7659, England, UK
Journal of Information and Computing Science
Vol. 2, No. 3, 2007, *pp*. 163-171

Analyzing Authentication in Kerberos-5 Using Distributed Temporal Protocol Logic

Shahabuddin Muhammad [+], Zeeshan Furqan and Ratan K. Guha

School of Electrical Engineering and Computer Science, University of Central Florida, USA

(*Received 3 January 2007, accepted 1 March 2007*)

Abstract. Recently a Distributed Temporal Protocol Logic has been devised to capture reasoning in the distributed environment of security protocols. Elsewhere we have constructed a proof-based verification framework using distributed temporal protocol logic to verify the authentication property of security protocols. In this paper, we apply our verification framework to a well-known protocol. In particular, we analyze the authentication property of the public-key extension of Kerberos-5 protocol. We demonstrate how we are able to identify a subtle design flaw in the protocol. This results into showing the applicability of our framework as well as demonstrating the ease with which distributed temporal protocol logic can be used to analyze authentication protocols.

Keywords: Kerberos, distributed temporal protocol logic, formal verification, authentication.

1. Introduction

Before running critical applications, such as e-commerce, in a distributed environment, one needs to be assured of the identities of the parties involved in the communication. Security protocols are designed to achieve this task. A security protocol is a sequence of messages between two or more parties in which encryption is used to provide authentication or to distribute cryptographic keys for new conversation [1]. However, the mere existence of security protocols in a distributed application is not enough to guarantee the security of the data. Designing security protocols is itself an error-prone process. History has shown that many carefully designed security protocols were later found out to have subtle flaws [2]. This situation led the researchers to formalize the verification of security protocols. Some of the considerable contributions in the formal verification of security protocols include logic-based techniques [3], [4], [5], [6], process-algebra [7], [8], theorem prover [9], graph-theoretic approach [10], and model-checkers [11], [12], [13], [14], [15], [16], [17]. A good survey on the formal verification of security protocols can be found in [18]. Once a protocol is designed, it is rigorously analyzed using any of the formal techniques to ensure that the protocol achieves what it is intended to achieve.

Recently a Distributed Temporal Protocol Logic (henceforth referred to as DTPL) has been devised to model reasoning in the distributed environment of security protocols in [19]. DTPL is an extension of the distributed temporal logic DTL of [20]. The distinguishing characteristic of DTPL is its capability to be used as a meta-level tool for comparative analysis of security protocol models and properties [21]. Benefiting from this property of DTPL, we have constructed a framework in [22] which can be used to verify authentication property of security protocols in a proof-based setting.

In this paper, we use our proposed framework of [22] to analyze the authentication property in public-key extension of the well-known authentication protocol, Kerberos-5 [23], [24]. Kerberos-5 is a widely deployed protocol designed to authenticate clients to multiple networked services using a single login. PKINIT [25] is an extension of Kerberos-5 in which public-key is used in the first pass of the protocol. We analyze authentication property in the recent version of PKINIT, PKINIT-26. This amounts towards showing the applicability of our framework. Moreover, we point out some sources of errors when we try to prove the authentication property of the protocol. Generally a proof-based method generates a proof only if a protocol is correct. However, we demonstrate how a proof-based framework can be used as a guide to discover design flaws even in case of the absence of a proof.

[+] Corresponding author. Tel.: +1-407-823 3364; fax: +1-407-823 5914.
E-mail address: muhammad@cs.ucf.edu.

This paper is organized as follows. In Section 2 we briefly describe the framework of [22]. In particular, we introduce the distributed temporal protocol logic and list all the axioms of the framework in that section. After describing the well-known authentication protocol PKINIT, Section 3 applies the framework to analyze the authentication property of the protocol. We conclude in Section 4.

2. The Logic-based Verification Framework

We briefly introduce the framework of [22] which is based on the distributed temporal protocol logic of [19]. A security protocol is a sequence of messages sent and received between two or more principals. Each principal is equipped with some actions and propositions. Table 1 lists the set of actions and propositions associated with a principal.

Table 1: The set of actions and propositions associated with a principal of a protocol.

Actions	*send*(*M*,*A*)	It represents sending a message *M* to a principal *A*.
	rec(*M*)	A principal receives a message *M*.
	nonce(*N*)	A principal generates a random number *N*.
	key(*K*)	A principal generates a key *K*.
Propositions	*knows*(*M*)	It represents a principal's knowledge of a message *M*.
	fresh(*M*)	It states that a message *M* is fresh.
	controls(φ)	It represents a principal's authority over any formula φ.
	$A \xleftrightarrow{K} B$	It represents that principals *A* and *B* share a secret-key *K*.
	$A \mapsto K$	It represents that *K* is the public-key of principal *A*.

In addition to the actions and propositions, DTPL defines a rich set of operators to capture various temporal activities of a principal in the distributed environment. These operators precisely capture the timings of the actions of a principal at various *configurations*. A configuration at time *i* represented as ξ_i is defined by a set of all the events of a principal up to time *i*. A principal's initial configuration is defined to be an empty set $\xi_i = \{\Phi\}$. Each action *a* (e.g, *send, rec, nonce, key*) of a principal changes its configuration from ξ_i to ξ_{i+1} such that $\xi_{i+1} = \xi_i \cup \{a\}$. Temporal operators can be applied on a principal's actions and propositions at a configuration. For instance, if a principal *A* sends a key right after generating it, then the action *key*(*K*) at configuration ξ_i temporally precedes the action *send*(*M, B*) at ξ_{i+1} such that we can specify the sequence of *A*'s actions at ξ_i as *key*(*K*), X *send*(*K, B*) where X is a temporal operator representing 'next'. Similarly, other temporal operators are also defined that capture various past time and future activities. We list these temporal operators in Table 2.

Table 2: The temporal operators in the DTPL.

Operator	Meaning	Operator	Meaning
X φ	next	†	in the end
Y φ	previous	*	in the beginning
F φ	sometime in the future	F_0 φ	now or sometime in the future
P φ	sometime in the past	P_0 φ	now or sometime in the past
G φ	always in the future	G_0 φ	now and always in the future
H φ	always in the past	H_0 φ	now and always in the past

2.1. Axioms of the framework

The framework of [22] comprises of a set of axioms capturing the capabilities of principals of the network. In particular, these axioms capture how a principal acquires knowledge through communication events and by applying encryption and decryption operations, the notion of freshness of a message, a principal's authority over generating good session keys, finding the originator of a received message, and so on. In the following, $@_A[\varphi]$ represents that a formula φ is true at principal *A*. We represent the communication and knowledge axioms of the framework as follows.

(C1) $@_A[rec(X_1 \ldots X_n) \Rightarrow rec(X_i)]$ for $i = 1,\ldots,n$.

DOI https://doi.org/2024-JICS-22794 | Generated on 2025-04-20 19:33:02

(C2) $@_A[rec(\{M\}_{k_{AS}}) \wedge knows(A \xleftrightarrow{k_{AS}} S) \Rightarrow rec(M)]$

(C3) $@_A[rec(\{M\}_k) \wedge knows(A \mapsto k) \Rightarrow rec(M)]$

(C4) $@_A[rec(\{M\}_{k^{-1}}) \wedge knows(B \mapsto k) \Rightarrow rec(M)]$

(C5) $@_A[rec(M) \Rightarrow knows(M)]$

(K1) $@_A[knows(X_1) \wedge \ldots \wedge knows(X_n) \Rightarrow knows(F(X_1,\ldots,X_n))]$

(K2) $@_A[knows(M) \Rightarrow \mathsf{G}_0\ knows(M)]$

(K3) $@_A[nonce(N) \Rightarrow knows(N)]$

(K4) $@_A[key(K) \Rightarrow knows(K)]$

The first axiom C1 states that concatenates of a received message is also deemed as a received message. C2 through C4 state that given the proper key, the contents of an encrypted received message are also considered to be received. C5 simply states that a principal knows its received message. K1 states that a principal knows any computable function *F* (e.g., encryption/signing by a known key, concatenation) of its known messages. K2 simply states that a principal does not forget its known messages. K3 and K4 state that a principal knows its generated messages (through actions *nonce* and *key*).

The following axioms capture the freshness of a message.

(F1) $@_A[nonce(N) \Rightarrow fresh(N)]$

(F2) $@_A[fresh(X) \Rightarrow \mathsf{X}\ fresh(X)]$

(F3) $@_A[fresh(X) \Rightarrow fresh(M_X)]$

The above axioms state that the action *nonce* generates a fresh nonce (F1), a fresh message remains fresh for the current run of a protocol (F2) and any message containing a fresh term is also fresh (F3). M_X in F3 represents either a message of the form …*X*… or $\{\ldots X\ldots\}_K$. That is, M_X is the result of applying some operations (such as encryption, signing, concatenation) on *X*.

The following axiom captures the authority of a principal for generating keys.

(J1) $@_S[controls(\varphi_k) \wedge send(M_k, A)] \Rightarrow \bigvee_{P \in \mathrm{Princ}} @_P[\mathsf{F}\ knows(\varphi_k)]$

That is, if a principal *S* is known to control a formula for generating keys φ_k and he sends a message containing the key *k* then the receiver knows φ_k to be true. Generally *S* represents a server that generates keys to be used between two principals.

The following axioms capture the notion of source association in which a principal investigates the source of a received message. First, we define the notion of origination of a message.

(O1) $@_A[send(M_N, B) \wedge \mathsf{H}(\neg send(M'_N, C) \wedge \neg rec(M'_N)) \Leftrightarrow Orig(M_N)]$

The above axiom states that if a principal sends a term in a message such that he never communicated that term in any message in the past then he originates the term in its sending message.

The following axiom finds the originator of a received message in symmetric-key cryptography. Since in symmetric-key cryptography, a key is assumed to be a principal's safe secret, encrypting a message under symmetric-key ensures the possession of the key, and hence the origination of the message by a principal having that key.

(O2) $@_A[knows(P \xleftrightarrow{k} Q) \wedge rec(\{X\}_k)] \Rightarrow \bigvee_{\mathrm{B} \in \{\mathrm{P},\mathrm{Q}\}} @_B[\mathsf{P}\ Orig(\{X\}_k)]$

Similarly, in asymmetric-key cryptography, a signed message originates from a principal who has access to the private-key with which the message was signed. That is,

(O3) $@_A[knows(B \mapsto k) \wedge rec(\{X\}_{k^{-1}})] \Rightarrow @_B[\mathsf{P}\ Orig(\{X\}_{k^{-1}})]$

Since public-key of a principal is either assumed to be publicly available or it can be easily obtained, reception of a message encrypted by the public-key of a principal does not provide any useful information about the originator of the message. Therefore, the security protocols adopt a challenge-response method in order to determine the originator of a received message. In this method, a principal originates a secret

challenge message and waits for the response. The secret challenge message is constructed such that only the intended principal could decrypt that challenge and generate the proper reply to the challenge. Reception of the reply message guarantees that the responder must have received the challenge message.

(O4) $@_A[(\neg send(M'_N, C)\,\mathsf{S}(Orig(\{M_N\}_k) \wedge fresh(N))) \wedge rec(M''_N) \wedge knows(B \mapsto k)]$

$\Rightarrow @_B[\mathsf{P}(Orig(M'''_N) \wedge \mathsf{P}\, knows(\{M_N\}_k))]$

where, N must exist in M''_N in a form other than $\{M_N\}_k$. Moreover, N also exists in M'''_N in a form other than $\{M_N\}_k$. In the above, the intuition is that since only B has the decryption key to discover N from $\{M_N\}_k$ and N is originated uniquely (so that no other principal knows N except the one who originated it), reception of any message in which N occurs in any form other than $\{M_N\}_k$ confirms that $\{M_N\}_k$ has been decrypted and the information (N) has been released by B.

In addition to the above axioms, the framework uses the modus ponens as its inference rule given below. This rule permits the derivation of ψ from the truth of φ and $\varphi \Rightarrow \psi$.

(MP) $\varphi \wedge (\varphi \Rightarrow \psi) \Rightarrow \psi$

So far, we have presented the framework of [22] that can be used to analyze authentication protocols in a proof-based environment. Next, we present how to apply the framework in order to analyze authentication in a well-known protocol.

3. Verifying Authentication in the Public-Key Kerberos PKINIT

PKINIT is the public-key extension of Kerberos 5 authentication protocol. First we briefly overview Kerberos 5 and give motivation behind PKINIT.

3.1. The Protocol Description

Kerberos [23], [24] is a widely deployed protocol designed to authenticate clients to multiple networked services using a single login. Messages in the Kerberos contain various encrypted tickets that are used to authenticate a user to the desired service. The recent version of Kerberos, Kerberos 5, is available for all major operating systems. A standard run of Kerberos 5 consists of three phases. A client C first obtains a *ticket granting ticket* (T_{tgt}) from *kerberos authentication server* (*KAS*) K. C then presents *TGT* to *ticket granting server* (*TGS*) T and obtains a *service ticket* (T_{st}). Finally C uses the service ticket to authenticate itself to an *application server* S. Kerberos message exchanges are depicted in Fig. 1. For simplicity, we omit some of the message ingredients from the protocol that essentially do not affect the analysis at hand.

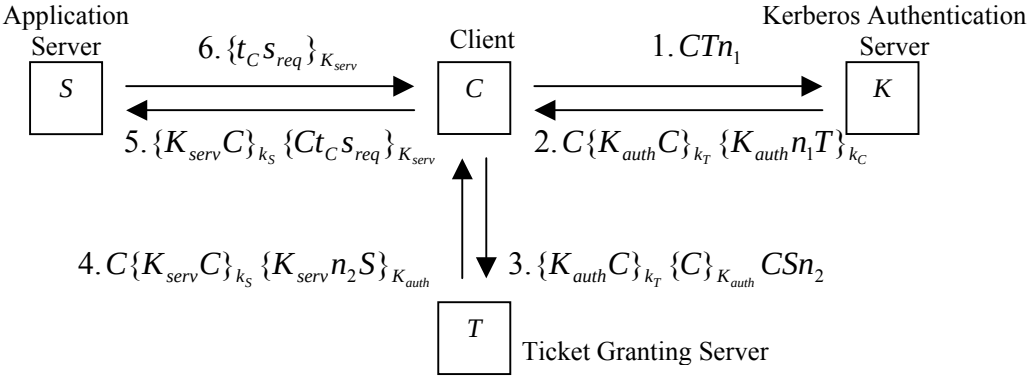

Figure 1: Message exchanges between the client C and the servers K, T, and S in the Kerberos protocol.

In Figure 1, $\{K_{auth}C\}_{k_T}$ = ticket granting ticket T_{tgt} (in message 2) and $\{K_{serv}C\}_{k_S}$ = service ticket T_{st} (in message 4) Moreover, k_C, k_T, k_S are the secret keys of C, T, and S respectively. n_1, n_2 are two distinct nonces and K_{auth} and K_{serv} are the authentication-key to be shared between C and T and the service-key to be shared between C and S respectively.

Notice that upon receiving each message, the client C creates an authenticator to be used for the next message exchange. The client uses $\{C\}_{K_{auth}}$ and $\{Ct_C s_{req}\}_{K_{serv}}$ as authenticators in the third and fifth message exchanges. The last message exchange $\{t_C s_{req}\}_{K_{serv}}$ is an acknowledgment message from the server and is

optional. PKINIT [25] is an extension to the basic protocol in which public-key authentication is used in the first pass of the protocol. The next two passes in PKINIT remain the same as that in Kerberos 5. In Kerberos 5, *KAS* derives the long-term shared secret k_C from the user's password. This leaves *KAS* vulnerable to attacks where even read-only access to *KAS* may result in the compromised secret keys of the clients. With the introduction of public-key cryptography, PKINIT does not need shared secret between a client and *KAS*, hence avoids the possibility of compromised long-term shared secrets. Since public-key cryptography is computationally expensive operation, PKINIT uses it only in its first pass of the protocol. However, it complicates the overall protocol since the rest of the passes use traditional secret-key cryptography. An abstract view of the first round of message exchanges in PKINIT can be represented as shown in Figure 2.

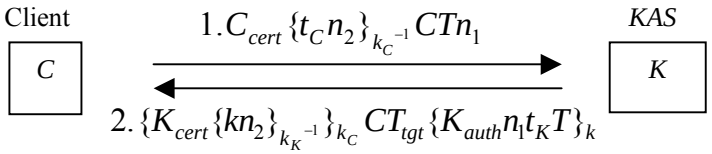

Figure 2: The first round of message exchanges between *C* and the kerberos authentication server *K* in PKINIT.

In its first pass, the client forwards his certificate C_{cert} along with a timestamp t_C and a nonce n_2 signed by his private-key k_C^{-1}. Client's certificate provides the information about client's public-key to *KAS* and the signed message affirms that it has been originated at the client. Client also concatenates its id *C*, the ticket granting server's id *T* and a nonce n_1 in the first message. *KAS* replies the client back with its certificate K_{cert} and a signed message containing a freshly generated symmetric-key *k* and the client's nonce n_2, all encrypted with the public-key of the client k_C. Both certificates C_{cert} and K_{cert} are provided by public-key infrastructure (PKI) that ensures binding of public-keys to the users. The reply message also contains the ticket T_{tgt} and a message containing the authentication key K_{auth}, nonce n_1, timestamp t_K, and *TGS* id *T*, all encrypted by the fresh key *k*.

3.2. Analyzing the Protocol

We briefly sketch the analysis of the first pass of PKINIT that uses public-key cryptography. We apply the aforementioned axioms of the framework in order to investigate the messages from each principal's perspective. In other words, the initiator *C* investigates its sent and received messages in order to find out the true responder *K* of the protocol. Similarly, the responder tries to find out the true initiator of the protocol by investigating its message. We assume that only principals *C* and *K* possess the secret keys k_C^{-1} and k_K^{-1} respectively and the nonces are distinct ($n_1 \neq n_2$) and uniquely originating. That is, a nonce can not be originated by more than one principal. Notice that n_2 serves as an open challenge to *KAS* in the first message. Client waits for the right response before proceeding to the second round of the protocol. That is, it waits for a signed message of the form $\{k\ n_2\}_{k_K^{-1}}$. The *C*'s and *K*'s runs of the first phase of the protocol are depicted in terms of DTPL in Figure 3 and Figure 4 respectively.

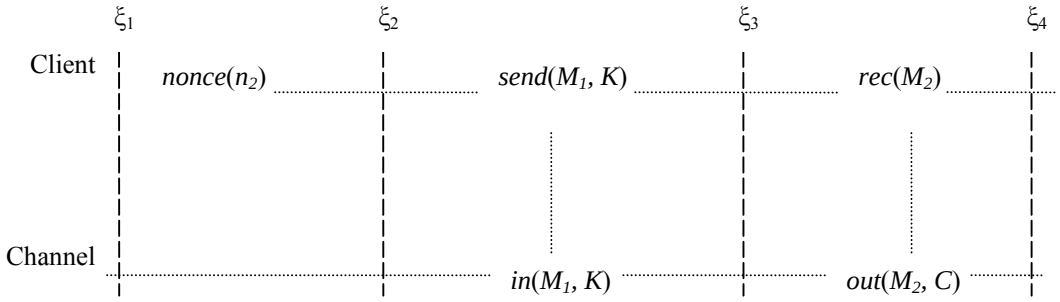

Figure 3: First pass of the client's run in PKINIT using DTPL. The client *C* sends its challenge n_2 and expects a message containing n_2 singed by the private-key of K.

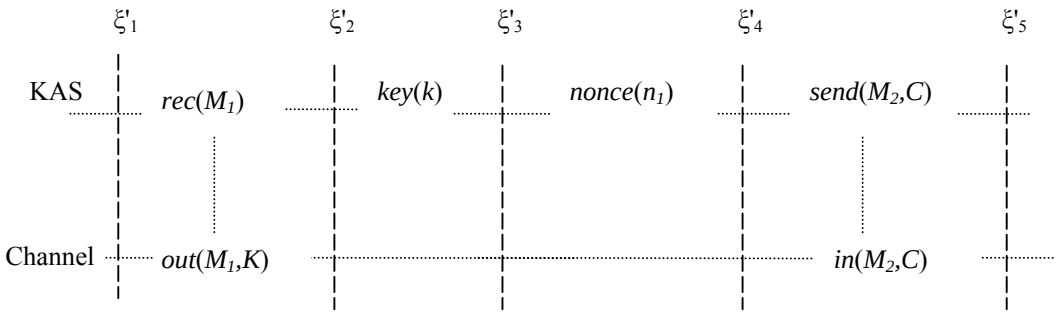

Figure 4: First pass of the *KAS*'s run in PKINIT using DTPL. The server *K* responds to the client's challenge by sending a singed message containing n_2 along with a session key *k*.

Corresponding to the above figures, the client's and server's sequence of messages can be represented in terms of DTPL as follows.

1. $@_C[rec(M_2) \wedge \mathsf{P}\,(send(M_1, K) \wedge \mathsf{P}\, nonce(n_2))]$
2. $@_K[send(M_2, C) \wedge \mathsf{P}\,(nonce(n_1) \wedge \mathsf{P}\,(key(k) \wedge \mathsf{P}\, rec(M_1)))]$

Where, $M_1 = C_{cert}\{t_C\, n_2\}_{k_C^{-1}} CTn_1$ and $M_2 = \{K_{cert}\{k\, n_2\}_{k_K^{-1}}\}_{k_C} CT_{tgt}\{K_{auth}\, n_1\, t_K T\}_k$. Notice the vertical dashed lines in the figures indicating various configurations in the run of a principal. Moreover, vertical dotted lines represent communication points between a principal and the distributed *channel*. DTPL defines a distributed channel in which a principal's sending and receiving actions are directly linked with the channel's *in*(*M*,*A*) and *out*(*M*,*A*) actions respectively. Since our framework focuses solely on the actions of the principals of a protocol, we ignore the channel in the figures. Also notice that each action (*send*, *rec*, *nonce*, *key*) of a principal changes its configuration from ξ_i to ξ_{i+1}.

The initial set of assumptions of the principals is as follows:

$@_C[* \Rightarrow knows(K \mapsto k_K)]$, $@_K[* \Rightarrow knows(C \mapsto k_C)]$, $@_C[* \Rightarrow knows(C \mapsto k_C)]$

Where, * (see Table 2) captures initial configurations ξ_1 and ξ'_1 for *C* and *K* respectively. More assumptions can be written, such as *K* knows its own public-key, which we do not need in the present analysis. These assumptions hold true as public-key certificates can be easily obtained upon request. Knowledge is treated in DTPL as non-decreasing as formulated by K2. We apply MP and K2 and use the above assumptions to get the following results at any configuration.

A1. $@_C[knows(K \mapsto k_K)]$

A2. $@_K[knows(C \mapsto k_C)]$

A3. $@_C[knows(C \mapsto k_C)]$

The server *K* investigates it messages and concludes the following.

3. $@_K[rec(\{t_C n_2\}_{k_C^{-1}})]$ by 2, C1 and MP at ξ'_2.
4. $@_K[rec(t_C n_2)]$ by 3, A2, C4 and MP at ξ'_2.
5. $@_K[rec(n_2)]$ by 4, C1 and MP at ξ'_2.
6. $@_K[knows(n_2)]$ by 5, C5 and MP at ξ'_2.
7. $@_C[\mathsf{P}\, Orig(\{t_C n_2\}_{k_C^{-1}})]$ by 3, O3, A2 and MP at ξ'_2.

Therefore, the kerberos authentication server *K* knows that *C* initiated the session and originated the signed component sometime before ξ'_2. Now the client *C* investigates it messages in the following.

8. $@_C[rec(\{K_{cert}\{kn_2\}_{k_K^{-1}}\}_{k_C})]$ by 1, C1 and MP at ξ_4.
9. $@_C[rec(K_{cert}\{kn_2\}_{k_K^{-1}})]$ by 8, C3, A3 and MP at ξ_4.
10. $@_C[rec(\{kn_2\}_{k_K^{-1}})]$ by 9, C1 and MP at ξ_4.

11. $@_C[rec(kn_2)]$ by 10, C4, A1 and MP at ξ_4.

12. $@_C[rec(n_2)]$ by 11, C1 and MP at ξ_4.

The client has received back its nonce n_2 which it generated as a challenge for *K*. Furthermore,

13. $@_K[\mathsf{P}\, Orig(\{kn_2\}_{k_K^{-1}})]$ by 10, O3, A1 and MP at ξ_4.

The client concludes that *K* has originated the signed message sometime before ξ_4. In addition to the above, the client carries out the following analysis based on freshness and concludes that it has been involved in the current run of the protocol.

14. $@_C[\, fresh(n_2)]$ by 1, F1 and MP at ξ_2.

15. $@_C[\, fresh(n_2)]$ by 14, F2 and MP at $\xi \supset \xi_2$.

16. $@_C[\, fresh(M_2)]$ by 15, F3 and MP at ξ_4.

That is, *K* not only originated the signed component in M_2, but it did so recently. The client *C* provides assurance in the origination of its fresh nonce n_2 by signing it with its secret key k_C^{-1}. The presence of n_2 in the received signed message $\{k\ n_2\}_{k_K^{-1}}$ ensures the origination of the message and hence the reception of n_2 at *KAS*. Other than that, the client does not provide any assurance in the rest of the message bindings with the legitimate *KAS*. This results in the lack of assurance in some crucial parameters from client's view of *kerberos authentication server*. Apart from the signed message in M_2, $\{kn_2\}_{k_K^{-1}}$, binding it with the *KAS*, public-key encryption in $\{K_{cert}\{kn_2\}_{k_K^{-1}}\}_{k_C}$ using k_C and symmetric-key encryption in $\{K_{auth}n_1t_KT\}_k$ using *k* do not bind the messages with its recipient - the client *C*. That is, simply from M_2 it can not be deduced that the server *K* is aware of the client *C* for this session of the protocol. This is due to the fact that n_2 could be easily obtained from M_1 and any principal could encrypt a message with the public-key of *C* in M_2. Moreover, a principal could simply forward $\{K_{auth}\ n_1t_KT\}_k$ after receiving it first from *KAS*.

3.3. Attack on the Protocol

The above-mentioned lack of assurance in parameter *C* in the message component $\{kn_2\}_{k_K^{-1}}$ results in the man-in-the-middle attack. The authors in [26] were the first to mention this attack on PKINIT-26. The attack, somewhat similar to that on the Needham-Schroeder public-key protocol in [17], exploits the above-mentioned weakness in the protocol in which ids of the principals are not tightly bound with the messages. Fig. 5 shows how it works.

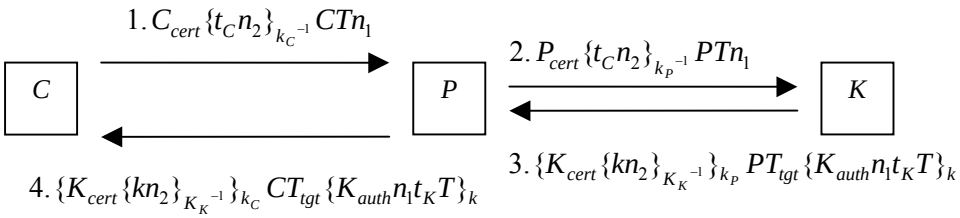

Fig. 5: Attack on PKINIT in which a penetrator *P* plays man-in-the-middle between *C* and *K*.

Observe that the penetrator *P* captures *C*'s message and makes some changes such that it appears to *KAS* as if it was generated by *P*. Given that *P* is a legitimate principal of the network, *KAS* follows the standard protocol step and comes up with *k*, K_{auth} and t_K. The reply from *KAS* is intended for *P* but the reply message does not contain any binding to ensure *KAS*'s perception of the initiator. Apart from the message component $\{k\ n_2\}_{K_K^{-1}}$, rest of the message can be constructed for any legitimate principal. Notice that T_{tgt} contains the id of the initiator as perceived by *KAS* (*P* in this case) but *C* can not decrypt T_{tgt} and never learns this information. This attack in the initial phase of the protocol propagates to the remaining two phases in which the client contacts *TGS* and the server. Every time the client initiates a request with one of the servers, *P* intercepts the messages and forges them such that the servers believe the messages to be originated by the penetrator *P*. In particular, *P*'s possession of K_{auth} (and hence K_{serv}) makes it possible to replace client's authenticators with that of the penetrator's authenticators. Client's inability to read T_{tgt} and T_{st} results in the successful completion of the protocol run.

4. Conclusion

We have applied a logic-based formal framework to analyze the authentication property in the public-key extension of Kerberos-5 protocol. We have shown that how we were able to capture a subtle design flaw in the protocol using distributed temporal protocol logic. The distinguishing characteristic of applying distributed temporal protocol logic is its fine representation of different temporal activities occurring in a distributed environment. This results into a clear understanding of a protocol run that makes it easy to apply logical rules of the framework at various configuration points. In addition to showing the applicability of the logical framework, we have demonstrated how a proof-based method can be used as a guide to discover flaws in security protocols.

5. Acknowledgements

This work was partially supported by ARO under grant DAAD 9-01-1-0502. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agencies or the University of Central Florida.

6. References

[1] R. M. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*. 1978, **21**(12): 993-999.

[2] S. Muhammad, Z. Furqan, and R. K. Guha. Understanding the Intruder through Attacks on Cryptographic Protocols. *Proceedings of the 44th ACM Southeast Conference* (ACMSE2006). Mar. 2006, 667-672.

[3] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*. 1990, **8**(1): 18-36.

[4] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. *Proceedings of the IEEE Symposium on Research in Security and Privacy*. May 1990, 234-248.

[5] M. Abadi and M. Tuttle. A Semantics for a Logic of Authentication. *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*. 1991, 201-216.

[6] P. Syverson and P. V. Oorschot. On Unifying Some Cryptographic Protocol Logics. *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*. 1994, 14-28.

[7] C. Bodei, P. Degano, R. Focardi, and C. Priami. Primitives for Authentication in Process Algebras. *Theoretical Computer Science*. 2002, **283**(2): 271-304.

[8] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Addison Wesley. 2000.

[9] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*. 1998, **6**: 85-128.

[10] J. D. Guttman and F. J. T. F'abrega. Authentication Tests and the Structure of Bundles. *Theoretical Computer Science*. 2002, **283**: 333-380.

[11] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. May 1997, 141-151.

[12] D. Basin, S. Modersheim, and L. Vigano. OFCM: A Symbolic Model Checker for Security Protocols. *International Journal of Information Security*. 2005, **4**(3): 181-208.

[13] J. K. Millen and V. Shmatikov. Constraint Solving for Bounded-Process Cryptographic Protocol Analysis. *Proceedings of the 8th ACM Conference on Computer and communications security*. 2001, 166-175.

[14] R. Kemmerer. Using Formal Methods to Analyze Encryption Protocols. *IEEE Journal on Selected Areas in Communications*. 1989, **7**(4): 448-457.

[15] D. Longley and S. Rigby. An Automatic Search for Security Flaws in Key Management Schemes. *Computers and Security*. 1992, **11**(1): 75-90.

[16] C. A. Meadows. Applying Formal Methods to the Analysis of a Key Management Protocol. *Journal of Computer Security*. 1992, **1**(1): 5-36.

[17] G. Lowe. Breaking And Fixing the Needham-Schroeder Public-Key Protocol Using FDR, Tools and Algorithms for the Construction and Analysis of Systems. In: T. Margaria and B. Steffen (eds.). *2nd International Workshop TACAS'96*, *LNCS series 1055*. Springer Verlag, 1996, 147-166.

[18] C. Meadows. Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends. *IEEE Journal on Selected Areas in Communications*. 2003, **21**(1): 44-54.

[19] C. Caleiro, L. Vigano, and D. Basin. Metareasoning About Security Protocols Using Distributed Temporal Logic.

Electronic Notes in Theoretical Computer Science. 125(1): 67-89, 2005.

[20] H-D. Ehrich and C. Caleiro. Specifying Communication in Distributed Information Systems. *Acta Informatica*. 2000, **36**(8): 591-616.

[21] C. Caleiro, L. Vigano, and D. Basin. Relating Strand Spaces and Distributed Temporal Logic for Security Protocol Analysis. *Logic Journal of IGPL*. 2005, **13**(6): 637-663.

[22] S. Muhammad, Z. Furqan, and R. K. Guha. A Logic-Based Framework for Verifying Authentication Protocols. *Submitted in International Journal of Internet Technology and Secured Transactions*.

[23] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network *Authentication Service* (v5). 2005. Available online. http://www.ietf.org/rfc/rfc4120 2005.

[24] C. Neuman and T. Ts'o, Kerberos. An Authentication Service for Computer Networks. *IEEE Communications*. 1994, **32**(9): 33-38.

[25] IETF. Public Key Cryptography for Initial Authentication in Kerberos. 1996-2005, *Sequence of Internet Drafts*. Available online. http://tools.ietf.org/wg/krb-wg/draft-ietf-cat-kerberos-pk-init/.

[26] I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and Fixing Public-Key Kerberos. *Sixth Workshop on Issues in the Theory of Security - WITS'06*. Mar. 2006.