abadi, R. Needham, Prudent Engineering Practice for Cryptographic Protocols, IEEE Trans. Software Eng. 22 (1) (1996) 6–15.

[41] K. Raeburn, Encryption and Checksum Specifications for Kerberos 5, `http://www.ietf.org/rfc/rfc3961.txt` (Feb. 2005).

[42] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, Confidentiality and Authentication in Kerberos 5, Tech. Rep. MS-CIS-04-04, UPenn (2004).

[43] I. Cervesato, A Specification Language for Crypto-Protocols based on Multiset Rewriting, Dependent Types and Subsorting, in: G. Delzanno, S. Etalle, M. Gabbrielli (Eds.), Workshop on Specification, Analysis and Validation for Emerging Technologies — SAVE'01, Paphos, Cyprus, 2001, pp. 1–22.

[44] D. Dolev, A. Yao, On the security of public-key protocols, IEEE Trans. Info. Theory 2 (29) (1983) 198–208.

[45] M. Backes, B. Pfitzmann, M. Waidner, A Composable Cryptographic Library with Nested Operations, in: Proc. CCS'03, ACM, 2003, pp. 220–230.

[46] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, M. Waidner, Cryptographically sound theorem proving, in: Proc., CSFW '06, 2006, pp. 153–166.

A MSR Roles for Later Exchanges

Here we recall the MSR roles for the TG and CS exchanges. These are the same as for basic Kerberos because PKINIT only modifies the AS exchange.

Figure A.1 shows the client's role in the TG exchange. The memory predicate $\mathsf{Auth}_C(X, T, AK)$ comes from the AS exchange (formalized in Figure 11 above).

Figure A.2 shows the TGS's role in the TG exchange. Figures A.3 and A.4 show the client and server roles, respectively, for the CS exchange.

$$\forall C : \mathsf{client}$$

$$\left[\begin{array}{l}
\forall X : \mathsf{msg} \quad \forall T : \mathsf{TGS} \quad \forall S : \mathsf{server} \quad \forall AK : \mathsf{shK}\ C\ T \quad \forall t_C : \mathsf{time} \quad \forall S : \mathsf{server} \\
\qquad \mathsf{Auth}_C(X,T,AK) \overset{\iota 3.1}{\Longrightarrow} \begin{array}{l} \exists n_3 : \mathsf{nonce} \\ \quad \mathsf{N}(X, \{C, t_C\}_{AK}, C, S, n_3), \\ \quad MemTGE(C,T,AK,n_3), \\ \quad \mathsf{Auth}_C(X,T,AK) \end{array} \\
\quad \mathrm{IF} \quad \mathsf{clock}_C(t_C) \\
\\
\forall Y : \mathsf{msg} \quad \forall S : \mathsf{server} \quad \forall SK : \mathsf{shK}\ C\ S \quad \forall n_3 : \mathsf{nonce} \quad \forall t_T : \mathsf{time} \\
\quad \begin{array}{l} \mathsf{N}(C, Y, \{SK, n_3, t_T, S\}_{AK}), \\ MemTGE(C,T,AK,n_3) \end{array} \overset{\iota 3.2}{\Longrightarrow} \mathsf{Auth}_C(Y,S,SK)
\end{array}\right]$$

Fig. A.1. The client's role in TG exchange.

$$\forall T : \mathsf{TGS}$$

$$\left[\begin{array}{l}
\forall C : \mathsf{client} \quad \forall S : \mathsf{server} \quad \forall AK : \mathsf{shK}\ C\ T \quad \forall k_T : \mathsf{dbK}\ T \quad \forall k_S : \mathsf{dbK}\ S \\
\forall n_3 : \mathsf{nonce} \quad \forall t_T, t_C, t : \mathsf{time.} \\
\begin{array}{l} \mathsf{N}(\{AK, C, t\}_{k_T}, \{C, t_C\}_{AK}, \\ \quad C, S, n_3) \end{array} \overset{\iota 4.1}{\Longrightarrow} \begin{array}{l} \exists SK : \mathsf{shK}\ C\ S \\ \quad \mathsf{N}(C, \{SK, C, t_T\}_{k_S}, \{SK, n_3, t_T, S\}_{AK}), \end{array} \\
\\
\mathrm{IF} \quad \mathsf{Valid}_T(C, S, t_C),\ \mathsf{clock}_T(t_T)
\end{array}\right]$$

Fig. A.2. The TGS's role in the TG exchange.

$$\forall C : \mathsf{client}$$

$$\left[\begin{array}{l}
\forall S : \mathsf{server} \quad \forall SK : \mathsf{shK}\ C\ S \quad \forall t'_C : \mathsf{time} \quad \forall Y : \mathsf{msg.} \\
\mathsf{Auth}_C(Y,S,SK) \overset{\iota 5.1}{\Longrightarrow} \begin{array}{l} \mathsf{N}(Y, \{C, t'_C\}_{SK}), \\ \mathsf{Auth}_C(Y,S,SK) \end{array} \\
\mathrm{IF} \quad \mathsf{clock}_C(t'_C)
\end{array}\right]$$

Fig. A.3. The client's role in the CS exchange.

$$\forall S : \mathsf{server}$$

$$\left[\begin{array}{l} \forall C : \mathsf{client} \quad \forall T : \mathsf{TGS} \quad \forall SK : \mathsf{shK}\ C\ S \quad \forall t'_C, t_T : \mathsf{time} \quad \forall k_S : \mathsf{dbK}\ S \\ \\ \mathsf{N}(\{SK, C, t_T\}_{k_S}, \{C, t'_C\}_{SK}) \xRightarrow{\iota 6.1} \begin{array}{l} \mathsf{N}(\{t'_c\}_{SK}), \\ Mem_S(C, SK, t'_C) \end{array} \\ \text{IF} \quad \mathsf{Valid}_S(C, t'_C) \end{array}\right]$$

Fig. A.4. The server's role in the CS exchange.