Breaking and Fixing Public-Key Kerberos*

I. Cervesato[1], A. D. Jaggard[1], A. Scedrov[2], J.-K. Tsay[2], and C. Walstad[2]

[1] Tulane University
{iliano|adj}@math.tulane.edu
[2] University of Pennsylvania
{scedrov@math|jetsay@math|cwalstad@seas}.upenn.edu

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 fixes to PKINIT that prevent our attack.

1 Introduction

Kerberos [1, 2] 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 [3]. Introduced in the early 1990s [4], 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; in the process, it adds a fair amount of 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 in recent drafts of PKINIT, 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

* This work is part of an on-going effort to formally analyze Kerberos 5 partially supported by ONR and NSF.

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, and Windows Server 2003 [5]; it has also been included in Heimdal since 2002 [6]. PKINIT is not yet supported in the MIT reference implementation.

The flaw we have uncovered in PKINIT [7] 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 attackers also obtains all the keys that the KDC would normally generate for the client to encrypt her service requests, hence compromising confidentiality as well. Note that 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 [8], our 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.[3] More precisely, the second message of this exchange (the reply) can easily be modified as to appear to correspond to a request (the first message) issued by a client different from the one for which it was generated. 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 although we have not yet proved 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 [9] and of cross realm authentication [10]. Although our work is carried out by hand, automated approaches exist and have also been applied to deployed protocols [11–13].

After discovering the attack on PKINIT, we worked in close collaboration with the IETF Kerberos Working Group, and 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

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

attack [14], as well as the particular instance proposed by the Working Group which has been adopted in the latest version [15] of the PKINIT specification. Our attack led to an August 2005 Microsoft security patch and bulletin [5].

2 Kerberos 5 and its Public-Key Extension

Networked computer systems put a multitude of shared resources at a user's fingertips: remote hosts, file servers, printers, and many other networked services are readily available without leaving one's desk. Authentication and other security mechanisms need to be in place so that this convenience is not abused, in particular now when connections to the Internet provide dangerous back-doors to one's organization. The Kerberos protocol [1, 2] allows a legitimate user to log on to her terminal once a day (typically) and then transparently access all the networked resources she needed 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 will now briefly review how Kerberos achieves secure authentication based on a single logon. We will be particularly interested in the initial exchange, which happens when the user first logs on, and review the messages in this exchange both with and without PKINIT.

Kerberos Basics The client process—usually acting for a human user—interacts with three other types of principals when using Kerberos 5 (with or without PKINIT). The client's goal is to be able to authenticate herself to various application servers (*e.g.*, email, file, and print servers). This is done by obtaining a "ticket-granting ticket" (TGT) from a "Kerberos Authentication Server" (KAS) and then presenting this to a "Ticket-Granting Server" (TGS) in order to obtain a "service ticket" (ST), the credential that the client uses to authenticate herself to the application server. A TGT might be valid for a day, and may be used to obtain several STs for many different application servers from the TGS, while a single ST is valid for a few minutes (although it may be used repeatedly) and is 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 AS exchange is given in Figure 1 once we ignore the boxed items. 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 some 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 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. 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 ticket-granting ticket, sent alongside the message encrypted for the client. The TGT is encrypted under a long-term key 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 but including them would obscure their exposition. We refer the reader to [2] for the complete specification of Kerberos 5, and to [9] for a formalization at an intermediate level of detail.

Public-Key Kerberos PKINIT [15] is an extension to Kerberos 5 that uses public key cryptography to avoid shared secrets between a client and KAS; it modifies the AS exchange but not other parts of the basic Kerberos 5 protocol. The long-term shared key (k_C) in the traditional AS exchange is typically derived from a password, which limits the strength of the authentication to the user's ability to choose and remember good passwords; PKINIT does not use k_C and thus avoids this problem. Furthermore, if a public key infrastructure (PKI) is already in place, PKINIT allows network administrators to use it rather than expending additional effort to manage users' long-term keys 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 [16]. 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 been defined in a sequence of Internet Drafts [15], we use "PKINIT-n" to refer to the protocol as specified in the n^{th} draft revision and "PKINIT" for the generic protocol. These various drafts are archived at [15]. We discovered the attack described in

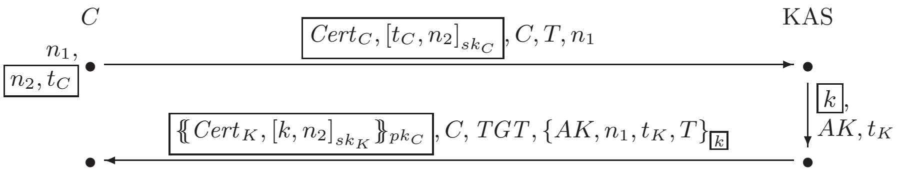

Fig. 1. Message Flow in the Traditional AS Exchange and in $\boxed{\text{PKINIT-26}}$

Sect. 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 Sect. 4. The current version of the protocol is PKINIT-30, which does not differ from the parts of PKINIT-27 we discuss here.

PKINIT can operate in two modes. In *Diffie-Hellman (DH) mode*, the key pairs (pk_C, sk_C) and (pk_K, sk_K) are used to provide digital signature support for an authenticated Diffie-Hellman key agreement which is 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. In *public-key encryption mode*, the key pairs are used for both signature and encryption. The latter is designed to (indirectly) protect the confidentiality of AK, while the former ensures its integrity.

We will not discuss the DH mode any further as our preliminary investigation did not reveal any flaw in it; we are still working on a complete analysis of this mode. Furthermore, it appears not to have yet been included in any of the major operating systems. The only support we are aware of is within the PacketCable system [17], developed by CableLabs, a cable television research consortium.

Figure 1, including boxed terms, illustrates the AS exchange in PKINIT-26. 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 Sect. 6, we assume that digital signatures are unforgeable [18]. The encryption of m with public key pk is denoted $\{\!\{m\}\!\}_{pk}$. As usual, we write $\{m\}_k$ for the encryption of m with symmetric key k.

The first line of Figure 1 shows our formalization 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 $\boxed{\text{boxed}}$ parts are added by PKINIT and contain 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

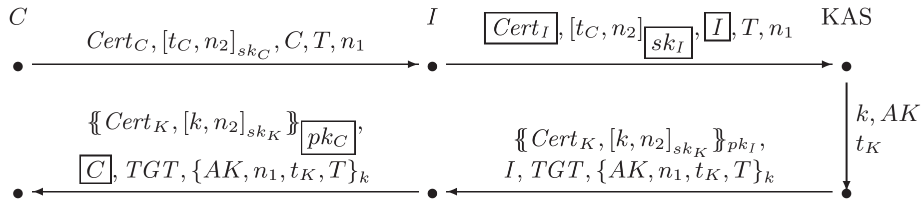

Fig. 2. Message Flow in the Man-In-The-Middle Attack on PKINIT-26

specifically for this request, with the box indicating data not included in our abstract formalization of basic Kerberos 5 [9].)

The second line in Figure 1 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) 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 is encrypted with a long-term key shared between K and the TGS T; the TGT contains C's name, AK, and other data. The message encrypted under k includes AK, the nonce n_1 from the request, K's local time t_K, and the name T of the TGS. 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 (boxed) message $\{\!\{Cert_K, [k, n_2]_{sk_K}\}\!\}_{pk_C}$. 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 [15]. Also, recall that PKINIT leaves the subsequent exchanges of Kerberos unchanged.

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 [15] in preparation for its formalization in MSR. 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.

Message Flow Figure 2 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 $[msg]_k$ can be understood as an abbreviation for the plaintext msg together with a keyed hash—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 Fig. 2. (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 Fig. 2). The ticket-granting ticket, denoted TGT, has the form $\{AK, I, t_K\}_{k_T}$; note that, since it is encrypted with the key k_T shared between K and the TGS T, it is opaque to C. 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 this part of the message using his public 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 [19] 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 [20] observed that the "basic authenticated Diffie-Hellman" mode of the Internet Key Exchange protocol (IKE) [21] had this very same vulnerability. In 1996, Lowe [22] found an attack on the Needham-Schroeder public key protocol [23] which 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 [22] and [19].

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 [17, 16].

Effects of the attack 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) or simply impersonate the later servers. In the first case, once C has AK and a TGT, she would normally contact the TGS to get a service ticket for some application server S. This request contains an *authenticator* of the form $\{C, t'_C\}_{AK}$ (*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: $\{I, t'_C\}_{AK}$. 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 a service ticket that is opaque to all but the TGS and application server. I may intercept this message and learn SK, replace an instance of his 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, which 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.

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. For the exchange with the TGS, 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, which the client expects to be opaque, and a fresh key SK encrypted under AK, and send these to C in what appears to be a properly formatted reply from the TGS. This very same behavior can be perpetrated at the next phase, by which C requests service to the end-server S, for communicating with whom the key SK was purportedly generated. Note that the attacker may take the first approach in the TG exchange and then the second in the CS exchange. The reverse is not possible because I cannot forge a valid service ticket.

Regardless of which approach the attacker uses to propagate the attack throughout the protocol run, C finishes the CS exchange believing that she has done so 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. 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 is possible because the two messages constituting PKINIT, as specified in [15], are insufficiently bound to each other.

More precisely, it 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 ticket granting ticket 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 target is the submessage signed by the KAS in the reply.

Following a methodology we successfully applied in previous work on Kerberos [9, 10], 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 Sect. 6 and a full account in [14]. Property 1 below shows the informal statement of the property that we saw violated in PKINIT-26 but that holds of subsequent revisions, hence demonstrating that this fix does indeed defend against our attack.

Property 1. *In PKINIT-27 (and subsequent versions), whenever a client C processes a message containing server-generated public-key credentials, the KAS previously produced such credentials for C.*

This property follows from a corollary to Thm. 2, which we prove in Sect. 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, proposed by the Working Group, that was included in the most recent revision [15] of the PKINIT specification.

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 the client's name, C, in the reply, in such a way that it cannot be modified *en route* and that the client can check it. Following well-established recommendations [24], we initially proposed a simple and minimally intrusive approach to doing so, which consists in mentioning C 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 $F(C, n_i)$ built from C and at least one of the nonces n_1, n_2 from C's request for credentials. With this abstract fix in place, the PKINIT exchange in public-key encryption mode appears as follows, where we have used a box to highlight the modification with respect to PKINIT-26.

$$\begin{array}{lcl} C & & \text{KAS} \\ n_1, n_2, t_C \bullet & \xrightarrow{\quad Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1 \quad} & \bullet \\ & & \downarrow k, AK, t_K \\ \bullet & \xleftarrow{\quad \{\!\{ Cert_K, [k, \boxed{F(C, n_i)}]_{sk_K} \}\!\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k \quad} & \bullet \end{array}$$

Here, F represents any construction that involves C and n_1 or n_2, 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). 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 by C. The presence of the nonces n_1 or n_2 uniquely identifies which of the (possibly several) requests of C this reply corresponds to. 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.

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 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.

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 checksum-based approach was later included in PKINIT-27 and its revisions [15]. This version of PKINIT has the following intended message flow:

$$
\begin{array}{lcl}
C & & \text{KAS} \\
n_1, n_2, t_C \;\bullet & \xrightarrow{\quad Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1 \quad} & \bullet \\
& & \downarrow k, AK, t_K \\
\bullet & \xleftarrow{\quad \{\!\{Cert_K, [k, \boxed{ck}]_{sk_K}\}\!\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k \quad} & \bullet
\end{array}
$$

Here, ck is a checksum of the client's request keyed with the key k, that is $ck = H_k(Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1)$. The function H can be any preimage-resistant keyed checksum. 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 [25], which discusses cryptographic algorithms for use in the Kerberos protocol suite, current candidates include hmac-sha1-96-aes128. New strong keyed checksums can be used for ck as they are developed. In future work, we intend to study PKINIT at the computational level, which will allow us to take into account the difficulty of finding a message with a given checksum. This will refine the purely formal work outlined here.

5 Formalizing PKINIT in MSR

MSR [10, 26, 27] 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).

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 ("$\_, \_$"). 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 msg; using a nonce as the key yields an ill-formed message. 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 [26]. 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 shared between a particular client C and server S is expressed by declaring it 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 $\mathsf{dbK}\ A$, again a subsort of key, but *not* of msg. 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 $\mathsf{secK}\ pk$. The constructors for encryption and digital signature are written $\{\!\{m\}\!\}_{pk}$ and $[m]_{sk}$, respectively.

Other types used in the formalization of PKINIT include time for timestamps, $\mathsf{CertList}$ for lists of digital certificates, and $\mathsf{someSecK}$ as an auxiliary type for working with digital signatures. We also use the constructor $H_k(m)$ to model the checksum (keyed hash) of message m keyed with symmetric key k.

States, Rules, and the Formalization of PKINIT-27 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 ticket granting ticket $\{AK, C, t_K\}_{k_T}$ is present on the network, and $I(\{AK, C, t_K\}_{k_T})$ that it has been captured by the attacker.

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

Fig. 3. KAS's Role in the PKINIT-27 Version of the AS Exchange

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, Fig. 3 describes the actions of the KAS in PKINIT-27 (see Sect. 4). Ignoring for the moment the leading $\forall K : \mathsf{KAS}$ 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 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 $\mathsf{N}(C, T, n_1)$ 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 ($\mathit{VerifySig}(s; m; P, Certs)$ holds if s is the signature, relative to certificates $Certs$, by principal P over the message m; we assume that this implies P has a key k such that the rank function $\rho_k(s; m) > 0$), $Valid_K$ 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, they are often handled procedurally in practice, 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 Fig. 3. The two-rule role describing the client's actions in the AS exchange has been omitted here for space reasons. The interested reader is referred to [14]. Formalizations of the TG and CS exchanges can be found in [9, 28, 10].

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. They are supported by two classes of functions, rank and corank, defined recursively on MSR specifications [9, 28]. In general, rank captures the amount of work done to generate a message and is connected to authentication, while corank captures the minimum effort needed to extract a secret and relates to confidentiality. Confidentiality and authentication can interact in complex ways, requiring both types of functions in a single proof. (This is not so much the case in the AS exchange, because it is the first exchange in Kerberos, but it is seen clearly in the later rounds as illustrated in [9, 28].)

As a taste of our methodology, we present the theorem which establishes that PKINIT-27 and subsequent versions meet the expected authentication goals. It is indeed the formal statement of Prop. 1 in Sect. 4. The interested reader is referred to [14] for the remaining properties of PKINIT and detailed proofs. We assume that digital signatures are unforgeable and we abstract collision resistance as the injectivity of $H_{\_}$.

Theorem 2. ***If*** *(1) the fact* $\mathsf{N}(\{\!\{Cert_C, [k, ck]_{sk_K}\}\!\}_{pk_C}, C, X, \{AK, n_1, t_K, T\}_k)$ *appears in a trace*[4]*; (2)* $ck = H_k(Cert_K, [t_C, n_2]_{sk_C}, C, T, n_1)$[5]*; (3) the fact* $\mathit{VerifySig}([k, ck]_{sk_K}; k, ck; K, Cert_K)$ *holds; and (4) 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*** K *fired rule* $\iota_{2.1}$*, consuming the fact* $\mathsf{N}(Cert_K, [t_C, n_2]_{sk_C}, C, T, n_1)$ *and creating the fact* $\mathsf{N}(\{\!\{Cert_C, [k, ck]_{sk_K}\}\!\}_{pk_C}, C, \{AK, C, t_K\}_{k_T}, \{AK, n_1, t_K, T\}_k)$.[6]

Proof. (Sketch) Because $\mathit{VerifySig}([k, ck]_{sk_K}; k, ck; K, Cert_K)$ holds, by assumption about the properties of $\mathit{VerifySig}$ there is some $sk : \mathsf{secK}\, pk_K$ such that $\rho_{sk}([k, ck]_{sk_K}; (k, ck)) > 0$ (where $pk_K : \mathsf{pubK}\, K$). Thus the fact $\mathsf{N}(\{\!\{Cert_C, [k, ck]_{sk_K}\}\!\}_{pk_C}, C, X, \{AK, n_1, t_K, T\}_k)$ has positive sk-rank relative to k, ck; 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.

In order for the intruder to perform cryptographic work using sk, she must have possession of this key; this is precluded by hypothesis, so some principal must have done cryptographic work using sk. The only principal rule which can use sk to do cryptographic work with respect to k, ck for some $ck : \mathsf{msg}$ 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 corresponding to a request message, and produce a reply message containing ck as the checksum over this request. By assumption about collision-freeness of checksums, the request that K processed must match the request described in the hypotheses.

[4] For some $C : \mathsf{client}$, $K : \mathsf{KAS}$, $k : \mathsf{shK}\, C\, K$, $sk_K : \mathsf{someSecK}$, $ck, X : \mathsf{msg}$, $Cert_C : \mathsf{CertList}$, $pk_C : \mathsf{pubK}\, C$, $T : \mathsf{TGS}$, $AK : \mathsf{shK}\, C\, T$, $n_1 : \mathsf{nonce}$, and $t_K : \mathsf{time}$.

[5] For some $t_C : \mathsf{time}$, $n_2 : \mathsf{nonce}$, and $Cert_K : \mathsf{CertList}$.

[6] For some $k_T : \mathsf{dbK}\, T$, $t_K : \mathsf{time}$.

As a corollary, if C processes a reply message containing the signed checksum of a request that C previously sent, then some KAS K fired rule $\iota_{2.1}$ as described by the theorem. This corresponds to Prop. 1 and is formally stated in [14].

7 Conclusions and Future Work

In this paper, we have unveiled a previously unknown man-in-the-middle attack against PKINIT [15], the public key extension to the popular Kerberos 5 authentication protocol [1, 2]. We discovered this attack as part of an ongoing formal analysis of Kerberos, which has previously yielded proofs of security for the core Kerberos 5 protocol [9] and its use for cross-realm authentication [10]. 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). The fixes we analyzed include the one proposed by the IETF Kerberos Working group and used in the most recent revision [15] of the PKINIT specification. Our attack was also addressed in a Microsoft security bulletin affecting several versions of Windows [5].

We intend to pursue the investigation of PKINIT in several directions. First, we would like to extend our analysis to the Diffie-Hellman mode; our preliminary examination of this mode suggests that it is immune to our attack, but we do not yet have formal proofs of its security. Second, we plan to refine our analysis by moving from the abstract Dolev-Yao model [29, 23] that we have used thus far to the more concrete *computational model*, which will allow us to reason about properties such as preimage-resistance, which are trivialized by our current formal model. We are also interested in analyzing other components of the Kerberos protocol suite.

Acknowledgments We would like to thank Ran Canetti for drawing out attention to then-possible binding issues of PKINIT in public key encryption mode. We also enjoyed the interesting and fruitful discussion with the Kerberos Working Group and especially with Jeff Hutzelman, Sam Hartman, Ken Raeburn, and the authors of the PKINIT documents.

References

1. Neuman, C., Ts'o, T.: Kerberos: An Authentication Service for Computer Networks. IEEE Communications **32** (1994) 33–38
2. Neuman, C., Yu, T., Hartman, S., Raeburn, K.: The Kerberos Network Authentication Service (V5) (2005) `http://www.ietf.org/rfc/rfc4120`.
3. Thomas, M., Vilhuber, J.: Kerberized Internet Negotiation of Keys (KINK) (2003) `http://ietfreport.isoc.org/all-ids/draft-ietf-kink-kink-06.txt`.
4. Kohl, J., Neuman, C.: The Kerberos Network Authentication Service (V5) (1993) `http://www.ietf.org/rfc/rfc1510`.
5. Microsoft: Security Bulletin MS05-042. `http://www.microsoft.com/technet/security/bulletin/MS05-042.mspx` (2005)
6. Strasser, M., Steffen, A.: Kerberos PKINIT Implementation for Unix Clients. Technical report, Zurich University of Applied Sciences Winterthur (2002)

7. CERT: Vulnerability Note 477341. (http://www.kb.cert.org/vuls/id/477341)
8. Yu, T., Hartman, S., Raeburn, K.: The perils of unauthenticated encryption: Kerberos version 4. In: Proc. NDSS'04. (2004) http://www.isoc.org/isoc/conferences/ndss/04/proceedings/Papers/Yu.pdf.
9. Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A.: An Analysis of Some Properties of Kerberos 5 Using MSR. In: Proc. CSFW'02. (2002)
10. Cervesato, I., Jaggard, A.D., Scedrov, A., Walstad, C.: Specifying Kerberos 5 Cross-Realm Authentication. In: Proc. WITS'05, ACM Digital Lib. (2005) 12–26
11. Kemmerer, R., Meadows, C., Millen, J.: Three systems for cryptographic protocol analysis. J. Cryptology **7** (1994) 79–130
12. Meadows, C.: Analysis of the internet key exchange protocol using the nrl protocol analyzer. In: Proc. IEEE Symp. Security and Privacy. (1999) 216–231
13. Mitchell, J.C., Shmatikov, V., Stern, U.: Finite-State Analysis of SSL 3.0. In: Proc. 7th USENIX Security Symp. (1998) 201–216
14. Cervesato, I., Jaggard, A.D., Scedrov, A., Tsay, J.K., Walstad, C.: Attacking Public-Key Kerberos. Draft available at http://www.deductivesolutions.com:8888/~iliano/PKINIT/TR.pdf (2005)
15. IETF: Public Key Cryptography for Initial Authentication in Kerberos (1996–2005) Sequence of Internet drafts available from http://tools.ietf.org/wg/krb-wg/draft-ietf-cat-kerberos-pk-init/.
16. De Clercq, J., Balladelli, M.: Windows 2000 authentication. http://www.windowsitlibrary.com/Content/617/06/6.html (2001) Digital Press.
17. Cable Television Laboratories, Inc.: PacketCable Security Specification (2004) Technical document PKT-SP-SEC-I11-040730.
18. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks. SIAM J. Computing **17** (1988) 281–308
19. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Designs, Codes and Cryptography **2** (1992) 107–125
20. Canetti, R., Krawczyk, H.: Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In: Proc. CRYPTO'02, Springer LNCS 2442 (2002) 143–161
21. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). http://www.ietf.org/rfc/rfc2409 (1998)
22. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In: Proc. TACAS'96, Springer LNCS 1055 (1996) 147–166
23. Needham, R., Schroeder, M.: Using Encryption for Authentication in Large Networks of Computers. Comm. ACM **21** (1978) 993–999
24. Abadi, M., Needham, R.: Prudent Engineering Practice for Cryptographic Protocols. IEEE Trans. Software Eng. **22** (1996) 6–15
25. Raeburn, K.: Encryption and Checksum Specifications for Kerberos 5. http://www.ietf.org/rfc/rfc3961.txt (2005)
26. Cervesato, I.: Typed MSR: Syntax and Examples. In: Proc. MMM'01. Springer LNCS 2052 (2001)
27. Durgin, N.A., Lincoln, P., Mitchell, J., Scedrov, A.: Multiset Rewriting and the Complexity of Bounded Security Protocols. J. Comp. Security **12** (2004) 247–311
28. Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A.: Confidentiality and Authentication in Kerberos 5. Technical report (2004) UPenn CIS Technical Report MS-CIS-04-04: ftp://ftp.cis.upenn.edu/pub/papers/scedrov/ms-cis-04-04.pdf.
29. Dolev, D., Yao, A.: On the security of public-key protocols. IEEE Trans. Info. Theory **2** (1983) 198–208