Nonmonotonic Cryptographic Protocols
CITI Technical Report 93-9

A. D. Rubin
rubin@citi.umich.edu

Center for Information Technology Integration
Dept. of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48103-4943

November 18, 1993

Abstract

This paper presents a new method for specifying and analyzing cryptographic protocols. Our method offers several advantages over previous approaches.

Our technique is the first to allow reasoning about nonmonotonic protocols. These protocols are needed for systems that rely on the deletion of information. There is no idealization step in specifying protocols; we specify at a level that is close to the actual implementation. This avoids errors that might otherwise render a specification that passes the analysis, useless in practice.

In our method, knowledge and belief sets for each principal are modified via actions and inference rules. Every message is considered to be broadcast, and we introduce the `update` function to maintain global knowledge. We show how our method uncovers the known flaw in the Needham and Schroeder protocol [11], and that the revision by the same authors [12] does not contain this flaw. We also show that our method correctly handles protocols that are trivially insecure, such as Nessett's noted example. [13]

We then apply our method to our *khat* protocol [14]. The analysis reveals a serious, previously undiscovered flaw in our nonmonotonic protocol for long-running jobs; one that seems obvious in hindsight, but escaped the attention of the authors and over 300 USENIX conference attendees. In addition, our analysis reveals a previously unknown vulnerability in phase II of *khat*. These are stunning confirmations of the importance of tools for analyzing cryptographic protocols.

1 Introduction

In computer networks, communicating parties must share a set of rules describing the messages they will send and receive. These rules, or protocols, are the foundation on which modern networks are built. As protocols are necessary to establish any useful communication, standard sets of rules are published and made widely available. This allows users all over the world to communicate with each other and share information on networks such as the Internet.

Unfortunately, the availability and widespread knowledge of communication protocols has also facilitated the malicious interference of active intruders on the network. To combat this, cryptographic protocols that rely on the encryption of data were developed. It is widely accepted that the security of data in networks should rely on the underlying cryptographic technology, and that the protocols should be open and available. [18] However, many protocols have been found to be vulnerable to attacks that do not require breaking the encryption, but instead manipulate the messages in the protocol to gain some advantage. The advantages potentially gained by an attacker range from the compromise of confidentiality to the ability to impersonate another user.

Analysis techniques have been developed to help discover flaws in protocols before they are trusted. Flaws were discovered in such well known protocols as the Needham and Schroeders key distribution protocol [5] and the CCITT X.509 protocol [2]. The BAN logic of Burrows, Abadi, and Needham [2] and its descendents [3, 4, 6, 7, 9, 15], have been pivotal in the ability to use knowledge and belief in the analysis of cryptographic protocols to discover flaws.

All of the logics developed to date reason monotonically. That is, once something is known, it is always known. This fact has been a fundamental obstacle in providing a complete logic because negation is missing. This means that there are valid formulas that cannot be derived. There have been two attempts to remedy this. Abadi and Tuttle [1] provide a semantics for the BAN logic that includes a new construct for negation. Moser [10] provides a nonmonotonic logic of belief. However, neither of these deal with nonmonotonicity of *knowledge*. The difference between nonmonotonicity of knowledge and nonmonotonicity of belief is discussed in Section 3.

At the Center for Information Technology Integration (CITI), we are researching a new method for analyzing protocols. This is the first method proposed for reasoning nonmonotonically about knowledge in cryptographic protocols. Our approach is a variation on the protocol specification techniques of Woo and Lam [19] where each principal's actions are defined separately. In addition, we do not require protocol *idealization*, and thus avoid many of its associated pitfalls as described by Mao and Boyd. [9] The notation we use is based on the original BAN logic [2], and we use a similar reasoning mechanism.

We show how our new method can be used to specify and analyze the Needham and Schroeder protocol. We then use it to analyze the *khat* protocol [14], which uses nonmonotonicity of knowledge, and we show that no other analysis techniques can be used to analyze this protocol. Finally, our method is used to uncover the flaw in a famous protocol presented by Nessett [13] that he used to demonstrate a weakness in BAN logic.

2 Protocol Specification in a Distributed System

A typical protocol specification consists of a list of messages between principals. For example, the Needham and Schroeder protocol specification [11] can be seen in Figure 1. $A \rightarrow B : M$ means that principal A sends message M to principal B. A protocol designer thus specifies a protocol by listing the messages principals send to each other.

Although such a specification is intuitive, it does not represent the way a protocol is implemented in a distributed system. In a distributed network, each principal need be aware only of his potential role in a protocol. For example, in the Needham and Schroeder protocol presented above, principal S is not concerned with messages 3, 4 and 5. In addition, there are calculations and actions (such as decryp-

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Figure 1: **The Needham and Schroeder protocol specification.** Protocols are specified by a principal name, followed by an arrow and another principal name, followed by a message.

tion and encryption) performed by principals during the run of a protocol that are not captured by the specification.

The need to idealize protocols before analyzing them is a weakness in this form of protocol specification. This paper proposes a new method for specifying protocols that conform to the distributed system model, one that does not require idealization.

The model of Woo and Lam [19] assigns roles to the principals in a protocol and treats them as independent processes. The actions of the principals are described with no regard to the actions of others in the system. Our method for protocol specification and analysis is based on this notion.

Specifying protocols as in Figure 1 has another disadvantage. Protocol analysis is seen as a process separate from the specification. Current analysis techniques take a completed specification as input and attempt to reason about the completed protocol.[1] For example, Figure 2 is a depiction of the BAN logic. [2]

We suggest that protocol analysis should be integrated with the specification process. Thus, as a protocol is developed, beliefs and states of knowledge that represent the current state of the system are updated. At any point, an inconsistency can be detected. This has the advantage of identifying potential causes of problems as well as the actual flaws. When a proto-

[1] A notable exception is the NRL Protocol Analyzer by Syverson and Meadows [16]. However, this system is not modeled after the BAN type of reasoning about belief and knowledge, but uses the term-rewriting algebraic properties of a protocol.

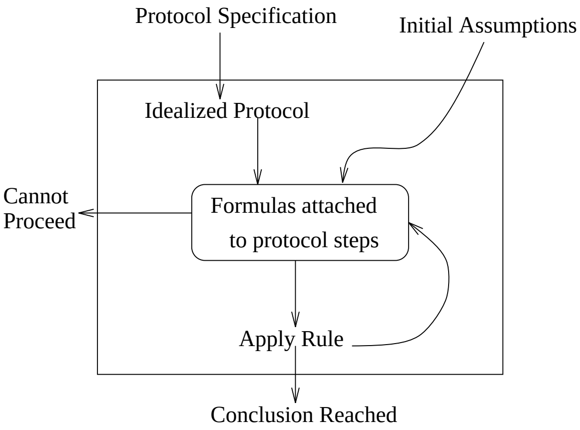

Figure 2: **Protocol analysis with the BAN logic.** The input to BAN is a protocol specification and the initial assumptions. At each step, formulas are attached to the protocol messages, and either a rule is applied, or the logic must halt. If possible, the desired conclusion is reached.

col has been completely specified, the analysis is complete as well.

3 Nonmonotonicity of Knowledge vs. Nonmonotonicity of Belief

With few exceptions, previous work in the application of the logic of knowledge and belief to the analysis of cryptographic protocols has considered only monotonic reasoning systems. In these systems, once something is believed, it is always believed. The same applies for knowledge.

The difference between knowledge and belief is subtle. A principal *knows* that his key is K. A principal *believes* that a nonce is fresh. In general, a principal knows things like secrets and data; a principal believes meta-data, or information about the data, such as freshness.

Monotonic systems have trouble reasoning with incomplete information. A belief that is assumed in the absence of other information, can be nullified by the introduction of new information. However, a monotonic system has no mechanism to do this. In fact, most previous systems have no refutation. The ability to

$B_i(p)$	$B_i(q)$	$B_i(p)$ unless $B_i(q)$
t	t	f
f	t	t
t	f	t
f	f	x

Figure 3: **The definition of Moser's *unless* operator.** The x in the last row indicates a special case. x is true iff $\exists r : B_i(p)$ unless $B_i(r) \in F$, where F is a conjunction of formulas containing the *unless* operator.

refute beliefs is important for reasoning about protocols. For example, if a session key is compromised, we need to change our belief that this is a good key.

Moser [10] gives a nonmonotonic logic of belief. This logic is biased towards belief in the absence of information. Thus, a final interpretation of a formula is believed unless there is some information that makes it inconsistent. The logic uses a construct called *unless* to achieve this. The value of a formula using *unless* can be seen from the truth table in Figure 3 (where F is a conjunction of formulas containing the *unless* operator and $B_i(p)$ means that principal i believes p). The x in the last row is a special case and is defined as follows:

$$x = \begin{cases} t & \text{if } \exists r : B_i(p) \text{ unless } B_i(r) \in F \\ & \text{and } B_i(r) \text{ is true} \\ f & \text{otherwise} \end{cases}$$

The logic of Moser suffers from intractability. In addition, the logic deals only with nonmononicity of *belief.* There is no known reasoning system that deals with the nonmonotonicity of *knowledge.* A situation where such reasoning applies is a protocol that requires a principal to no longer possess information it previously knew. This is different from a principal not believing a statement it previously believed.

The *khat* protocol [14], described in Section 4, is an example of a protocol that requires reasoning with nonmonotonicity of knowledge. The protocol relies on a public workstation "forgetting" some information. The BAN logic, along with its extensions, does not provide a way for representing this behavior.

In this paper, we introduce a method for analyzing protocols such as *khat*, where information is erased and no longer known by a principal. Our method uses *observers* sets for each secret that contain the principals who know it. These are similar to the knowledge sets of Kailar *et al.* [8] However, Kailar *et al.* use these sets to reason about belief, whereas we apply the concept in a slightly different way to reason about the nonmonotonicity of knowledge.

4 The KHAT protocol

The *khat* system of Rubin and Honeyman [14] was built to solve the problem of long running jobs in an authenticated environment where a trusted server issues tickets with limited lifetimes for services. *Khat* stands for Kerberized *at*, and is based on the UNIX *at* command. When using this service, a user schedules a job for a future time and date, with the option of renewing tickets until the job completes.

When a user submits a *khat* job, the program creates a spool file containing everything necessary to run the job at a later date, such as environment variables, and sends it to the *khat* server. The server stores the spool file for the job, and the user's workstation erases it from memory. The *khat* client generates a new key, N, which is used to encrypt the secret key, K, that will serve as the session key when it is time for the job to run, and the server and client need to communicate. N is also stored by the server and erased by the client. The process of securing the session key, K on the client is depicted in Figure 4.

The *khat* protocol is initiated by the usual ticket granting method in Kerberos. A ticket for the *khat* service is granted to the client after the initial authentication. This process is well known and is believed to be secure. Toussaint provides a proof that the kerberos ticket granting protocol is secure. [17] We take the results of the ticket granting process as the initial assumptions in our analysis. Thus, the *khat* protocol begins after ticket granting completes.

The *khat* system can be divided into two phases. Phase I works as follows.

1. A Kerberos ticket for *khat* is granted to the client and a session key is established.

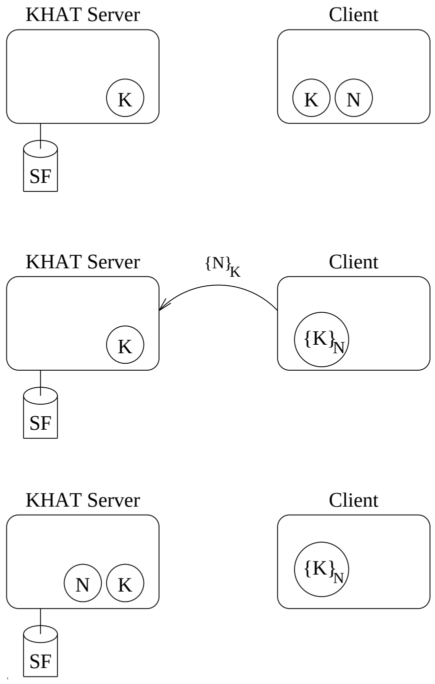

Figure 4: **How K is secured on the client in *khat*.** The first step is illustrated in the top diagram. The spool file, SF is stored by the server, and the client and server share a secret session key, K. The client generates a new key, N. Then, as shown in the next diagram, N is used to encrypt the session key, K, which remains on the client machine, while, N is sent to the server, sealed under the session key. Finally, as shown in the bottom diagram, K and N are stored on the server. When it is time for the job to run, N is sent to the client to unseal $\{K\}_N$.

2. The client generates a spool file for the job.

3. The spool file is sent to the server under the session key.

4. The server stores the spool file.

5. The client generates a new key, N, sends $\{N\}_K$ to the server, and erases the spool file and N from its memory.

6. The server stores N.

Phase II occurs when it is time for the job to run. The server wakes up once a minute to see if any *khat* jobs are ready. If so, phase II is initiated as follows:

1. The server sends N to the client.

2. The server sends the spool file to the client under the session key, K.

3. The client runs the job.

For a more complete discussion of *khat*, the reader is referred to the original paper. [14]

It is clear that this protocol cannot be specified as a simple list of messages such as in Figure 1. The specification method we present in this paper is more appropriate because we can include steps such as step 5 in Phase I.

The analysis depends on the assumptions in our threat model. We are concerned with an active intruder who has access to all network resources and can intercept, replace or delete any message. In addition, we are concerned with vacant workstations and information on them that can be useful to an intruder. This threat is also discussed in the *khat* paper [14]. In that paper, an informal discussion of the security risks of *khat* is given. The method we present here grew out of an attempt to analyze *khat* more formally, and to provide a method for analysis of any system that must reason about nonmonotonicity of knowledge.

5 Specifying a Protocol

In this section we provide definitions and notation for specifying a protocol. There are two types of definitions. Those in the first type are global to the protocol, and definitions of the second type are local to each principal.

To accommodate different levels of trust among principals, we place the beliefs of the principals in the local sets. If the assumption were to be made that each principal in the system is either trusted by everyone or trusted by no one, as is the case in many simple authentication systems, then we could have put these beliefs in the global set. To maintain generality, the level of trust and belief will be local. Thus, jurisdiction, the ability to assign session keys, is a belief that must be held by the parties sharing the keys, but not by everyone else.

The protocol designer may wish to specify and analyze a protocol for a system with untrustworthy principals. We include a *trust matrix* in the specification where the trust between each pair of principals is established. This is explained in Section 5.3.

As pointed out by Mao and Boyd, some statements in the BAN logic are not intuitive, such as the notion of believing a key or a nonce. [9] To remedy this, we define two local sets. One set is composed of the items that a principal possesses, such as encryption keys and nonces. The other set contains the principal's beliefs, such as the freshness of a key, or the possessions of another principal. Items in the possession sets are labeled by their origin. Each possession is accompanied by information that either states that it was generated by the principal himself, or states from whom it was received.

We define actions for dealing with the knowledge in a protocol, and inference rules for reasoning about belief. The actions are specified by the protocol designer and can be chosen from a specific set of actions defined below. Inference rules can be added by the designer although they will usually be the same across protocols.

5.1 Global Sets

The first step of the specification of any protocol using our method is to instantiate the global sets with values. It should be noted the contents of these sets change as a protocol run is simulated in the analysis. The specification of a protocol is simply the starting point of the analysis. In this section we give the definitions of the global sets used for protocol specification. We introduce $\mathcal{W}$, for world, to represent all the principals. Also, for each set, the subset n represents its cardinality, but this value changes from set to set.

Principal Set: This set contains the principals who participate in a protocol. $P = \{P_1, P_2, \cdots, P_n\}$. Any P_i may be marked as an initiator of the protocol. We will assume there is only one initiator.

Free Variable Set: This set contains variables that can be instantiated with any value. $FV = \{V_1, V_2, \cdots, V_n\}$.

Bound Variable Set: This set contains variables that are bound to certain values. $BV = \{X_1, X_2, \cdots, X_n\}$.

Rule Set: This set contains inference rules for deriving new statements from existing assertions. These are the same as the inference rules in the BAN logic. $R = \{R_1, R_2, \cdots, R_n\}$ where R_i is of the form $\frac{C_1, C_2, \cdots, C_n}{D}$, C_i is a condition and D is a statement.

Secret Set: This set contains all of the secrets that exist at any given time in the system. The cardinality of this set changes during the analysis as new secrets, such as session keys, are added. $S = \{S_1, S_2, \cdots, S_n\}$.

Observers Sets: For each S_i, $\mathit{Observers}(S_i)$ contains all the principals who could possibly know the secret S_i by listening to network traffic or generating it themselves. The members of the *Observers* sets can be stated explicitly or maintained as formulas representing their membership.

The set, P, contains names of the participants in a protocol. A typical example might be, $P = \{A, B, AS\}$, where A and B are regular principals and AS is the authentication server.

Recall that in BAN logic $\#(X)$ means that X is fresh. The set BV contains variables such as X in the following rule (nonce verification):

$$\frac{\text{P believes } \#(X),\ \text{P believes Q said X}}{\text{P believes Q believes X}}$$

The first occurrence of the variable X can take on any value. However, the other Xs in the rule are bound to the value of the first X once it is instantiated; so X is in BV. The set FV contains variables such as the Y in the rule:

$$\{\{Y\}_K\}_K = Y$$

Here, Y can take on any value independent of the rest of the protocol.

An example of the set S is $\{K_{ab}, K_{as}, K_{bs}\}$. This set contains secret keys held among A and the authentication server, among B and the authentication server, and a session key among A and B. The session key would not be a member of S in a specification where K_{ab} is distributed in the protocol, but would be added to the set during the analysis at the point in which it was generated by the authentication server. This process is discussed in the analysis section. In this example,

$Observers(K_{ab}) = \{A, B\}$, $Observers(K_{as}) = \{A, S\}$, and $Observers(K_{bs}) = \{B, S\}$. Also, $\mathcal{W} \in Observers(K)$ means that all principals know K.

5.2 Local Sets

Local sets are private to each principal in a protocol specification. In this section we define these sets. Later, we will show how they are used in the actual specification and analysis of a protocol. For each principal, P_i, we define the following sets:

Possession Set(P_i) This set contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. $POSS(P_i) = \{poss_1, poss_2, \cdots, poss_n\}$. $poss_i$ contains two fields: the actual data and the origin of the data.[2]

Belief Set(P_i) This set contains all the beliefs held by a principal. This includes the belief that the keys it holds between itself and other principals are good, beliefs about jurisdiction, beliefs about freshness, and beliefs about the possessions of other principals. $BEL(P_i) = \{bel_1, bel_2, \cdots, bel_n\}$.

Behavior List(P_i) This item is a list rather than a set because the elements are ordered. $BL = \{AL, bvr_1, bvr_2, \cdots, bvr_n\}$. AL is an action list as will be defined below.

Figure 5 shows the structure of BL. The first element of BL, is an action list. The remaining elements, bvr_i, are pairs, $(Mess, AL)$ consisting of a message operation, $Mess$, and an action list, AL.

A message operation is one member of the set $\{Send(P_j, msg), Receive(P_j, msg)\}$. $Send(P_j, msg)$ means that P_i sends the message, msg to P_j. Similarly, $Receive(P_j, msg)$ means that P_i receives message msg from P_j.

[2]Note that the second field represents whether or not P_i generated the data, or who sent it to P_i. It does *not* represent who originated the data.

BEHAVIOR LIST

$action_1, action_2, action_3, \cdots$

Message operation

$action_1, action_2, action_3, \cdots$

Message operation

$action_1, action_2, action_3, \cdots$

Message operation

$action_1, action_2, action_3, \cdots$

$\cdots$

END OF LIST

Figure 5: **The structure of a behavior list.** The list contains a list of actions, followed by a list of pairs, (message operation, action list). After each action, any relevant inference rules are applied.

In this case, msg will be marked as coming from P_j and added to $POSS(P_i)$.

In a *send* operation, msg contains the information transmitted. In a *receive* operation msg contains the fields of the expected message. This represents P_i's expectation about the structure of the message. This is similar to the notion of recognizability of the GNY logic. [7]

An action list is an ordered list of zero or more actions that are performed by P_i. Actions consist of operations such as encryption and decryption, deletion of information, application of functions, and the decision whether to abort the protocol. They are covered in more detail in section 5.5. Every action is followed by a check of the inference rules. If the conditions of a rule are satisfied as a result of the action, then the rule is applied. These rules are used to update the belief sets of the principals.

Action lists play an important role in protocol specification. Previous approaches to cryptographic protocol analysis take the actions of the principals for granted. Operations such as encryption and decryption are implicit. Our method makes every action explicit, including verification that the operations completed successfully, and an abort in case they did not. This method is a better model of protocol execution in a real system than previous approaches because all of the actions are included as part of the specification instead of implicitly assumed.

5.3 The Trust Matrix

Our method does not require that any assumptions be made about trust between principals. Instead, the protocol designer explicitly specifies the trust relationship between every pair of principals. We define the matrix, TRUST:

$$TRUST[i,j] = \begin{cases} 1 & \text{if } P_i \text{ trusts } P_j \\ 0 & \text{if } P_i \text{ does not trust } P_j \end{cases}$$

The rows and columns enumerate the principals in P. Obviously, when $i = j$, TRUST[i,j] = 1. P_i trusts P_j means that P_i behaves as though P_j will follow the protocol. We give an example of this using a nonmonotonic protocol.

Say that A believes that B possesses X. Now say that the protocol requires that B forget X. As both A and B know the protocol, B should now remove the belief that B possesses X from its belief set. However, if B does not trust A, then he cannot be sure that A actually no longer possesses X. In the actions described below, we stipulate the condition that A trusts B before removing a belief about the possessions of A.

5.4 A Word About Nonces

Message freshness can be guaranteed only with time-stamps and nonces. Conceptually, a nonce is a large random number whose purpose is to link a challenge and a response. If A sends a nonce, N_a, to S, then any message including $f(N_a)$, for some function f, and encrypted under K_{as}, is assumed to be fresh if and only if the following conditions are satisfied:

1. No previous message containing $f(N_a)$ has been received.
2. K_{as} is fresh. That is, we assume K_{as} is known only to A and S.

Our method uses inference rules to propagate belief about freshness. In section 5.5, we introduce a new construct, LINK(N_a) to link a response to a challenge. When a principal generates a nonce, N_a, the formula LINK(N_a) is added to his belief set. When a message is received containing, N_a, the LINK item is removed from the belief set, and all parts of that message are labeled as being fresh. A reply to the challenge can be accepted only once. If that message were to be received again, the absence of the LINK item in the belief set would hinder the conclusion that this message is fresh. In fact, this is how our analysis technique exposes the weaknesses in protocols vulnerable to replay attacks. Our analysis of the Needham and Schroeder protocol (Section 6.1) gives an example of this.

5.5 Actions

Actions describe how a principal constructs messages, encrypts and decrypts data, computes functions, aborts a protocol, and performs any other operation. The action lists that precede and follow message operations in a principal's behavior list determine sequence of events performed by the principal during a protocol run. As demonstrated below, some of the actions replace inference rules in the BAN logic, and others explicitly represent operations that were taken for granted in previous approaches.

In this section we define the actions used in our method, and the following section presents and discusses the inference rules. Our method requires some new notation and dispenses with some previous constructs. As will be shown, the *said*, *sees*, *controls*, and $Q \overset{K}{\leftrightarrow} P$, constructs of the BAN logic are not needed. The new definitions follow:

X contains Y means that Y appears as a submessage of the message X, more formally, for some (possibly null) x_1, x_2, $X = x_1 \cdot Y \cdot x_2$.[3] It is always the case that X contains X.

S := f(S) represents assignment. The value of S is replaced by the value of the function f applied to S.

X from P means that X is labeled as having been received from P. This will also be true if P generated X.

[3] We adopt the usual convention of $\cdot$ for concatenation.

LINK(N_a) is used to link challenges and responses. This formula is added to the belief set of a principal who generates the nonce N_a, and allows only one subsequently received message to contain the nonce N_a. After such a message is received, the formula is removed from the belief set.

With these new definitions, we now define the actions for a given principal, P_i. Although not specified in the definitions, we assume that *from* labels are inherited in operations. For example, if $\{X\}_k$ is from Q, and is in $POSS(P_i)$, and this is decrypted, then X is also labeled "from Q" when it is added to $POSS(P_i)$.

1. **Encrypt(X, k)**

   condition: $X, k \in POSS(P_i), P_i \in Observers(k)$

   result: $POSS(P_i) := POSS(P_i) \cup \{\{X\}_k\}$

   description: This action is used when a principal encrypts data. If P_i possesses X and knows k then he can possess $\{X\}_k$.

2. **Decrypt($\{X\}_k, k$)**

   condition: $P_i \in Observers(k)$, $\{X\}_k, k \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup \{X\}$

   description: This action is used when a principal decrypts data. If P_i possesses X, encrypted under k, and P_i knows k, then P_i can possess X.

3. **Generate-nonce(N)**

   result: $POSS(P_i) := POSS(P_i) \cup \{N\}$, $BEL(P_i) := BEL(P_i) \cup LINK(N)$

   description: This action is used when a principal generates a nonce to link a challenge and a response. LINK(N) is removed from $BEL(P_i)$ when the response is received. This is used to determine freshness.

4. **Generate-secret(s)**

   result: $S := S \cup \{s\}$, $Observers(s) = \{P_i\}$, $POSS(P_i) := POSS(P_i) \cup \{s\}$, $BEL(P_i) := BEL(P_i) \cup \#(s)$

   description: This action is used when a principal generates a secret data item, such as a key. A new secret, s, is added to S, and the *Observers* and possession sets are updated.

5. **Concat($X_1, X_2, \cdots, X_n$)**

   condition: $X_1, X_2, \cdots, X_n \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup \{X_1 \cdot X_2 \cdots X_n\}$

   description: This action is used when a principal constructs a message, X, out of submessages $X_1, X_2, \cdots, X_n$.

6. **Split(X)**

   condition: X *contains* $x_1 \cdot x_2 \cdots x_n$, $X \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup \{x_1, x_2, \cdots, x_n\}$

   description: This action is used to break a message into its components. Split is the opposite of concatenation.

7. **Forget(X)**

   condition: $X \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) - \{X\}$, $\forall P_j \in P$ **if** $TRUST[j, i] = 1$ **then** $BEL(P_j) := BEL(P_j) - \{X \in POSS(P_i)\}$

   description: This action is used when P_i no longer is in possession of X. All principals who trust P_i believe that P_i no longer possesses X.

8. **Forget-secret(s)**

   condition: $P_i \in Observers(s), s \in POSS(P_i)$

   result: $Observers(s) := Observers(s) - \{P_i\}$, $POSS(P_i) := POSS(P_i) - \{s\}$, $\forall P_j \in P$ **if** $TRUST[j, i] = 1$ **then** $BEL(P_j) := BEL(P_j) - \{s \in POSS(P_i)\}$

   description: This action is used when P_i no longer knows the secret s. All principals who trust P_i believe that P_i no longer possesses s.

9. **Apply(f, X)**

condition: $f, X \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \cup \{f(X)\}$

description: This action is used when P_i applies the function f to X. After the application, P_i possesses $f(X)$.

10. Check-freshness(X)

condition: $X \in POSS(P_i)$

result: $BEL(P_i) := BEL(P_i) \cup \{\#(X)\}$

description: This action is used to verify that time-stamp X is fresh.

11. Abort

condition: Protocol run is illegal.

result: Analysis reports failure.

description: This could happen under various circumstances where there is an inconsistency or other flaw in the protocol specification.

The difference between actions such as *generate* and actions such as *generate-secret* is that items generated as secrets are expected to be sent encrypted, and others are expected to be transmitted in the clear at some point. Many protocols send challenges in the clear; therefore, there is no need to maintain these items as secrets.

The actions described above are used to control the knowledge and possessions of the principals in a protocol. Except for *Check-freshness* and *abort*, all of the actions modify the possession or *Observers* sets. Inference rules are used to modify the belief sets.

The difference between actions and inference rules is that actions are explicitly specified as a part of the protocol. Rules, however, are used to reason about the beliefs of principals as the protocol executes. The protocol builder does not explicitly state how belief evolves in a protocol, but rather, states the inference rules that will mechanically control the propagation of belief.

5.6 The Update Function

Before discussing inference rules, we define an important function for processing *send* message operations. When a principal, P_i sends a message to P_j on the network, any principal can read it. In our threat model, we view any message as being broadcast and available to all. As pointed out by Nessett, the BAN logic does not deal with protocols in which a principal publishes a secret key [13]. (This is discussed in Section 6.3.) The purpose of the *update* function is to update the *Observers* sets of all secrets that are sent on the network. *Update* is defined in Figure 6.

If a secret, x_1, is sent in the clear, then $Observers(x_1)$ is set to $\mathcal{W}$ to indicate that any principal now knows x_1. The union of $\mathcal{W}$ with any set is $\mathcal{W}$.

The initial call after a *send* operation, $Send(X, P_j)$ is Update(X,$\mathcal{W}$). If a submessage, $\{Y\}_k$ is contained in X, *update* will recurse after it is called. Then, W will contain $Observers(k)$ because only the principals in this set should be added to the *Observers* sets of the elements of Y.

When X is of the form $\{Y_1\}_k$, *update* is called with $Observers(k) \cap W$. This is because for

---

UPDATE(X, W)

  if $X = NULL$ **then** return;

  if $X = x_1$ **then**
    $Observers(x_1) := Observers(x_1) \cup W$;
    return;

  if $X = \{Y_1\}_k$ **then**
    update($Y_1, Observers(k) \cap W$);
    return;

  if $X = Y_1 \cdot Y_2$ **then**
    update(Y_1, W); update(Y_2, W);
    return;

END;

Figure 6: **The Update function.** In this function, x_1 is a single, unencrypted data element, such as a nonce or a key. Y_1 and Y_2 are formulas containing combinations of clear or encrypted elements such as these. The parameter W represents the principals to be added to the knowledge set of a secret.

a secret nested in the encryption by different keys, only principals possessing all of the keys should be added to the *Observers* set. For example, in the call *update*($\{\{X\}_{k_1}\}_{k2}$,$\mathcal{W}$), the principals added to $Observers(X)$ should be in both of $Observers(k_1)$ and $Observers(k_2)$.

There is a subtle flaw in the *update* function that we will demonstrate by example and fix. Take *update*($\{\{X\}_{k_1}, k_1\}_{k_2}, \mathcal{W}$). Now, say we have a principal, P, who is in $Observers(k_2)$, but is not in $Observers(k_1)$. After the function call, P will be added to $Observers(k_1)$. However, P will not be added to $Observers(X)$. The problem is that *update* recurses on $\{X\}_{k_1}$ before P is added to $Observers(k_1)$. To fix this, *update* is called continuously until no more *Observers* sets change. We will assume this implicitly in our protocol specifications.

We now give an example (see Figure 7). Take $X = N_a \cdot \{K_{ab}, T_s\}_{k_1} \cdot \{data\}_{k_2}$. The initial call is *update*($N_a \cdot \{K_{ab} \cdot T_s\}_{k_1} \cdot \{data\}_{k_2}$,$\mathcal{W}$). This will match the last condition, so *update* will now be called twice, *update*(N_a,$\mathcal{W}$) and *update*($\{K_{ab} \cdot T_s\}_{k_1} \cdot \{data\}_{k_2}$,$\mathcal{W}$). The first call will cause $Observers(N_a)$ to be $\mathcal{W}$, as desired.

The second call will further split into two calls, *update*($\{K_{ab} \cdot T_s\}_{k_1}, \mathcal{W}$) and *update*($\{data\}_{k_2}, \mathcal{W}$). The first of these will match the third rule and result in *update*($K_{ab} \cdot T_s$,$Observers(k_1)$). After further iterations, the principals in $Observers(k_1)$ will be added to the *Observers* sets of K_{ab} and T_s. Similarly, $Observers(data)$ will be the union of itself with $Observers(k_2)$.

5.7 Inference Rules

When using our method, the protocol developer must choose from the eleven actions defined above. However, as in the BAN logic, he is given the freedom to specify his own inference rules. Some rules, such as the nonce verification rule, will be used by any useful protocol. In our method, this protocol is specified a bit differently.

The nonce verification rule as defined by Burrows *et al.* [2] is not entirely intuitive. Their rule is:

$$\frac{\text{P believes } \#(X),\ \text{P believes Q said X}}{\text{P believes Q believes X}}$$

It is not clear what it means for Q to believe X, if X is a nonce. In our method, if P received X from Q, then $X \in POSS(P_i)$, and X will be labeled as being from Q. We define the nonce verification rule as follows:

$$\frac{\#(X) \in BEL(P),\ X \text{ from } Q \in POSS(P)}{BEL(P) := BEL(P) \cup \{Q \text{ believes } \#(X)\}}$$

Thus, P can establish that Q believes that X is fresh, and this fact is added to P's belief set, $BEL(P)$. Notice that rules are used to propagate belief during a protocol run, whereas actions deal with knowledge.

The message meaning rule is defined in BAN as:

$$\frac{\text{P believes Q} \stackrel{K}{\leftrightarrow} \text{P, P sees } \{X\}_K}{\text{P believes Q said X}}$$

This rule states that if P believes that Q and P share a secret key, K, and P sees X, encrypted under K, and P did not encrypt X under K, then P believes that Q once said X. This implies that knowing a principal shares a secret key with another principal is enough to guarantee that any message encrypted under that key was sent by that principal. In our rule, this requirement is made explicit. We define the message meaning rule as follows:

$$\frac{\{X\}_k \text{ from } Q \in POSS(P),\ \{P, Q\} \subseteq Observers(k)}{BEL(P) := BEL(P) \cup \{X \in POSS(Q)\}}$$

This rule states that if $\{X\}_k$ was received by P, from Q, and both P and Q know the key k, then P believes that Q possesses X. This is possible because messages are labeled with their origin when they are added to the possession set. When principal P applies action 2 (decrypt), X will be in his possession set.

Another rule is needed to reason about the freshness of submessages:

$$\frac{\#(x_1) \in BEL(P),\quad \{X \text{ contains } x_1, X \text{ contains } x_2\} \subseteq POSS(P)}{BEL(P) := BEL(P) \cup \#(x_2)}$$

This rule states that if P believes that x_1 is fresh, and P possesses a formula containing both x_1 and x_2, then P believes that x_2 is fresh. This rule reflects the fact that any part of a message which contains something fresh, is fresh.

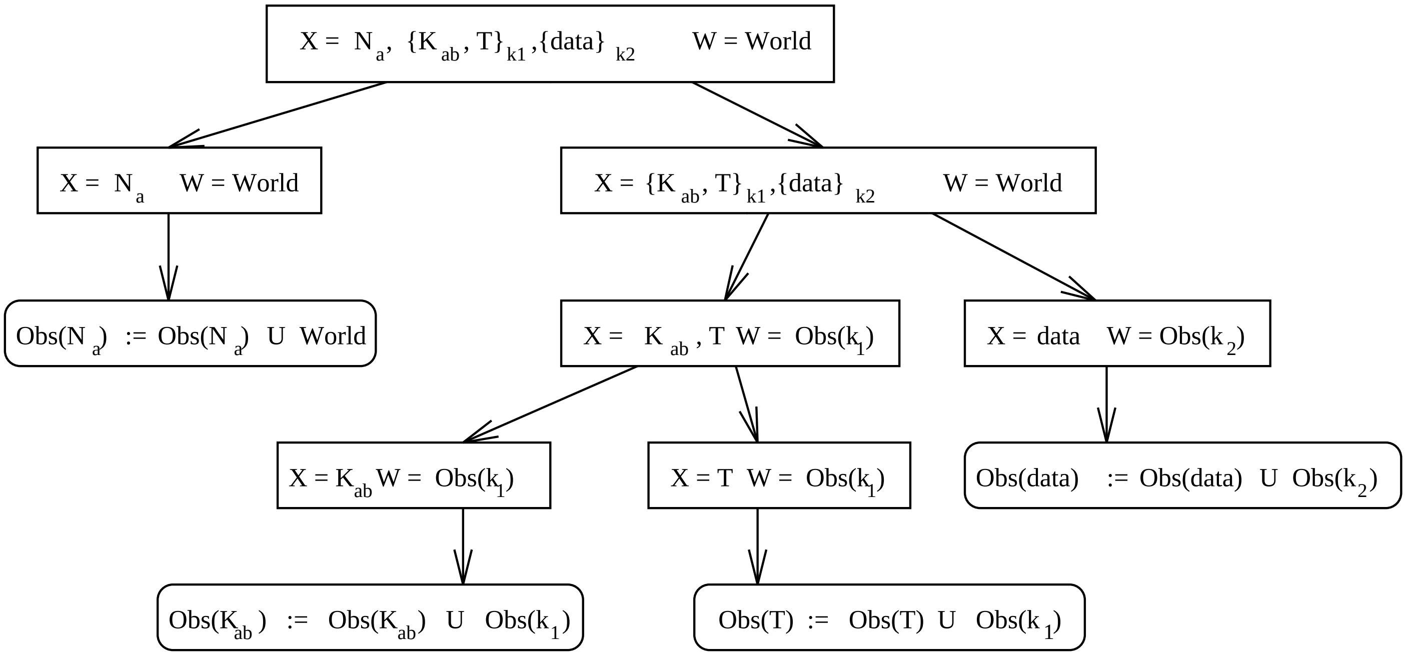

Figure 7: **The update function.** This diagram shows the recursive tree for an example of the *update* function. The initial call is $update(N_a \cdot \{K_{ab} \cdot T_s\}_{k_1} \cdot \{data\}_{k_2}, \mathcal{W})$. Solid rectangles represent internal nodes on the tree, and the leaves in the rounded rectangles show the action that is taken. This examples shows how the *Observers* sets of secrets are updated when a message is sent.

The following rule is the most important inference rule for reasoning about freshness in a protocol. The only way for a message to be fresh is for it to contain a valid time-stamp or a nonce that has never previously been used in a response. The rule for determining freshness using a nonce is the linkage rule:

$$\frac{\begin{array}{c}\#(k) \in BEL(P), P \in Observers(k), \\ LINK(N_a) \in BEL(P), X \text{ contains } f(N_a), \\ X \text{ contains } x_1, \{X\}_k \text{ from } Q \in POSS(P)\end{array}}{BEL(P) := (BEL(P) - LINK(N_a)) \cup \{\#(x_1)\}}$$

This rule is simpler than its unfortunate length makes it appear. It is the only rule that can be used to add information about the freshness of an item which is not known to contain fresh submessages, to a principal's belief set.

The linkage rule states that the submessages of a message X are believed to be fresh under certain conditions. If LINK(N_a) is in P's belief set, then the nonce N_a has not been used before. This is the first condition. If the rule is applied successfully, the LINK item is removed. So the rule could not fire again for the same nonce. Other conditions state that the nonce N_a must be sealed under a key that is fresh, and must be available to P.

The rule states that message X must contain $f(N_a)$ to represent the fact that sometimes a function of a nonce, rather than the actual nonce is used to respond to a challenge. The net result of applying this rule is that any submessage of a valid response to a challenge is believed to be fresh by the recipient of the response. Also, there is a guarantee that any replay of a valid response will not result in a principal believing that the submessages are fresh.

6 Examples

The best way to explain how a protocol is analyzed using our method is by example. In section 6.1, the Needham and Schroeder protocol is specified, and we step through the analysis. In section 6.2, we apply our method to the *khat* protocol.

6.1 Needham and Schroeder

First we specify the protocol, and then we show how our method can be used to analyze it. We demonstrate that the known flaw in the protocol exists; principal B cannot achieve the belief that the session key with A is fresh. Then, we show how the addition of two messages, as proposed by Needham and Schroeder (in a later paper [12]) to solve the problem, allows B to achieve the desired belief.

6.1.1 The Specification

The Needham and Schroeder protocol assumes that both A and B trust S. So, the trust matrix contains 1s in the appropriate spots to represent this. The trust between A and B is irrelevant, and so the matrix values do not matter.

First we define the global sets. $P = \{S, B, A\}$. A is marked as the initiator of the protocol. R contains the rules defined in Section 5.7. $S = \{K_{as}, K_{bs}\}$. Each of these secret keys has an *Observers* set. $Observers(K_{as}) = \{A, S\}$ and $Observers(K_{bs}) = \{B, S\}$. Some of these sets will change once the analysis begins.

Next, we define the initial values of the local sets. Notice that initially, principals believe in the freshness of the key they share with the server, S. Similarly, the server believes in the freshness of its shared secret with each principal. Also, $P \bowtie K_{pq}$ represents the key, K_{pq} and the fact that it is to be used for communicating with principal P. In this protocol, the function f subtracts one from its argument.

Principal A

$POSS(A) = \{K_{as}\}$
$BEL(A) = \{\#(K_{as})\}$
$BL(A) =$
• Generate-nonce(N_a)
  Concat(A, B, N_a)
  Send($S, \{A \cdot B \cdot N_a\}$)
  Update($\{A \cdot B \cdot N_a\}, \mathcal{W}$)
  Receive($S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)
  Decrypt($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, K_{as}$)
  Split($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$)
  Send($B, \{A \bowtie K_{ab}\}_{K_{bs}}$)
  Update($\{A \bowtie K_{ab}\}_{K_{bs}}, \mathcal{W}$)
  Receive($B, \{N_b\}_{K_{ab}}$)
  Decrypt($\{N_b\}_{K_{ab}}, K_{ab}$)
  Send(B, Encrypt(Apply(f, N_b), K_{ab}))
  Update($\{f(N_b)\}_{K_{ab}}, \mathcal{W}$)

Principal B

$POSS(B) = \{K_{bs}\}$
$BEL(B) = \{\#(K_{bs})\}$
$BL(B) =$
  Receive($A, \{A \bowtie K_{ab}\}_{K_{bs}}$)
  Decrypt($\{A \bowtie K_{ab}\}_{K_{bs}}, K_{bs}$)
  Generate-nonce(N_b)
  Send(A,Encrypt(N_b, K_{ab}))
  Update($\{N_b\}_{K_{ab}}, \mathcal{W}$)
  Receive($A, \{f(N_b)\}_{K_{ab}}$)
  Decrypt($\{f(N_b)\}_{K_{ab}}, K_{ab}$)

Principal S

$POSS(S) = \{K_{as}, K_{bs}\}$
$BEL(S) = \{\#(K_{as}), \#(K_{bs})\}$
$BL(S) =$
  Receive($A, \{A \cdot B \cdot N_a\}$)
  Split($\{A \cdot B \cdot N_a\}$)
  Generate-secret(K_{ab})
  Send(A,Encrypt(Concat($N_a, B \bowtie K_{ab}$,
    Encrypt($A \bowtie K_{ab}, K_{bs}$)), K_{as}))
  Update($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, \mathcal{W}$)

Once the protocol has been specified, the analysis begins. However, the analysis technique described here can be used to test the protocol as it is being developed.

The first action in $BL(A)$ is marked with a • because A is the initiator of the protocol. For each action, its condition is tested. If it does not hold, the protocol analysis is aborted, and the specification is infeasible. If the condition holds, then the result is applied and the required sets are updated. Next, the inference rules are examined to see if any apply. Finally, the action is marked with a ∘ to show that it has been successful, and the mark, •, is moved to the next action.

Every `Send` action is followed by an `Update` action. The `Send` action specifies to whom the message is sent. After an `Update` action, the mark moves to the first `Receive` action with no ∘ of the principal identified in the corresponding *Send* action.

6.1.2 The Analysis

The first four actions in $BL(A)$ are executed resulting in new members of the sets $POSS(A)$ and $BEL(A)$. Also, the `Update` action causes $Observers(N_a) = \mathcal{W}$. So far, no inference rules can be applied.

$POSS(A) = \{K_{as}, N_a, \{A \cdot B \cdot N_a\}\}$
$BEL(A) = \{\#(K_{as}), \text{LINK}(N_a)\}$
$BL(A) =$
∘ Generate-nonce(N_a)
∘ Concat(A, B, N_a)
∘ Send($S, \{A \cdot B \cdot N_a\}$)
∘ Update($\{A \cdot B \cdot N_a\}, \mathcal{W}$)
  Receive($S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)
...

After the `Update` action, the next action to be executed is in S's behavior list because the `Send` action specifies S.

- Receive($A, \{A \cdot B \cdot N_a\}$)

The five actions of $BL(S)$ are executed. There are still no relevant inference rules. The set S now contains $\{K_{as}, K_{bs}, K_{ab}\}$. After applying the `Update` function, $Observers(K_{ab}) = \{S, A\}$ because $A \in Observers(K_{as})$. The term $\{A \bowtie K_{ab}\}_{K_{bs}}$ does not cause B to be added to the *Observers* set of K_{ab} because B is not a member of $Observers(K_{as})$, and so B is not in $Observers(K_{as}) \cap Observers(K_{bs})$. The possession set contains subparts of messages that were built as the messages were constructed, but we omit these here for space consideration as they do not contribute in any way to the analysis. The new values of S's local sets are:

$POSS(S) = \{K_{as}, K_{bs}, \{A \cdot B \cdot N_a\}$ from A,
  $K_{ab}, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}\}$
$BEL(S) = \{\#(K_{as}), \#(K_{bs}), \#(K_{ab})\}$
$BL(S) =$
∘ Receive($A, \{A \cdot B \cdot N_a\}$)
∘ Split($\{A \cdot B \cdot N_a\}$)
∘ Generate-secret(K_{ab})
∘ Send(A,Encrypt(Concat($N_a, B \bowtie K_{ab}$,
  Encrypt($A \bowtie K_{ab}, K_{bs}$)), K_{as}))
∘ Update($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, \mathcal{W}$)

The next action is in A's BL.

- Receive($S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)

The term $\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$ is added to POSS(A). The next action to be executed is:

- Decrypt($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, K_{as}$)

This will add the term $\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$ to $POSS(A)$. The next action, `Split` will add the individual components too.

At this point, the conditions for the linkage rule are satisfied. We take X to be the term $\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$ that was just added to $POSS(A)$. The reader can verify that the following are all true:

1. $\#(K_{as}) \in BEL(A)$
2. $A \in Observers(K_{as})$
3. LINK(N_a) $\in BEL(A)$
4. X *contains* $g(N_a)$, where g is the identity function
5. $\{X\}_{K_{as}}$ from $S \in POSS(A)$

Once the linkage rule is applied, the freshness of each subpart of X is added to belief set of A. Also, the LINK formula is removed from the belief set so that the nonce N_a cannot be used again.

At this point, the global sets have not changed. The sets for principal A are as follows (We omit items in the possession and belief sets, such as large concatenated messages, that serve no further purpose.):

Principal A

$POSS(A) = \{K_{as}, N_a, B \bowtie K_{ab}, \{A \bowtie K_{ab}\}_{K_{bs}}\}$
$BEL(A) = \{\#(K_{as}), \#(K_{ab}), \#(\{A \bowtie K_{ab}\}_{K_{bs}})\}$
$BL(A) =$
∘ Generate-nonce(N_a)
∘ Concat(A, B, N_a)
∘ Send($S, \{A \cdot B \cdot N_a\}$)
∘ Update($\{A \cdot B \cdot N_a\}, \mathcal{W}$)
∘ Receive($S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)
∘ Decrypt($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, K_{as}$)
∘ Split($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$)
• Send($B, \{A \bowtie K_{ab}\}_{K_{bs}}$)
  Update($\{A \bowtie K_{ab}\}_{K_{bs}}, \mathcal{W}$)
  Receive($B, \{N_b\}_{K_{ab}}$)
  Decrypt($\{N_b\}_{K_{ab}}, K_{ab}$)
  Send(B, Encrypt(Apply(f, N_b), K_{ab}))
  Update($\{f(N_b)\}_{K_{ab}}, \mathcal{W}$)

The `Send` and `Update` actions in A's behavior list are executed next. The `Update` function adds B to $Observers(K_{ab})$. The next action to be executed is in B's behavior list, as specified by the last `Send` action.

- Receive$(A, \{A \bowtie K_{ab}\}_{K_{bs}})$

The next action on B's behavior list is:

- Decrypt$(\{A \bowtie K_{ab}\}_{K_{bs}}, K_{bs})$

After this action is executed, $\{A \bowtie K_{ab}\}$ is added to $POSS(B)$. However, the linkage rule does not apply because there is no LINK statement in $BEL(B)$. Thus, B *cannot conclude that* K_{ab} *is fresh!*

In fact, when B receives $\{f(N_b)\}_{K_{ab}}$ from A, it cannot apply the linkage rule because one of the conditions is that K_{ab} is fresh. For the remainder of the protocol, B can never conclude that anything received under K_{ab} is fresh.

This is the same flaw discovered in the Needham and Schroeder protocol by Denning and Sacco. [5] We apply Needham and Schroeder's fix [12] by adding several actions to the beginning of the behavior lists of A and B. To $BL(A)$, we add the actions:

Send$(B, \{A\})$
Update$(\{A\}, \mathcal{W})$
Receive$(B, \{A, J\}_{K_{bs}})$
Decrypt$(\{A \cdot J\}_{K_{bs}}, K_{bs})$
Split$(\{A \cdot J\})$

And to $BL(B)$, we add the actions:

Receive$(A, \{A\})$
Generate-nonce(J)
Send$(A$, Encrypt(Concat$(A, J), K_{bs}))$
Update$(\{A \cdot J\}_{K_{bs}}, \mathcal{W})$

Then, A will include J in the original message to S, and S will include it in $\{A \bowtie K_{ab}\}_{K_{bs}}$ that gets forwarded to B.

It is clear that when B generates J, a LINK statement is added to $BEL(B)$. When B receives the message containing K_{ab} from A, it will be able to conclude $\#(K_{ab})$. Also, because $Observers(K_{bs}) = \{B, S\}$ throughout the protocol, no intruder could generate or modify the forwarded message from A to B that is sealed under K_{bs}.

Thus, our analysis reveals no flaws in the revised Needham and Schroeder protocol.

6.2 KHAT

Our method for analyzing cryptographic protocols does not include temporal reasoning. Thus, we specify and analyze the two phases of the *khat* protocol separately.

One advantage of our method is that the *khat* protocol can be specified in the same manner as the Needham and Schroeder protocol; we specify all the global and local sets. The behavior lists will contain actions that precisely describe the protocol. Section 4 showed that the previous method of listing the messages between principals is inadequate as a specification technique.

Our analysis reveals a significant flaw in the *khat* protocol. We provide a fix to the protocol, and use the analysis to demonstrate that the flaw no longer exists.

6.2.1 The Specification

The *khat* protocol involves two principals: the client (C) and the server (S). In this protocol, the trust matrix must reflect the fact that they trust each other. The client trusts the server to issue valid tickets, and the server trusts the client to forget the information specified in the protocol. If the TRUST$[i, j]$, where i is the server and j is the client, is not 1, then the server will not believe that the client no longer possesses information which should be forgotten. Thus, a fundamental assumption of the protocol is identified.

When phase I begins, we assume that a secure channel has been established using the Kerberos ticket for the *khat* service. Thus, K is the session key between C and S, and $Observers(K) = \{C, S\}$. $P = \{S, C\}$, C is marked as the initiator, and $S = \{K\}$. In this specification, SF represents the spool file for the user's job. The local sets are now defined:

Client

$POSS(C) = \{K\}$
$BEL(C) = \{\#(K)\}$
$BL(C) =$

- Generate-secret(SF)
  Generate-secret(N)
  Encrypt$(K$,$N)$
  Send$(S$,Encrypt(Concat$(SF, N), K))$

Update($\{SF \cdot N\}_K, \mathcal{W}$)
Forget-secret(N)
Forget-secret(SF)[4]

Phase II

Receive($S, \{N \cdot \{SF \cdot TGT_C\}_K\}$)
Split($\{N \cdot \{SF\}_K\}$)
Decrypt($\{K\}_N, N$)
Decrypt($\{SF \cdot TGT_C\}_K, K$)
Split($\{SF \cdot TGT_C\}$)
Check-Freshness(TGT_C)

Server

$POSS(S) = \{K\}$
$BEL(S) = \{\#(K)\}$
$BL(S) =$
Receive($C, \{SF \cdot N\}_K$)
Decrypt($\{SF \cdot N\}_K, K$)
Split($\{SF \cdot N\}$)

Phase II

Generate-secret(TGT_C)
Send(Concat(N,Encrypt(Concat(SF, TGT_C), K)))
Update($\{N \cdot \{SF \cdot TGT_C\}_K\}$, W)

6.2.2 The Analysis

We begin our analysis with Phase I of the protocol. After the analysis reaches the first `Forget-secret` statement, the local sets are as follows (once again, for the sake of clarity, we omit some encrypted items in the possession and belief sets that don't contribute to the analysis):

Client

$POSS(C) = \{K, SF, N, \{K\}_N, \}$
$BEL(C) = \{\#(K), \#(N), \#(SF)\}$
$BL(C) =$
∘ Generate-secret(SF)
∘ Generate-secret(N)
∘ Encrypt(K,N)
∘ Send(S,Encrypt(Concat(SF, N), K))
∘ Update($\{SF \cdot N\}_K, \mathcal{W}$)
• Forget-secret(N)
Forget-secret(SF)

Server

$POSS(S) = \{K, N, SF\}$
$BEL(S) = \{\#(K)\}$
$BL(S) =$
∘ Receive($C, \{SF \cdot N\}_K$)
∘ Decrypt($\{SF \cdot N\}_K, K$)
∘ Split($\{SF \cdot N\}$)

Notice that the server cannot conclude $\#(SF)$ or $\#(N)$. This is a serious flaw because an intruder can use a replay attack for the remainder of the session[5] to reschedule the user's job.

To solve this problem, we modify the protocol so that along with the *khat ticket*, the server sends a list of fresh nonces to the client. Each time the user schedules a job, he includes an unused nonce in the message. In the analysis, the server will have a collection of LINK statements in its belief set, and the freshness of N and SF can be guaranteed.

6.2.3 The Corrected Protocol

The corrected protocol is as follows:

Client

$POSS(C) = \{K\}$
$BEL(C) = \{\#(K)\}$
$BL(C) =$

Part of ticket granting

Receive(S,$\{N_1, N_2, \cdots N_n\}$)
Split($\{N_1, N_2, \cdots N_n\}$)

Phase I

Generate-secret(SF)
Generate-secret(N)
Encrypt(K,N)
Send(S,Encrypt(Concat(SF, N, N_i[6]), K))
Update($\{SF \cdot N \cdot N_i\}_K, \mathcal{W}$)
Forget-secret(N)
Forget-secret(SF)

Phase II

Receive($S, \{N \cdot \{SF \cdot TGT_C\}_K\}$)
Split($\{N \cdot \{SF\}_K\}$)
Decrypt($\{K\}_N, N$)

[4] For completeness sake, we should also specify to forget $\{SF \cdot N\}_K$ and other formulas that are added to $POSS(C)$ by Concat and Encrypt, but we will omit these from the BL for clarity. They would be included in an actual specification (and their existance helped the author discover a bug in the actual *khat* implementation).

[5] That is, the remaining lifetime of the *khat* ticket from the ticket granting service.

Decrypt($\{SF \cdot TGT_C\}_K, K$)
Split($\{SF \cdot TGT_C\}$)
Check-freshness(TGT_C)

Server

$POSS(S) = \{K\}$
$BEL(S) = \{\#(K)\}$
$BL(S) =$
Part of ticket granting
- Generate-nonce(N_1)
  Generate-nonce(N_2)
  ...
  Generate-nonce(N_n)
  Send(C,Concat($N_1, N_2, \cdots, N_n$))
  Update($\{N_1, N_2, \cdots N_n\}, \mathcal{W}$)

Phase I
Receive($C, \{SF \cdot N \cdot N_i\}_K$)
Decrypt($\{SF \cdot N \cdot N_i\}_K, K$)
Split($\{SF \cdot N \cdot N_i\}$)
Phase II
Generate-secret(TGT_C)
Send(Concat(N,Encrypt(Concat(SF, TGT_C), K)))
Update($\{N \cdot \{SF \cdot TGT_C\}_K\}, \mathcal{W}$)

Now, after the analysis reaches the first `Forget-secret` statement, $BEL(S)$ contains (among other things) $\#(K)$, $LINK(N_2)$, $\cdots$, $LINK(N_n)$, $\#(SF)$, and $\#(N)$. The flaw described earlier no longer exists. If an intruder attempts to replay the message containing the spool file, the server will recognize that the nonce, N_i has already been used. In the analysis, this is reflected by the absence of $LINK(N_1)$ from $BEL(S)$. The linkage rule cannot be applied in this case. Thus, the server will not conclude that the spool file in the replayed message is fresh, and the protocol will be aborted.

We continue our analysis with the client's actions:

- Forget-secret(N)
  Forget-secret(SF)

After these actions, N and SF are removed from $POSS(C)$. Also, the beliefs that C possesses N and SF are removed from $BEL(S)$ because S trusts C according to the trust matrix.

[6] N_i is the first unused nonce in the list received from the server.

At this point phase I is over. It is clear from the values of the *Observers* sets, which are updated with every `Send` action, that nobody can learn the value of SF from the messages sent. Also, the possession set of C represents what an intruder can learn by compromising the workstation while a job is pending. The only useful possession is $\{K\}_N$. Of course, without N, this is useless. Because $Observers(N) = \{S\}$, no intruder can gain anything by compromising the workstation before phase II begins.

To preserve space, we include only the most interesting part of the analysis that remains. When phase II begins, the next three actions are the server's.

- Generate-secret(TGT_C)
  Send(Concat(N,Encrypt(Concat(SF, TGT_C), K)))
  Update($\{N \cdot \{SF \cdot TGT_C\}_K\}, \mathcal{W}$)

After the `Update` action, $Observers(N) = \mathcal{W}$. Thus, if an intruder has compromised the workstation and obtained $\{K\}_N$, then the secrecy of K has also been lost. Thus, analysis reveals that once it is time for the job to run, a previous compromise of the workstation results in an insecure session key. This further results in the compromise of the TGT.

Our analysis reveals a new vulnerability in phase II of *khat*. Although the analysis did not mechanically produce this result, use of our technique generated conclusions from which the vulnerability became apparent. In Section 7 we discuss how to test a protocol for known weaknesses.

6.3 Nessett criticism

In his well known note [13], Nessett criticizes the BAN logic. He presents the following protocol that uses assymetric keys:

$$A \rightarrow B : \{N_a, K_{ab}\}_{K_a^{-1}}$$

$$B \rightarrow A : \{N_b\}_{K_{ab}}$$

The problem is that K_{ab} is encrypted under A's private key. Thus, anyone intercepting the first message can decrypt it with the corresponding public key and obtain the session key.

Although the inference rules needed for public keys are not included in this paper, it is a simple matter to construct them. The global set R of inference rules was intentionally left for the user to specify to provide ways of extending the analysis.

Once the protocol is specified, our analysis immediately reveals the flaw. After the first message is sent, the update function sets $Observers(K_{ab})$ to $\mathcal{W}$ because the $Observers$ set of the public key K_a is $\mathcal{W}$. In addition, B does not believe that K_{ab} is fresh.

Interestingly, our analysis also reveals that in addition to its obvious and intended flaw, the Nessett protocol uses nonces improperly.

7 Analyzing Known Threats

Our specification and analysis technique can also be used to test a protocol against a known attack. This can be done by including the intruder, Z, in the set of principals. $BL(Z)$ contains the actions that the intruder performs. The analysis determines what Z is able to learn during the course of the protocol. The trust matrix can even be used to analyze what happens when Z is actually trusted.

By specifying $BL(Z)$ differently, one can determine whether an intruder could trick a participant into revealing some sensitive information using a given attack. In this sense, a user can interact with the analysis to check a new protocol for given flaws and vulnerabilities.

8 Conclusions

In this paper, we introduce a new method for specifying authentication protocols that offers several advantages over existing methods. The method also includes a logical analysis based on the propagation of belief and knowledge. A fundamental assumption in our threat model is that any message in the system is essentially a broadcast.

We specify protocols as a collection of independent processes. This model closely resembles the structure of the actual distributed system in which the protocols are implemented. Our specifications are designed to resemble the actual implementation as much as possible. This eliminates flaws introduced in the process of converting a specification (which may contain no flaws itself) to an actual program.

One weakness of many analysis techniques that require protocol idealization is that flaws in the protocol may not appear in the idealized version. Thus, the analysis is incapable of revealing them. Our method does not require idealization and thus avoids this problem.

We demonstrate that our method can be used to reason about a new class of protocols for which previous approaches are inadequate. We use actions such as `Forget` and `Forget-secret` along with knowledge and belief sets to reason about nonmonotonicity of knowledge in protocols.

The Needham and Schroeder protocol has become a benchmark used by designers of analysis techniques to test their methods. We demonstrate how the known flaw in that protocol is revealed. In addition, we use our method to uncover a new flaw in our *khat* protocol and to discover a vulnerability in phase II of the protocol. Finally, we show that our method easily uncovers flaws in protocols, such as Nessett's, that methods such as BAN cannot detect.

9 Future Work

The method presented in this paper has been successful to the extent that it discovered a new flaw in the *khat* protocol and also revealed an unknown vulnerability. The analysis would be even better if we could make some claim about the soundness and completeness of the reasoning. One possible way to do this is to define the semantics of the logic. However, it is not clear that these properties hold, and useful semantics of logics of authentication are extremely rare because of the difficulty of defining them.

We would like to use the techniques we have developed to specify and analyze other protocols as well as the ones presented here. In particular, the ability to specify and analyze protocols with nonmonotonicity of knowledge opens the door to a whole new class of protocols. One possible application of this method involves analyzing aspects of public key systems, which rely on forgetting some large primes, in a new way. Previous approaches take for granted that when a principal generates a key, it discards the pieces necessary to reconstruct it. Our method gives a user the flexibility to analyze the system at a finer granularity.

It is our hope that methods such as the one we present in this paper will help in the development of protocols with higher assurance of security.

Acknowledgements

The author thanks Peter Honeyman for his help in editing this paper, guidance, and motivation; Atul Prakash for his excellent advice to pursue this topic; Edna Brenner and Mary Jane Northrop for proofreading; and Paul Syverson for providing valuable reference materials. This work was partially funded by IBM.

References

[1] Martin Abadi and Mark R. Tuttle. A semantics for a logic of authentication. *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[2] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8, February 1990.

[3] Claudio Calvelli and Vijay Varadharajan. An analysis of some delegation protocols for distributed systems. *Proceedings of the Computer Security Foundationn Workshop V*, pages 92–110, 1992.

[4] E. A. Campbell, R. Safavi-Naini, and P. A. Pleasants. Partial belief and probabilistic reasoning in the analysis of secure protocols. In *Proceedings of the Computer Security Foundationn Workshop V*, pages 84–91, Washington, 1992.

[5] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[6] Klaus Gaarder and Einar Snekkenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3:81–98, 1991.

[7] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 234–248, May 1990.

[8] R. Kailar and V. D. Gilgor. On belief evolution in authentication protocols. *Proceedings of the Computer Security Foundation Workshop IV*, pages 103–116, June 1991.

[9] Wenbo Mao and Colin Boyd. Towards formal analysis of security protocols. *Proceedings of the Computer Security Foundationn Workshop VI*, pages 147–158, June 1993.

[10] Louise E. Moser. A logic of knowledge and belief for reasoning about computer security. *Proceedings of the Computer Security Foundation Workshop II*, pages 57–63, 1989.

[11] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[12] R.M. Needham and M.D. Schroeder. Authentication revisited. *Operating Systems Review*, 21:7, January 1987.

[13] D. M. Nessett. A critique of the burrows, abadi and needham logic. *Operating System Review*, 24(2):35–38, April 1990.

[14] A. D. Rubin and P. Honeyman. Long running jobs in an authenticated environment. *USENIX Security Conference IV*, pages 19–28, October 1993.

[15] Einar Snekkenes. Exploring the ban approach to protocol analysis. *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 171–181, May 1991.

[16] Paul Syverson and Catherine Meadows. A logical language for specifying cryptographic protocol requirements. *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 165–177, May 1993.

[17] M. J. Toussaint. A new method for analyzing the security of cryptographic protocols. *Journal of Selected Areas in Communication*, 11(5):702–714, June 1993.

[18] V. L. Voydock and S. T. Kent. Security mechanisms in high–level network protocols. *Computing Surveys*, 15(2):135–171, June 1983.

[19] Thomas Y.C. Woo and Siman S. Lam. A semantic model for authentication protocols. *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 178–194, May 1993.