2007

Extending Distributed Temporal Protocol Logic To A Proof Based Framework For Authentication Protocols

Shahabuddin Muhammad
University of Central Florida

EXTENDING DISTRIBUTED TEMPORAL PROTOCOL LOGIC TO A PROOF BASED FRAMEWORK FOR AUTHENTICATION PROTOCOLS

by

SHAHABUDDIN MUHAMMAD
B.E. NED University of Engg. and Tech., 1999
M.S. University of Central Florida., 2005

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2007

Major Professor: Ratan K. Guha

ABSTRACT

Running critical applications, such as e-commerce, in a distributed environment requires assurance of the identities of the participants communicating with each other. Providing such assurance in a distributed environment is a difficult task. The goal of a security protocol is to overcome the vulnerabilities of a distributed environment by providing a secure way to disseminate critical information into the network. However, designing a security protocol is itself an error-prone process. In addition to employing an authentication protocol, one also needs to make sure that the protocol successfully achieves its authentication goals.

The Distributed Temporal Protocol Logic (DTPL) provides a language for formalizing both local and global properties of distributed communicating processes. The DTPL can be effectively applied to security protocol analysis as a model checker. Although, a model checker can determine flaws in a security protocol, it can not provide proof of the security properties of a protocol. In this research, we extend the DTPL language and construct a set of axioms by transforming the unified framework of SVO logic into DTPL. This results into a deductive style proof-based framework for the verification of authentication protocols. The proposed framework represents authentication protocols and concisely proves their security properties. We formalize various features essential for achieving authentication, such as message freshness, key association, and source association in our framework. Since analyzing security protocols greatly depends upon associating a received message to its source, we separately analyze the source association axioms, translate them into our framework, and extend the idea for public-key protocols. Developing a proof-based framework in temporal logic gives us another verification tool in addition to the existing model checker. A

security property of a protocol can either be verified using our approach, or a design flaw can be identified using the model checker. In this way, we can analyze a security protocol from both perspectives while benefiting from the representation of distributed temporal protocol logic.

A challenge-response strategy provides a higher level of abstraction for authentication protocols. Here, we also develop a set of formulae using the challenge-response strategy to analyze a protocol at an abstract level. This abstraction has been adapted from the authentication tests of the graph-theoretic approach of strand space method. First, we represent a protocol in logic and then use the challenge-response strategy to develop authentication tests. These tests help us find the possibility of attacks on authentication protocols by investigating the originator of its received messages. Identifying the unintended originator of a received message indicates the existence of possible flaws in a protocol. We have applied our strategy on several well-known protocols and have successfully identified the attacks.

To my parents, my wife, and my siblings

ACKNOWLEDGEMENTS

In the name of Allah, the Most Gracious, the Most Merciful. All praises are to Him, the Lord of the heavens and the earth who has guided us and to whom we shall return. I thank Him for His infinite blessings and ask Him to fill my heart, mind, and soul with knowledge and wisdom.

It has been an exceptional journey that I could have ever dreamt of. I feel a deep sense of gratitude for my mother *Nishat Afza* and my father *Saifuddin*. Their love, devotion, continuous support, and prayers paved the path of my success. My words do no justice to how I feel about them and to how much I appreciate them. I am highly indebted to them for their efforts in teaching me moral values, giving me the thirst of knowledge, broadening my vision, and guiding me in every aspect of my life.

A journey is easier when you travel together. My deepest gratitude goes to my wife, *Kishwar Jabeen*, whose continuous support and encouragement made it possible to continue this journey. There is no doubt that she is the reason that I am able to finish my studies. She was not only there for me when I needed moral support, but she supported me technically as well. I am really thankful to her. Furthermore, I really like to thank my brothers, *Sabahuddin* and *Salahuddin*, and my sister, *Khadija Seemi*, who have been a constant source of guidance, love, and support throughout my academic career. Truly, I feel very fortunate to be a member of such a loving and wonderful family.

I am also very thankful to my adviser, *Professor Ratan Kumar Guha*. He has proven himself to be a great mentor during my PhD career. I would like to thank him for giving his students complete freedom in their research, treating them with respect and friendliness, and providing them an ideal

environment for learning and confidence. His classic style in dealing with his students deserves high appreciation.

Last but not least, I must mention my friends with whom I spent great time and learned a lot. Particularly, I would like to thank my friend and colleague *Zeeshan Furqan* who played a vital role during my entire PhD career. I really enjoyed studying all nights long with him, debating for hours and hours in the issues related to our research, and learning from our team-work experience. I would also like to thank my colleagues and all the members of my defense committee, *Dr. Mostafa Bassiouni*, *Dr. Sheau-Dong Lang*, and *Dr. Mainak Chatterjee*, who put their trust in me and made my PhD goal a reality.

In the end, I would like to thank the department of electrical engineering and computer science and the funding agencies who provided me with the resources to continue my work. This work was partially supported by NSF under grant EIA0086251 and ARO under grant DAAD19-01-1-0502.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Distributed systems have been in use for commercial purposes as well as in sensitive domains, such as defense, for many years. E-commerce has also become a widely used successful tool to promote businesses on the internet. In scenarios where critical information is being widely communicated across distributed networks, security becomes an extremely important issue. Before initiating a financial transaction, one needs to make sure that its sensitive data will be securely transmitted to the intended recipient. Several types of encryption schemes have been developed to ensure secure transmission of data. However, history has shown that even assuming ideal cryptography[1], one can not assure that its secrets will safely reach to the intended destination. The reason for this lack of security is not the underlying cryptography, but the security holes in the cryptographic protocols themselves.

A security protocol (or cryptographic 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 conversations [NS78]. The network is assumed to be hostile as it contains intruders with the capabilities to encrypt, decrypt, copy, forward, delete, and so forth. Considering an active intruder with such powerful capabilities, it becomes extremely difficult to guarantee proper working of a security protocol. Several examples show how carefully designed protocols were later found out to have security breaches [MFG06b]. This situation led the researchers to formalize the verification of security protocols.

[1] Ideal cryptography means that a principal must have a key in order to perform any cryptographic operation (such as encryption, decryption, signatures) using that key.

Logic-based verification is one of the widely used formal verification techniques in the domain of security protocols. This is due to both, the simplicity of the logic-based methods and the conciseness of the proof they generate [CDF03]. The logic of belief of [BAN90], known as BAN logic, provided the initial impetus in applying modal logic in a proof-based environment. Analyzing a protocol in BAN begins by first formalizing the message exchanges of the protocol in the language of its logic. Then all the initial assumptions of the protocol and assertions about each message exchange are written. Finally, BAN rules are applied on the assumptions and the assertions in order to derive the conclusion. Although BAN logic has also been criticized for various reasons explained in the next section, it has been successfully used to discover various attacks on well-known protocols. With the emergence of BAN, several researchers have applied various logic-based techniques for the formal verification of security protocols. BAN logic has also been extended in many ways [GNY90, Oor93, SV93, AT91].

Recently a new logic, called distributed temporal protocol logic (DTPL), has been proposed in [CVB05a] which provides an object-level tool to model distributed communication. DTPL's distinguishing characteristics is its capability to be used as a metalevel tool for comparative analysis of security protocol models and properties. In this thesis, we have developed a BAN-like proof system based on distributed temporal protocol logic. We devise a logic-based verification system that involves the notion of derivability (i.e. proofs) in which a formula φ is derivable from a set of formulae Γ using the inference rules. Since DTPL provides an intuitive framework that captures reasoning in a distributed environment, we utilize the existing DTPL and develop it such that it can be used to verify security protocols in a proof-based setting. For this purpose, we take advantage from the work of [SC01] (SVO logic). The reason for using SVO is that it clarifies many

concepts in previous logics and unifies four of its predecessors in a sound way. We demonstrate how DTPL incorporates the essential features for protocol verification of SVO in its framework. We also describe how DTPL helps us clarify some concepts, such as freshness, in the existing logic-based techniques. As in other logic-based techniques, we see security protocols to possess essential ingredients to guarantee authentication, such as message freshness, key association, message origination etc. We represent these ingredients in terms of DTPL. Authentication protocols can be broadly categorized based on different types of cryptography they use and on how they achieve authentication goal. In particular, when finding the originator of a received message, we categorize authentication protocols into symmetric-key, asymmetric-key, and challenge-response protocols.

Our verification framework contains the advantages of both, the expressibility of the existing DTPL model as well as the conciseness of a proof-based system. Moreover, due to DTPL's rich interpretation structure of Winskel [Win87], analyzing a protocol becomes clearer as compared to other logic-based techniques. Since our work is based on the notion of derivability, the verification process does not explicitly model an intruder, thereby obviating the need to apply all combinations of intruder behaviors for protocol analysis. This results in considerable simplicity in the way a proof-based method analyzes a protocol. In addition to developing the existing DTPL framework for three types of protocols, we demonstrate the applicability of our work by applying it on well-known protocols.

We have also used the distributed temporal protocol logic in order to analyze security protocol at a higher level of abstraction. In particular, we analyze a protocol by first representing the run of each participant of the protocol in terms of its corresponding life-cycle. Next, we try to achieve

match in the parameters among the participants of a protocol. Authentication is achieved if critical parameters of a participant matches with the parameters of the rest of the participants of a protocol. In the effort to find the matching parameters, each participant investigates its own life-cycle and tries to find out the originators of each of its received messages. More specifically, each participant initiates a challenge and waits for the response of that challenge in order to assure the identity of the originator of its received messages. We use this challenge-response criterion to develop simple tests like the one developed in another framework of [GF02]. Given these tests, one can determine the identity of the originator of the response of a challenge generated by a participant. A protocol fails to achieve authentication either if we find a mismatch in the parameters among the participants, or if we find an unintended originator of a received message at any participant's local life-cycle. The resulting verification strategy not only contains the expressibility of the existing DTPL model but it also provides a concise tool that can be used as a heuristic to investigate authentication protocols. In order to demonstrate the applicability of our work, we analyze a well-known protocol using our method and compare it with the existing DTPL model.

The rest of this thesis is organized as follows. Chapter 2 describes a brief survey of efforts in the field of formal verification of cryptographic protocols. Chapter 3 discusses the notion of a secure system and the issues related to it. In particular, we discuss authentication protocols, their properties, and the design goals that should be focused on in order to achieve a secure system. We discuss our proposed scheme in Chapter 4. In this chapter, we first present a brief introduction of the Distributed Temporal Protocol Logic. Then we describe how this logic can be used to develop a proof-based verification framework. Next, we apply our proposed framework on well-known protocols in Chapter 5. More specifically, we apply the proposed framework on the Needham-

Schroeder Secret-Key and the PKINIT protocols. Finally, we extend our framework in order to derive a higher level authentication tool in Chapter 6. The conclusion is followed in Chapter 7.

CHAPTER 2
RELATED WORK

Probably the authors Needham and Shroeder in [NS78] were the first to mention the need for techniques to formally verify the correctness of security protocols. Formal methods are useful for the analysis of security protocols. They allow one both to do a thorough analysis of the different paths which an intruder can take, and to specify precisely the environmental assumptions that have been made [Mea03].

Formal verification of system properties relies on having a complete description of the system under consideration. For security protocols, this implies modeling of both the communicating parties and the potential penetrator. A cryptographic protocol is required to achieve its goals in the presence of saboteur. Designers should foresee all the possible attacks on the protocol under development. Formalism has been applied to a wide range of cryptographic protocols in order to verify security properties.

All formalism based strategies begin by expressing the cryptographic protocol in a formal notation or model and then proving that the expressed model achieves its security goals. The verification process can be distinguished in two major categories, theorem proving and model checking. In the first method, logic is used along with formal proof. Mathematical proof is considered as the strongest argument to guarantee the correctness of a system. This method is build upon traditional mathematical reasoning. Existing logical notations are utilized with the addition of new notations to express the formal model into logic. Theorem proving techniques are applied on the resulting model to prove its correctness. These proofs are complex, challenging, and require the presence of mathematical experts.

Once a formal model of a system has been established, model checking can be utilized to establish the accuracy of the system. Model checking is an alternative resort to theorem proving. A model checker is a tool that explores the state space of the model to determine if there are any paths through the space that corresponds to a successful attack. There have been several attempts to model checking for security protocol verification. Almost all approaches use similar penetrator model. The main difference is in the ways one specifies a protocol and its properties, and how a particular model checker performs the analysis. Below we summarize the work that has been done in the field of formal verification of cryptographic protocols.

2.1 Dolev-Yao Model

The significant early work on formal verification of cryptographic protocols is contributed to Dolev and Yao [DY83]. This work pioneered in modeling a penetrator and defining its abilities such as encryption and decryption(by some keys a penetrator possesses), message forwarding, copying, deleting and so on. Since then, virtually all formal verification techniques use the same penetrator model.

The work was mainly focused for the protocols that transmit a secret plaintext M between two users. So the secrecy property was the primary target instead of the authentication property even though they are closely linked with each other. Three example protocols were mentioned in order to clarify how to achieve security in a protocol. Moreover, it was stressed that adding additional layers of encryption did not necessarily increase the security of a protocol. Rather, sometimes it is detrimental to the very purpose of the design. The authors presented precise mathematical models for two classes of protocols named "*the cascade protocols*" and "*the name-stamp protocols*". The

cascade protocols contain encryption and decryption as the only operations performed on messages whereas the name stamp protocols allow users to append, delete and check names encrypted together with the plaintext.

The authors suggest the following two conditions to be necessary in order to assure the security of cascade protocols:

1. The messages transmitted between two participants must contain some layers of encryption functions.

2. In generating a reply message, each participant A never applies decryption function D_A without also applying encryption function E_A.

Similarly, for a two party name stamp protocol T, the authors define that T is insecure if a string $\gamma \in V^*\overline{N_i(X,Y)}$ exists such that $\overline{\gamma} = \lambda$. Here V is the string of operators that an intruder can apply on any message and $\overline{N_i(X,Y)}$ is the sequence of texts transmitted between X and Y, when X wishes to send plaintext M to Y. Otherwise T is defined to be secure. Consider the following two party name stamp protocol:

$$1.\ X \rightarrow Y \quad : \quad (X, E_y(E_y(M)X), Y)$$

$$2.\ Y \rightarrow X \quad : \quad (Y, E_x(E_x(M)Y), X)$$

The authors have proven this protocol to be flawed. For the above mentioned protocol, the sequence of operators applied by participant X in the first message $= \overline{N_1(X,Y)} = E_Y i_X E_Y$. That is, the participant X first applies encryption E_Y (using Y's public key) on message M, then appends his own id X (append operation i_X), and finalizes the message by applying the encryp-

tion operation E_Y again. Similarly, in the second message the participant Y applies the operators $\overline{N_2(X,Y)} = E_X i_Y E_X$. The authors also defined additional operators such as d for deleting an id from a string, d_X for deleting a known id X from a string, and D_X for the decryption using X's private key. The attack on this protocol exists as the authors found the string $\gamma = D_Z d D_Z E_Z i_X E_Z D_X d_Z D_X E_X i_Z d D_Z d D_Z E_Z i_X E_Z D_X d_Z D_X E_X i_Z \overline{N_2(X,Y)} \in V^* \overline{N_i(X,Y)}$ such that $\overline{\gamma} = \lambda$. γ simply represents the sequence of operators a saboteur applies on a protocol message resulting into a null string. In this way, a saboteur can obtain the secret plaintext from an encrypted message.

2.2 The Logic of BAN

BAN [BAN90] is a logic of belief which was the first attempt in applying modal logic to verify security protocols. Since security protocols involve principals sending and receiving messages to each other, each principal holds certain beliefs about these messages. For example, a principal A believes that a message M is fresh. In BAN notations, it is represented by $A \mid\equiv \#(M)$. Since the notations used in BAN are non-intuitive, we replace those notations by their meaning in plain English as done in [AT91] and [SC01]. Therefore, we replace $A \mid\equiv \#(M)$ by A *believes* $fresh(M)$. Principals also hold beliefs about communication, such as A *believes* B *said* M, A *believes* B *sees* M and so on. Some of the other expressions of the language of BAN are A *controls* M (A is trusted on the values of M), $A \overset{k}{\longleftrightarrow} B$ (k is a shared key between A and B), and $\overset{k}{\mapsto} A$ (k is a public-key of A). BAN analyzes a protocol by following a sequence of steps and applying a set of rules. We will explain these steps with the help of an example in this section. Some of the important BAN rules include:

Message Meaning Rule: This rule states that if A believes k to be a shared-secret between him

and B and he received a message encrypted by k, then he is entitled to believe that B said M. This rule is used to find out the sender of a received message such that some principals already share some secrets.

$$\frac{A\ believes\ A \overset{k}{\longleftrightarrow} B \qquad A\ received\{M\}_k}{A\ believes\ B\ said\ M}$$

Public-key equivalent of this rule can also be written in the similar way.

Nonce Verification Rule: It states that if A believes a message M to be fresh and he believes that a principal B said that message sometime in the past, then he is entitled to believe that B still believes in M (because of its freshness). This rule is used to make sure that principals do not become victim of replay attack in which an intruder replays an old message.

$$\frac{A\ believes\ fresh(M) \qquad A\ believes\ B\ said\ M}{A\ believes\ B\ believes\ M}$$

Jurisdiction Rule: It states that if a principal believes in a message M such that he has authority over M then M is believable. It is mainly used for servers who are responsible for generating keys for other principals.

$$\frac{A\ believes\ B\ controls\ M \qquad A\ believes\ B\ believes\ M}{A\ believes\ M}$$

The above-mentioned rules provide the main machinery in achieving security goals, such as authentication, in BAN logic. To facilitate the goal derivation, BAN also provides some other rules as given below.

A believes the concatenation and concatenates of its believed messages[1]:

$$\frac{A\ believes\ X \quad A\ believes\ Y}{A\ believes\ (X, Y)} \qquad \frac{A\ believes\ (X, Y)}{A\ believes\ X}$$

A believes that the other principals believe in the concatenates of their concatenated messages:

$$\frac{A\ believes\ B\ believes (X, Y)}{A\ believes\ B\ believes\ X}$$

A holds B responsible for saying all the concatenates of a said message:

$$\frac{A\ believes\ B\ said (X, Y)}{A\ believes\ B\ said\ X}$$

A message is fresh if any of its concatenates is fresh:

$$\frac{A\ believes\ fresh(X)}{A\ believes\ fresh(X, Y)}$$

Given that a principal holds the corresponding decryption key, contents of an encrypted received message are also considered to be received:

$$\frac{A\ believes\ A \overset{k}{\longleftrightarrow} B \qquad A\ received \{M\}_k}{A\ received\ M}$$

[1] In BAN, a comma is used to represent concatenation of two messages X and Y, i.e. X, Y.

The above rule can also be extended for public-key encryption and public-key signature as follows:

$$\frac{A\ believes(\overset{k}{\mapsto} A) \quad A\ received\{M\}_k}{A\ received\ M} \qquad \frac{A\ believes(\overset{k}{\mapsto} B) \quad A\ received\{M\}_{k^{-1}}}{A\ received\ M}$$

Here, k represents a public key whereas k^{-1} represents a private key in asymmetric cryptography. In symmetric cryptography, $k = k^{-1}$. Concatenates of a concatenated received message are also believed to be received:

$$\frac{A\ received(X, Y)}{A\ received\ X}$$

Given the above rules, BAN logic analyzes a protocol using the following steps.

1) Idealize a protocol (explained by the following example).

2) State all the initial assumptions of the protocol.

3) For each message transmission in the protocol of the form $A \rightarrow B : M$, write assertion of the form $B\ received\ M$.

4) Apply BAN rules on the assumptions and assertions to derive beliefs held by other principals of the protocol.

We use a very simple protocol to demonstrate the application of BAN logic. The "Wide-Mouthed Frog protocol" has been analyzed using BAN in [BAN90]. The protocol comprises of only two steps as shown in Fig. 2.1. In the first message, a principal A sends a session key k_{AB} along with a timestamp T_a to S. After checking the timeliness of the first message, S adds its own timestamp and sends the second message to B.

Step 1: The idealized protocol is given below.

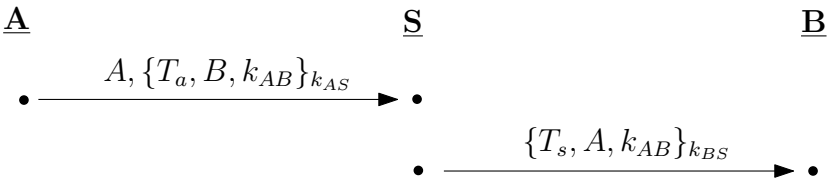

Figure 2.1: The Wide-Mouthed Frog Protocol

Message 1. $A \rightarrow S : \{T_a, (A \overset{k_{AB}}{\longleftrightarrow} B)\}_{k_{AS}}$

Message 2. $S \rightarrow B : \{T_s, A\ believes (A \overset{k_{AB}}{\longleftrightarrow} B)\}_{k_{BS}}$

Notice that the idealization step not only requires the understanding of the working of the protocol, but it also demands announcing the corresponding beliefs of a principal at the time of sending its messages. As in the above, the first message is idealized such that $A \overset{k_{AB}}{\longleftrightarrow} B$ (A's belief about k_{AB}) replaces B, k_{AB}. Similarly, second message attaches the S's belief about k_{AB}. Also note that plaintext messages (A in the first message) are not part of the idealization.

Step 2: The initial assumptions are stated as follows. [BAN90] calls some of the assumptions as dubious as explained in Step 4.

$A\ believes\ A \overset{k_{AS}}{\longleftrightarrow} S$ $\qquad$ $B\ believes\ B \overset{k_{BS}}{\longleftrightarrow} S$

$S\ believes\ A \overset{k_{AS}}{\longleftrightarrow} S$ $\qquad$ $S\ believes\ B \overset{k_{BS}}{\longleftrightarrow} S$

$S\ believes\ fresh(T_a)$ $\qquad$ $B\ believes\ fresh(T_s)$

$B\ believes\ A\ controls\ A \overset{k}{\longleftrightarrow} B$

$B\ believes (S\ controls (A\ believes\ A \overset{k}{\longleftrightarrow} B))$

Step 3: Assertions can be easily written using step 1 as follows.

$S\ received\ \{T_a, (A \overset{k_{AB}}{\longleftrightarrow} B)\}_{k_{AS}}$

$B\ received\ \{T_s, A\ believes (A \overset{k_{AB}}{\longleftrightarrow} B)\}_{k_{BS}}$

Step 4: Now the protocol analysis is almost trivial as we simply need to apply the BAN rules using

the assumptions and the assertions stated above. We omit the detailed derivation steps and write the conclusion as follows.

S *believes* A *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$

A *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$

B *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$

B *believes* A *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$

Observe that the above derivation became possible when we used some strange assumptions in step 2. The most dubious assumption is that B believes A to generate good keys in B *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$ as also mentioned in [BAN90]. The authors in [BAN90] claim a protocol to be secure in BAN logic only if it abides by the assumptions taken during the analysis. That is why, the use of this protocol is restricted to only those scenarios where A represents a trusted and competent authority on generating good session keys.

So far, we have presented a brief account on the logic of belief and its application. Now we present some of the extensions of BAN logic.

2.3 BAN Extensions

With the emergence of BAN, researchers have suddenly realized the potential of applying logic-based techniques for the formal verification of authentication protocols. Even though BAN has successfully identified flaws in some well-known protocols [BAN90], it has been rigorously analyzed for potential weaknesses and several corrective measures have been suggested. The immediate successor of BAN was a logic by Gong et. al. in [GNY90] by the name GNY logic. One of the improvements in GNY was to introduce the notion of *recognizability* which captures the recipient's

expectation of the contents of a message before actually receiving it. For example, a principal may recognize a particular structure of a message or any form of redundancy in the message. This is in contrast to BAN which assumed that redundancy is always present in encrypted messages. GNY also introduced the notion of *not-originated-here* to identify if a principal receives his own conveyed messages. In addition to extending the applicability of BAN to a wider range of protocols, GNY also separated the notion of possession and beliefs. This allowed one to treat content of a message and the information implied by a message separately because a principal may possess a value but may not believe in it.

Abadi et. al. in [AT91] contributed towards providing new semantics to the logic of BAN. In search of providing a sound semantics, they claimed to have identified many sources of confusion in the original work of BAN. For instance, the authors have given possible-world definition of 'belief' as a form of resource-bounded, defeasible knowledge. They reformulated the BAN logic, called AT logic, and proved their axiomatization to be sound with respect to their model of computation and semantics. The AT logic was closer to traditional modal logics than BAN [SC01].

Paul van Oorschot extended the GNY logic to reason about protocols that involve Diffie-Hellman type key agreement [Oor93]. It was called VO logic. It can be seen that various successors of BAN logic tried to extend BAN in various aspects. Observing this diversity in the BAN suite of logics, the need was felt to come up with a logic which should be sound with respect to its model and which could unify its predecessors. SVO logic of [SO94] is such a logic which unifies four of its predecessors, BAN, GNY, AT, and VO. Next, we briefly describe the SVO logic.

2.4 The logic of SVO

The aim of the SVO logic was to unify four of its predecessors in a sound (with respect to its computational model) way. We briefly describe the SVO logic given in [SC01] as follows.

2.4.1 SVO Notations

In addition to the BAN notations as describe above, SVO uses the following notations.

$\neg\varphi$: SVO added the negation of formulae into its language.

A says X: It represents whatever is said in the current run of a protocol.

A has X: It represents all the initial messages of A plus all the messages that A has received, freshly generated, or can construct using these messages.

$PK(A, k)$: SVO represents BAN notation for public keys ($\overset{k}{\mapsto} A$) by $PK(A, k)$. Furthermore, it splits it into three different kinds of public-keys as $PK_\psi(A, k)$, $PK_\sigma(A, k)$, and $PK_\delta(A, k)$. $PK_\psi(A, k)$ represents public ciphering key of A. Only A can read messages encrypted with k. $PK_\sigma(A, k)$ represents public signature key of A. k verifies that a message signed by k^{-1} is from A. $PK_\delta(A, k)$ represents public key-agreement key of A. A Diffie-Hellman key formed with k is shared with A. For a detailed account on Diffie-Hellman key agreement, see [DH76].

$\lfloor X \rfloor_k$, $\{X\}_k$: SVO separately represents signatures $\lfloor X \rfloor_k$ and encryptions $\{X\}_k$. When used in signatures, k represents a private-key in $\lfloor X \rfloor_k$.

$\langle X \rangle_{*A}$: It is used if A can not recognize a message (e.g., $\{X\}_k$ if A does not know k). However, A will recognize $\langle X \rangle_{*A}$ as the same thing if received again even if it can not decrypt the message.

$X\ from\ A$: To represent a message is coming from A.

2.4.2 SVO Inference Rules

SVO uses only two inference rules, Modus Ponens and Necessitation, given as follows.

Modus Ponens: From $\vdash \varphi$ and $\varphi \rightarrow \psi$ infer $\vdash \psi$.

Necessitation: From $\vdash \varphi$ infer $\vdash A\ believes\ \varphi$.

Axioms of the logic are all instances of tautologies of classical propositional calculus, and all instances of the following axiom schemata.

2.4.3 SVO Axioms

Belief Axioms: The following are the classic axioms of modal logic.

1. $(A\ believes\ \varphi \wedge A\ believes\ (\varphi \rightarrow \psi)) \rightarrow A\ believes\ \psi$

2. $A\ believes\ \varphi \rightarrow \varphi$

3. $A\ believes\ \varphi \rightarrow A\ believes\ (A\ believes\ \psi)$

4. $\neg(A\ believes\ \varphi) \rightarrow A\ believes\ (\neg\ A\ believes\ \psi)$

These axioms represent that a principal believes the logical consequence of its beliefs, a principal's beliefs are always true, a principal can tell what it believes, and a principal can also tell what it does not believe[2].

Source Association Axioms: These axioms associate a principal who is responsible for sending an encrypted/signed message. It is called message meaning rule in BAN.

5. $(A \overset{k}{\longleftrightarrow} B \wedge C\ received\{X\ from\ B\}_k) \rightarrow (B\ said\ X \wedge B\ has\ X)$

6. $(PK_\sigma(A,k) \wedge B\ received\ X \wedge SV(X,k,Y)) \rightarrow A\ said\ Y$

[2]In modal logic, these axioms are named **K**, **T**, **4**, and **5** respectively and are known as axioms of the Lewis system **S5** [Che80].

Note that here 'believes' operator is separated from the axiom. Moreover, in the axiom for public-keys, $SV(X, k, Y)$ means that applying k to X confirms that X is the result of signing Y with a private cognate of k.

Key Agreement Axioms: This axiom captures Diffie-Hellman like key agreement. Diffie-Hellman key agreement is an important component in widely used authenticated key established protocols such as IETF standard Internet Key Exchange (IKE) protocol [DH99]. The axiom states that session keys that are the result of good key-agreement keys are good.

7. $(PK_\delta(A, k_A) \wedge PK_\delta(B, k_B)) \rightarrow A \overset{F_0(k_A,k_B)}{\longleftrightarrow} B$

8. $\varphi \equiv \varphi[F_0(k, k')/F_0(k', k)]$

$F_0(k', k)$ represents function that combines k' with k^{-1} to form a shared key.

Receiving Axioms: These axioms state that the concatenates of a concatenated message and the contents of an encrypted or a signed message are also viewed as received messages.

9. $A\ received(X_1, \ldots, X_n) \rightarrow A\ received\ X_i$, for $i = 1, \ldots, n$.

10. $(A\ received\{X\}_k \wedge A\ has\ k^{-1}) \rightarrow A\ received\ X$

11. $(A\ received\lfloor X \rfloor_k) \rightarrow A\ received\ X$

Axiom 11 assumes that principals possess public keys.

Possession Axioms: A principal possesses all of its received messages plus any message obtained as a result of applying any computable function (e.g., encryption, signatures, etc.) on existing messages.

12. $A\ received\ X \rightarrow A\ has\ X$

13. $A\ has(X_1, \ldots, X_n) \rightarrow A\ has\ X_i$, for $i = 1, \ldots, n$.

14. $(A\ has\ X_1 \wedge \ldots \wedge A\ has\ X_n) \rightarrow A\ has\ F(X_1, \ldots, X_n)$

Comprehension Axiom: This axiom basically represents recognizability axiom of GNY [GNY90]. That is, a principal recognizes a function of a message only if he knows the message itself.

15. $A\ believes(A\ has\ F(X)) \rightarrow A\ believes(A\ has\ X)$

Saying Axioms: The following axioms holds a principal responsible for saying each component of a concatenated message. A principal who recently says X has said X.

16. $A\ said(X_1, \ldots, X_n) \rightarrow A\ said\ X_i \wedge A\ has\ X_i$, for $i = 1, \ldots, n$.

17. $A\ says(X_1, \ldots, X_n) \rightarrow (A\ said(X_1, \ldots, X_n) \wedge A\ says\ X_i)$, for $i = 1, \ldots, n$.

Freshness Axioms: These axioms state that a message containing any fresh component is also fresh.

18. $fresh(X_i) \rightarrow fresh(X_1, \ldots, X_n)$, for $i = 1, \ldots, n$.

19. $fresh(X_1, \ldots, X_n) \rightarrow fresh\ F(X_1, \ldots, X_n)$, where F is any computable function which depends upon all of its arguments.

Jurisdiction Axiom: A principal having authority on some messages is always right about those messages.

20. $(A\ controls\ \varphi \wedge A\ says\ \varphi) \rightarrow \varphi$

Nonce Verification Axiom: Something if said in the past such that it is fresh, is as if it is said in the current run of a protocol.

21. $(fresh(X) \wedge A\ said\ X) \rightarrow A\ says\ X$

Symmetric Goodness Axiom: Symmetric-keys are equivalently good between two principals as follows.

22. $A \overset{k}{\longleftrightarrow} B \equiv B \overset{k}{\longleftrightarrow} A$

The above-mentioned SVO axioms model a broad range of security protocols and provide a

unified framework to analyze a protocol in proof-based setting. Since our work also focuses on developing a proof-based verification environment for security protocols, we will use these SVO axioms as a reference when we develop our framework in Chapter 4.

2.5 FDR

Gavin Lowe used Failures Divergences Refinement (FDR) [Ltd93] checker, a model checker for CSP, to analyze cryptographic protocols [Low96]. All participants in a protocol are modeled by CSP processes. Channels are defined that represent standard communication channel *comm* as well as channels to capture intruder capabilities like *fake* and *intercept*. Moreover, additional channels are defined like *I_running* and *I_commit* to represent that the initiator is running the protocol with a responder, and the initiator is committing a session respectively. The intruder is also modeled as having several capabilities. At any instant, the state of the intruder can be parameterized by the knowledge it has acquired.

In order to exemplify the method, we take the example of Needham-Schroeder public-key protocol. Assume the sets *Initiator* respresents initiators, *Responder* represents responders, *Key* represents public keys, and *Nonce* represents nonces. Also assume $a, a' \in Initiator, b \in Responder, k \in Key, n_a, n_b \in Nonce$. Then the three messages in NS protocol can be represented using three sets

of communication events as follows.

$$
\begin{aligned}
MSG1 &\triangleq \{Msg1.a.b.Encrypt.k.n_a.a'\} \\
MSG2 &\triangleq \{Msg2.b.a.Encrypt.k.n_a.n_b\} \\
MSG3 &\triangleq \{Msg3.a.b.Encrypt.k.n_b\} \\
MSG &\triangleq MSG1 \cup MSG2 \cup MSG3
\end{aligned}
$$

The channels are defined as follows. "channel $comm, fake, intercept : MSG$" represent standard communication channel, intruder's faking messages, and intruder's intercepting the messages. These channels assumes MSG as their type. Moreover, additional channels "channel $user, session, I\_running, R\_running, I\_commit, R\_commit : Initiator.Responder$" are the channels of the type $Initiator.Responder$ that represent a user's request to connect the initiator and the responder, a session channel, initiator's taking part in a run of the protocol, responder's taking part in a run of the protocol, initiator committing to a session, and the responder committing to a session respectively.

A CSP process $INITIATOR(a, n_a)$ represents an initiator with identity a and nonce n_a. Without

intruder action, this process is defined as follows.

$$
\begin{aligned}
INITIATOR(a, n_a) \quad \triangleq \quad & user.a?b \rightarrow I\_running.a.b \rightarrow \\
& comm!Msg1.a.b.Encrypt.key(b).n_a.a \rightarrow \\
& comm.Msg2.b.a.Encrypt.key(a)?n'_a.n_b \rightarrow \\
& if n_a = n'_a \; then \; comm!Msg3.a.b.Encrypt.key(b).n_b \rightarrow \\
& I\_commit.a.b \rightarrow session.a.b \rightarrow Skip \\
& else \; Stop
\end{aligned}
$$

Now renaming is applied in order to cater the intruder who can intercept messages 1s and 3s and can fake messages 2s. The resulting initiator is given as follows.

$$
\begin{aligned}
INITIATOR1(a, n_a) \quad \triangleq \quad & INITIATOR(A, N_a) \\
& [[comm.Msg1 \leftarrow comm.Msg1, comm.Msg1 \leftarrow intercept.Msg1, \\
& comm.Msg2 \leftarrow comm.Msg2, comm.Msg2 \leftarrow fake.Msg2, \\
& comm.Msg3 \leftarrow comm.Msg3, comm.Msg3 \leftarrow intercept.Msg3]]
\end{aligned}
$$

The responder can also be defined similarly. The intruder is defined such that it can fake all the messages using its knowledge base. It can also intercept all the message and learn new nonces if it possesses the right decryption key.

In order to test whether any protocol meets its authentication goal, FDR takes two inputs, a specification and an implementation. FDR then tests whether the implementation refines the

specification. The system is defined as the parallel composition of the agents and the intruder, synchronizing on the set of channels. The system is represented as follows:

$$AGENTS \triangleq INITIATOR1 |[\{|comm, session.A.B|\}]| RESPONDER1,$$

$$SYSTEM \triangleq AGENTS |[\{|fake, comm, intercept|\}]| INTRUDER.$$

The above mentioned system represents parallel composition of all the processes in the system. Running the system results into several traces. It is to be checked now that each trace of the implementation is also a trace of the specification. The specification for authentication of a responder AR is given as:

$$AR_0 \triangleq R\_running.A.B \rightarrow I\_commit.A.B \rightarrow AR_0$$

$$A_1 \triangleq \{|R\_running.A.B, I\_commit.A.B|\}$$

$$AR \triangleq AR_0 ||| RUN(\Sigma \backslash A_1)$$

AR_0 means that an $I\_commit.A.B$ event should only occur after an $R\_running.A.B$ event. If Σ is the set of all events, then $AR_0 ||| RUN(\Sigma \backslash A_1)$ represents occurring of all events in an arbitrary order. The above specification says that the initiator A commits to a session with the responder B only if the responder has really taken part in the protocol run. FDR can now be used to verify that *SYSTEM* refines AR, indicating that the protocol correctly authenticates the responder.

2.6 NRL Protocol Analyzer

Meadows NRL Analyzer in [Mea96] is a special purpose verification tool for the analysis of cryptographic protocols. It is a logic-based approach written in Prolog. It models the protocol as an interaction between a set of state machines and attempts to prove a protocol secure by specifying insecure states and attempting to prove them unreachable. The analyzer uses exhaustive search backwards from the insecure state or by the use of proof techniques for reasoning about state machine models. The analyzer considers the domain in the form of word problem - a version of the term-rewriting model of Dolev and Yao [DY83]. The Dolev-Yao model assumes that there is some set of words (for example some secret keys possessed by honest participants, or some encrypted messages) that the intruder does not know. The intruder's goal is to find out these words. Moreover, in NRL protocol analyzer, each participant of the protocol contains its own local state. The global state of the system is simply the composition of these local states with some state information for the environment or the penetrator. Each participant maintains some learned facts $lfacts$ in its local store. For example if user A attempts to initiate a conversation with user B during local round N at time T, then the corresponding $lfact$ is represented as follows.

$$lfact(user(A), N, init\_conv, T) \quad = \quad [user(B)]$$

Similarly if user B receives a message X during local round S at time P apparently from user A attempting to initiate a conversation, then the corresponding $lfact$ is represented as follows.

$$lfact(user(B), S, rcvd\_init\_conv, P) \quad = \quad [user(A), X]$$

The NRL protocol analyzer uses unification in which an incomplete state description represents a set of states. The steps of the protocol are represented as conditional rewriting rules and the goals are formalized as unreachability theorems.

Meadows uses the NARROWER algorithm [PL85] that begins with a trivial set of words that an intruder possesses. A set of secrets is defined which the intruder is not supposed to know. Then the algorithm uses induction on the length of path in an attempt to find any path, starting from the initial state, such that the intruder may learn the words in the secret set. The system also models the knowledge and belief of the intruder and defines a set of rules whereby an intruder can learn new information based on protocol steps. The NRL protocol analyzer defines a set of requirements using some pre-defined actions that specifies a class of protocols. For instance, consider the following requirement that contains two conditions:

- $\neg(\diamond\, accept(B, A, M, N) \wedge \diamond\, learn(Z, M))$
- $accept(B, A, M, N)) \rightarrow \diamond(send(A, B, (Query, M)) \wedge \diamond\, request(B, A, Query, N))$

The first condition says if participant B accepted a message M from participant A at some point in the past (the past time operator $\diamond$), then the intruder did not learn M at some point in the past. Second condition says that if B accepted message M from A in B's local round N then A sent M to B as a response to a query at B's local round N. After the transition rules are defined for honest agents and the operations available to all agents are described, the atoms needs to be defined that serves as the basic building block of the words in the protocol. Finally, the rewrite rules are described. An example of a rewrite rule is given below which says that encryption and decryption with the same key are self canceling.

$rr1 : pke(privkey(X), pke(pubkey(X), Y)) \Rightarrow Y$

$rr2 : pke(pubkey(X), pke(privkey(X), Y)) \Rightarrow Y$

The tool needs high level of expert user interaction. It performs backward search from some insecure state. If the initial state is found, then the path to the initial state represents the counterexample.

2.7 The Inductive Approach

Paulson in [Pau98] introduced the inductive approach with automated support provided by his Isabelle proof assistant. Traces are described as the sequence of events that could occur as the protocol agents execute in a hostile environment. Traces are defined inductively from a set of rules that correspond to the possible actions of the agents, including spies. Paulson uses inductive definitions that list the possible actions that an agent or system can perform. Then the induction rule is used to reason about the consequences of an arbitrary finite sequence of actions. The attacker is modeled using inductively defined operators *analz* and *synth*.

The operator *partsH* represents the set of all components of H that can be obtained from it. The set *analzH* represents the most that could be obtained from H without breaking ciphers. The set *synthH* represents the set of messages a penetrator can build up from the elements of H. Only the known messages (or elements) can be used to build up new messages.

A protocol is described in terms of events of different forms. Two forms of events in a trace are defined: $Says\ A\ B\ X$, and $Notes\ A\ X$, which means respectively that A sends message X to B and A stores X internally. Three additional rules are defined in order to capture the notion of empty trace (an empty list []), fake messages (if $X \in synth(analzH)$ is a fraudulent message and $B \neq Spy$

then, $Says\ Spy\ B\ X$) and accidents (if S distributed the session key K in a run involving the nonces Na and Nb, then $Notes\ Spy\ \{Na, Nb, K\}$). Induction is applied on the set of traces. For the set of traces, the induction principle says that $P(evs)$ holds for each trace evs provided property P is preserved under all the rules for creating traces. $P[]$ is proved to cover the empty trace. For each of the other rules, an assertion of the form $P(evs) \Rightarrow P(ev\#evs)$ is proved where event ev, containing the new message, is added to the trace evs.

The requirements to check a security property of a protocol is given in a syntax identical to that used to model the protocol. For instance, in the case of Needham-Schroeder public-key protocol, the requirement may be that if an initiator A sends the nonce N_a to the responder B in its first message and receives the second message back that contains N_a, then B must have sent this message. Unlike other model checking approaches, Paulson's approach is totally based on theorem proving. Because of its inductive nature, Paulson's method places no limit on the number of instances and an arbitrary number of instances can be considered with this approach. However, being a theorem prover, it can not generate counterexamples in case of a failure and there is no guarantee of termination.

2.8 Murϕ

Murϕ in [MMS97] is an example of general purpose model-checker in which global state variables along with some shared variables (to model communication) are used to represent the state of the system. The transition rules describe the change between honest agent's states and the addition of new messages into the network. The rules that capture the behavior of the penetrator are constructed. Since the description should be finite state, it cannot represent infinite behavior of the

penetrator. That is, the penetrator can learn only finite number of words specific to the particular protocol. The specification for the protocol is represented by an invariant on the reachable global states of the system. It is difficult to specify secrecy property in this approach because of the problem in keeping track of the knowledge of each participant of the protocol.

The Murϕ language is a high-level language for describing nondeterministic finite-state machines. First the protocol is modeled in this language, and then the desired properties to be verified is specified by invariants, which are boolean conditions that have to be true in every reachable state. The state showing the violation of the invariant contributes the flaw in the protocol. Lets take the example of Needham-Schroeder public-key protocol and see its model in murϕ. The data structure for the initiator is given below.

```
const
    NumInitiators: 1;
type
    InitiatorId: scalarset (NumInitiators);
    InitiatorStates: enum{I_SLEEP,I_WAIT,I_COMMIT};
    Initiator: record
        state: InitiatorStates;
        responder: AgentId;
    end;
var
    ini: array [InitiatorId] of Initiator;
```

The state of each initiator is stored in the array `ini`. `I_SLEEP, I_WAIT, and I_COMMIT` rep-

resent that the initiator has not started the protocol, the initiator has started the protocol, and the initiator is committing the protocol respectively. The behavior of an initiator is modeled by two murϕ rules. In the first rule, the initiator starts the protocol by sending the first message of NS protocol and changes its local state from `I_SLEEP` to `I_WAIT`. The second rule models the reception and checking of the second message of NS protocol, and then committing (`I_WAIT` to `I_COMMIT`) and sending the final message.

Finally the invariants represents the correctness specification of the protocol as follows.

```
invariant "responder correctly authenticated"
    forall i: InitiatorId do
        ini[i].state = I_COMMIT &
        ismember(ini[i].responder, ResponderId)
        ->
        res[ini[i].responder].initiator = i &
        ( res[ini[i].responder].state = R_WAIT |
          res[ini[i].responder].state = R_COMMIT )
end;
```

It says that for each initiator `i`, if it committed to a session with a responder, this responder (with the identifier stored in `ini[i].responder`) must have started the protocol with initiator `i`, i.e, have stored `i` in its field `initiator` and be in state `R_WAIT` or `R_COMMIT`. The intruder maintains a set of overheard messages and an array of all known nonces. Three rules represent an intruder: one for overhearing and intercepting messages, one for replaying messages from the set of overheard messages, and one for generating messages using the known nonces and injecting them into the

network.

2.9 Strand Spaces

Guttman et al. presented a graph-theoretic approach [FG98], Strand Space Method (SSM), to prove certain security properties manually, for example, authentication and secrecy. We first briefly mention some of the basic terms used in SSM.

The set of *actions Act* that principals can take during the execution of a protocol include actions such as send (denoted by +) and receive (denoted by -). An *event* is a pair $\langle action, a \rangle$, where $action \in Act$, and $a \in A$ is the argument of the action from the set of terms A. A *strand* is a sequence of events that a participant may engage in. For a legitimate participant, each strand is a sequence of message sends and receives; it represents the action of that party in a particular run of the protocol. A collection of strands for various legitimate protocol parties with penetrator-strands defines *strand space*. A strand space Σ over A is a set S together with the trace mapping $tr : S \rightarrow (\pm A)^*$. A *bundle* consists of a number of strands hooked together where one strand sends a message and another strand receives the same message. In other words, a bundle is a portion of a strand space large enough to represent a full protocol exchange. A *node* is a pair $\langle s, i \rangle$ with $s \in S$ and i an integer satisfying $1 \leq i \leq length(tr(s))$. The set of nodes is denoted by N. Each node belongs to a unique strand. $Term(n) = (tr(s))_i$, i.e, the *ith* signed term in the trace of s. There is an edge $n_1 \rightarrow n_2$ if and only if $term(n_1) = +a$ and $term(n_2) = -a$ for some $a \in A$. When $n_1 = \langle s, i \rangle$ and $n_2 = \langle s, i + 1 \rangle$ are members of N, there is an edge $n_1 \Rightarrow n_2$.

The penetrator's capabilities are also encoded in terms of strands of different forms like *M, F, T, C, S, K, E, D*, representing text message, flushing, tee, concatenation, separation, key encryption,

and decryption respectively. Strand space method uses the idea of $\preceq$-minimality in order to get the minimal member from a set of terms with respect to the causal relations $\Rightarrow$ and $\rightarrow$. Given a protocol containing some secrets, SSM tries to find out who could first *originate* that secret by tracking backward (using the relations $\Rightarrow$ and $\rightarrow$) all possible strands (of either a legitimate user or a penetrator) that could have originated the secret .

SSM uses the agreement properties by Lowe [Low97] to prove authentication. Agreement property says that a protocol guarantees a participant B (say, as the responder) agreement for certain data items x if each time a participant B completes a run of the protocol as responder using x, apparently with A, then there is a run of the protocol with the principal A as initiator using x, apparently with B. The notion of secrecy is defined as a data value x is secret in a bundle C if for every $n \in C, term(n) \neq x$. A tool has been developed based on this model called Athena [SBP01].

2.10 BRUTUS

Brutus [CJM00], a special-purpose model checker for security protocols, is a finite state-transition system that models the principals and the intruder running a protocol. It checks secrecy and the authentication properties in a protocol. Secrecy is checked by defining a set of terms that the intruder is not allowed to obtain and the authentication is checked using the correspondence properties of Woo and Lam [WL93].

Brutus uses some rules to define message derivability relation in order to model the capabilities of the adversary. A protocol is modeled as an asynchronous composition of a set of named communicating processes which model the honest agents and the penetrator. Brutus makes the model finite by placing a bound on the number of *sessions* (number of times a principal may attempt to

execute the protocol). Each session is modeled as a principal instantiating some role in the protocol, called an *instance*. Each instance is described as a separate copy or instantiation of a principal and consists of a single execution of the sequence of actions that make up that agent's role in the protocol, along with all the variable bindings and knowledge acquired during the execution. A principal can have multiple instances, but each instance is executed once. The entire model for the protocol is obtained by combining these instances with a single instance of the adversary. A *trace* is defined as each possible execution of the model, and can be obtained from a finite alternating sequence of global states and actions. Two kinds of *actions* are defined: *send* and *receive*, and some user-defined actions.

In order to specify properties of a protocol, Brutus uses first-order logic in the model. Modal logic is also combined with the predicate logic in order to capture the notion of past-time operator. In this way, one can use the past-time operator to talk about the things that happened in the history of a particular protocol run. The atomic propositions of the logic allows to refer to the bindings of variables in the model, to actions that occur during execution of the protocol, and to the knowledge of the different agents participating in the protocol. The logic used can be seen as a variant of the linear-time temporal logic with the past-time operator where one can express actions and knowledge. After the protocol is modeled, the model checker runs and checks the desired specifications in each of its states. Like any model checker, it gives the the trace of the run (a counterexample) whenever any state does not meet the specification.

2.11 Other Approaches

ASTRAL in [KD97] is another example of model checker and [Coh00, HS00] have developed special purpose theorem provers for cryptographic protocol analysis. Special purpose model checkers were designed as in [SBP01] and in [MCJ97].

Another tool-based approach , the Interrogator, was developed by Millen at MITRE. These can be found out in [KMM94]. Protocol is modeled as communicating state machines in which intruder can destroy, intercept, modify every messages. Given a final state, in which intruder knows message supposed to be secret, Interrogator constructs every possible attack scenario and verify the protocol against this state. Interrogator have successfully revealed published vulnerabilities of several protocol.

Kemmerer applied the formal method FDM with the specification language InaJo to the problem [Kem89]. FDM uses state machine with conditional transition rules for protocol steps. The security properties were represented using predicates on the states. Later, Kemmerer and colleagues applied a model-checker for real-time concurrent system specification language ASTRAL for the analysis of cryptographic protocols [KD97].

In early 1990s Abrial applied his B-method in which the protocol is formalized in the Abstract Machine Notation (AMN) of the B-method and the security properties were represented in terms of invariants. The proof of the security properties was performed using the process of step-wise refinement in which first the goal of a protocol is expressed as a single magic step. Then, this top level step is progressively refined towards the actual protocol steps, showing that the appropriate invariants are preserved. Bieber and others refined this approach, for example, in [BB94]

Longley and Rigby [LR92] describe a rule-based system used to test the vulnerability of a key management scheme to specified attacks. The system uses an exhaustive search to determine if a given attack is successful. When the system halts, then the history of rule firings can give an attack strategy.

Abadi and Gordon's spi-calculus in [AG97] is an extension of Milner's π-calculus, a development of Milner's process algebra CCS (Calculus of Communicating Systems). The pi calculus (without extension) suffices for some abstract protocols. The spi calculus considers cryptographic issues in more detail. The protocols are represented as processes in the spi calculus and their security properties are stated in terms of notions of protocol equivalence.

Abadi proposed type checking [Aba99] which is a new approach toward protocol analysis. They developed informal principles and formal rules for achieving secrecy properties in security protocols. In this approach, each piece of data and each communication channel has to be labeled as either secret or public. Secret data should not be sent on public channels, and secret channels should not be made available to everyone. This approach has a potential disadvantage of defining security violations in terms of type inconsistencies. Hence the security requirements must be considered when the specifications are being written.

Researchers have tried different approaches towards the problem of verifying cryptographic protocols over the span of 20 years. It has been shown that the protocol security problem is undecidable [EG83, HT96, CDL99]. So one can argue that the designed tools to solve this problem will not be successful all the time and will continue to need some human assistance. We have summarized the list of notations used in various approaches in Table 2.1.

Table 2.1: Symbol Table

Symbol	Meaning	Symbol	Meaning
φ	Any logical formula	Γ	Set of formulas
M	Plaintext	A, B, X, Y, S	Participant ID
T	Two-party name-stamp protocol	E_A	Encryption function
V	String of operators that an intruder can apply on any message	D_A	Decryption funcation
		$\overline{N_i(X,Y)}$	Sequence of texts transmitted between X and Y
d	Deleting an id from a string	i_X	Append operation
d_X	Deleting a known id X from a string	T_i	Time stamp i
$A \mid\equiv \#(M)$	A believes that a message M is fresh	D_X	Decryption using X's private key
$\overset{k}{\mapsto} A$	k is a public-key of A	$A \overset{k}{\longleftrightarrow} B$	k is a shared key between A and B
k	Encryption (public) key	X, Y	Concatenation of two messages X and Y in BAN
k_{AB}	Shared key b/w A and B	k^{-1}	Decryption (private) key
$PK(A,k)$	k is public key of A	$\neg\varphi$	Negation of formulae
$PK_\sigma(A,k)$	k is public signature key of A	$PK_\psi(A,k)$	k is public ciphering key of A
$\lfloor X \rfloor_k$	X is signed by k	$PK_\delta(A,k)$	k is public key-agreement key of A
$F_0(k',k)$	Function that combines k' with k^{-1} to form a shared key	$\{X\}_k$	X is encrypted by k
		N_a	Participant A's nonce
$[M]_{K_X}$	Message M with a signature that can be verified using public key K_X	$\langle X \rangle_{*A}$	A can not recognize a message
		$X\ from\ A$	A message X is coming from A

CHAPTER 3
ANALYSIS OF A SECURE SYSTEM

In today's world overwhelmed with electronic devices, trust in electronic commerce applications and reliance web security have become an important issue. Today's environment for critical applications such as e-commerce are vulnerable, in that the eavesdroppers may be listening on communication lines, or routers might have been re-programmed to look for credit card details. The communication channel is assumed to be available to any rogue entity known as penetrator, saboteur, or intruder. In addition to passive eavesdropping, a penetrator is generally assumed to be equipped with functionalities such as encryption, decryption, copying, forwarding, blocking and so on. Protecting critical applications in such environment became a cumbersome task. Below we describe the notion of a secure system and lay down the foundation upon which the security goals can be achieved.

3.1 Security Engineering

Security in general and broader sense contains several addressable issues like secrecy, authentication, etc. Security of a system can be represented in terms of three parameters. First and foremost comes the *policy*. The security policy aims to define the exact semantics and underlying notions related to it. Security is typically defined for a specific domain and within a given context. The definition of security for one domain may not be applicable for another domain. The purpose of defining a policy is to elucidate the terms and notions related to specific situation and to give it a clear semantics. After the policy for security is well understood, then comes the need to devise a *mechanism* for that policy. A mechanism is a way to achieving security goals. Security threats

needs to be clearly understood and then one needs to define security goals in order to counter those threats. Security threats encompass several issues including unauthorized access, data leaks, system integrity loss, data manipulation, denial of service, data fraud, viruses, data theft, trojan horses, data destruction, information loss, and program manipulation. In the similar vein, security goals may include entity identification, entity authentication, anonymity, data integrity, data authenticity, confidentiality, source verifiability, availability, privacy, non-repudiation, etc. Mechanism to establish a secure system includes several layers of operations that may range from defining specific arrangements and settings for each security device, to coming up with a set of protocols designed to achieve each security goal. The goal of a protocol should be clearly stated and then the protocol should be rigorously analyzed to see if it meets its goals. Last, but certainly not the least, comes the *assurance* to see if the underlying mechanism meets the policy, and to devise methods for system recovery in case of a successful attack. History has shown that even after the security system enjoys circumspect designed and attains careful implementation, the security leaks do not cease to exist. Measures to assure system recovery is essential in order to be able to backup the system in the case of a failure.

3.2 Communication Channels

In terms of message delivery, the communication channels can be broadly categorized into three groups. *Unreliable* channels have infinite possible delays and provide no guarantee on the correct order of received messages. *Resilient* channels result in finite delays but contain no guarantee on the order of messages. In *operational* channels, messages are instantly transmitted to the recipient. Therefore, the analysis of a secure system must imbue a penetrator with powerful capabilities to

model all sorts of possible attacks arising either due to the leaks in the security protocol, or flaws in the communication channel. Generally the communication channel can be assumed as a hostile environment which is open to all the participants, legitimate or otherwise, communicating with each other.

3.3 Authentication Protocols

In this exposition, we focus on the authentication property of a security protocol. In particular, our target is entity authentication. Authentication is the act of determining the identity of a principal (such as a person, computer, or server) in a computer system. An authentication protocol is a description of how some secrets are distributed to principals, and how these secrets are used to determine principal's identities [AT91]. Authentication protocols typically consists of some participants (say initiator or responder) playing in the protocol and a trusted server. In the case of public-key protocols, the trusted server, called *certificate authority*, is assumed to provide the public keys to the participants. The notion of public-key cryptography was first given by Diffie and Hellman [DH76]. Hitherto, several researchers have proposed and developed public-key schemes based on different problems such as discrete log problem, knapsack problem, and number factoring problem. On the other hand, in the case of symmetric-key protocols, the job of the trusted server, called *authentication server*, is to provide a short-lived symmetric key to the participants so that they can use it for a particular session of the protocol.

3.4 Assumptions

Design and analysis of security protocols have undergone a wide set of assumptions and constraints that the secure applications and systems put on them. We lay out these assumptions and list all the properties that we assume during the design of a secure protocol.

1. In a protocol environment each participant is associated with a unique ID. Moreover, for each ID there is a key (or a pair of public-private keys in case of asymmetric cryptosystem) associated with it.

2. In a typical protocol using symmetric cryptosystem, it is assumed that the participants possess relevant symmetric keys even before the start of the protocol. In case of asymmetric system, each participant is in possession of his private key K^{-1}. Its public cognate, K, is generally assumed to be well known.

3. A protocol message primarily consists of a set of basic terms with some operations applied on them. Basic operations are concatenation and encryption. We assume that in a protocol two messages are equal if and only if they share common atomic terms and apply identical operators in the exact order. Typically, it is represented as: $\{X\}_{K_1} = \{Y\}_{K_2}$ *iff* $X = Y \wedge K_1 = K_2$.

4. Generally, during the analysis of a protocol, encrypting a message more than once is not considered. Earlier work has shown that double or multiple encryptions do not serve a better purpose than a single encryption. Instead, sometimes it is detrimental to encrypt a message more than once [DY83].

5. We assume ideal cryptography where all participants share limited computational and cryptanalytic abilities. We assume that the protocol is *uniform* as is assumed in [DY83], that is, all participants have similar vocabularies. Thus if we exchange all X by Y and Y by X in Σ_x, we will get Σ_y. It is a fair assumption on behalf of all participants and their knowledge-base.

6. While analyzing a protocol, some assumptions may be made that greatly simplify the overall analysis mechanism. We assume that there are no type flaws, no redundant messages, and no permutation of message components of a concatenated message, in order to make the analysis feasible. However, certain kinds of redundancies are important in a protocol. For instance, redundancies are always considered that help clarify the syntax of the protocol messages.

3.5 Achieving Authentication Goals

Properties like freshness, timeliness, and uniqueness of some data items are essential in achieving goals such as authentication and secrecy. The notion of secrecy is usually defined by the idea that the secret term x should not be present in any protocol run played by all the participants of a protocol [FG98]. Moreover, after the protocol has finished, all the participants playing by the protocol should *agree* on their shared secrets. The design of an authentication protocol should respect these properties as described in detail as follows.

3.5.1 Data Freshness

It is vital to provide guarantee that the messages in a run of authentication protocol are fresh. It ensures that the communicating messages are not the replay of the old messages. For this purpose,

generally a fresh data item is generated at each run of a protocol. Any message that is *bound* to this freshly generated data is also assumed to be fresh. It is important to note that the fresh data item should be tightly bound with the rest of the message whose freshness is required for the proper operation of the protocol. This binding can be achieved, for example, by first concatenating the fresh data item with the rest of the message, and then encrypting the entire message with some secret key. In order to generate fresh data item, different schemes have been analyzed and incorporated. The use of a sequence number from a random selected seed, a time stamp, and a random number generated once (called a *nonce*) are some of the solutions proposed in order to conceptualize the notion of freshness. There are some pros and cons associated with each of these approaches. The use of a sequence number, for instance, is advantageous where the processing power of the computing device is available at high costs. In that case, generating a pure random number every time a session of a protocol is initiated, is infeasible. However, the advantage of using sequence number comes with the problem of having a non-volatile memory so that the server does not reset its count every time it is shut down and restarted. Moreover, guessing a sequence number is easier than guessing a nonce, hence a counter based scheme is attacker-prone. Using a nonce is more reliable but computationally expensive method. Time stamp, on the other hand, seems an attractive choice, but it suffers from clock synchronization problems. It is a good design strategy to come up with a hybrid of these approaches and utilize the advantages of each of them according to the constraints of the system at hand.

3.5.2 Agreement Properties

Gavin Lowe [Low97] proposed agreement properties for authentication protocols. A protocol guarantees an agreement property for a participant B (e.g. acting as a responder) for a certain set of parameters x, if each time the principal B completes a run of the protocol as a responder using x, supposedly with A, then there is a unique run of the protocol with the principal A initiating a session with the same parameters x, supposedly with B. A weaker non-injective agreement does not ensure uniqueness, but requires only that each time a principal B completes a run of the protocol as responder using x, supposedly with A, then there is a run of the protocol with the principal A as the initiator using x, supposedly with B [Low97]. We take an example protocol from the literature, namely Needham-Shroeder (NS) public-key protocol. NS protocol is a classical protocol that has been serving as a test bench for several methods in this field.

In its simplest form, NS can be represented as:

$$1.\ A \rightarrow B \quad : \quad \{AN_a\}_{K_b}$$

$$2.\ B \rightarrow A \quad : \quad \{N_a N_b\}_{K_a}$$

$$3.\ A \rightarrow B \quad : \quad \{N_b\}_{K_b}$$

A first challenges B by sending its nonce N_a with its ID encrypted with $B's$ public key. B dreams up a new nonce N_b and replies the challenge by sending an encrypted message containing both nonces. A finally completes the protocol by sending back $B's$ nonce in the encrypted form.

The set of parameters for this protocol contains four variables: X (presumably the *initiator*), Y (presumably a *responder*), N_x (initiator's secret), and N_y (responder's secret). Notice that by

the time the initiator receives a reply in its second message, its parameter list X, Y, N_x, and N_y binds to A, B, N_a, and N_b respectively. In order to ensure authentication from initiator's side of the protocol, he needs to guarantee (by convincing himself) that the corresponding apparent responder also shares the same vocabulary, that is, A, B, N_a, and N_b. In the similar vein, the responder should also convince himself about the agreement of his data items with that of the initiator.

Agreement on data items seems necessary in order to ensure authentication between participants. A good number of attacks on security protocols resulted in disparity of data items among legitimate participants playing in a protocol. Disagreement either in the participant's ids or in the values of their secrets is a major cause of subtle attacks found in the literature. The following subsection further illustrates the point with an example.

3.5.3 Agreement Between Secret Values

Agreement is an important concept, in that the goal of a protocol may not be achieved even after the secrecy property is satisfied. Agreement in the shared secrets means that after a protocol has finished, all the participants playing by the protocol should *agree* on their shared secrets. To make our point, we consider another protocol, by Otway and Rees, that resulted into mismatched keys after an attack in [FHG99]. Otway-Rees protocol [OR87] consists of three participants in which two participants want to obtain a shared secret given by a server. Attacked Otway-Rees protocol is

given below:

$$
\begin{aligned}
1.\ A \rightarrow B &: MAB\{N_a MAB\}_{K_{AS}} \\
2.\ B \rightarrow P &: MAB\{N_a MAB\}_{K_{AS}}\{N_b MAB\}_{K_{BS}} \\
3.\ P \rightarrow S &: MAB\{N_a MAB\}_{K_{AS}}\{N_b MAB\}_{K_{BS}} \\
4.\ S \rightarrow B &: M\{N_a K_{AB}\}_{K_{AS}}\{N_b K_{AB}\}_{K_{BS}} \\
3'.\ P \rightarrow S &: MAB\{N_a MAB\}_{K_{AS}}\{N_b MAB\}_{K_{BS}} \\
4'.\ S \rightarrow P &: M\{N_a K'_{AB}\}_{K_{AS}}\{N_b K'_{AB}\}_{K_{BS}} \\
5.\ P \rightarrow A &: M\{N_a K'_{AB}\}_{K_{AS}}
\end{aligned}
$$

Notice how the penetrator P attacks the protocol such that A and B get unmatched secret session keys after communicating with the server S. The penetrator simply forwards the message obtained from B to the server twice in order to get two mismatched keys K_{AB} and K'_{AB}. Then, the penetrator uses simple *separation* operation on the concatenated message from the server to complete the protocol run for participants A and B.

3.5.4 Secure Functions

Authentication methods can be implemented using different schemes in a security protocol. Encryption is the main method used to achieve authentication goals. Symmetric or asymmetric encryption can be used in a protocol so as to implement authentication. Message Authentication Code (MAC) functions of the text and the keys, and Hash (normally non-invertible) functions and involving secret keys are other ways of implementing security goals like authentication. Several

protocols have been designed such that they utilize combination of these techniques to meet their objectives. For instance, consider the following simplified version of the TLS protocol [DA99] that utilizes both the public key cryptography (to sign a message) and the message hashing (to obtain a key) to achieve its goals.

$$
\begin{aligned}
&1.\ C \rightarrow S \quad : \quad C \\
&2.\ S \rightarrow C \quad : \quad S[g^x]_{K_S} \\
&3.\ C \rightarrow S \quad : \quad [g^y]_{K_C}\{T_1 CS\}_{K'} \\
&4.\ S \rightarrow C \quad : \quad \{T_2 CS\}_{K'}
\end{aligned}
$$

In the above protocol, T_1 and T_2 are fixed tags to distinguish the third message from the fourth. $[M]_{K_X}$ is the message M with a signature that can be verified using the public key K_X. Here K' is a symmetric key created by hashing the value g^{xy}.

3.5.5 Underlying Cryptosystem

Design of the authentication protocols depends heavily on the underlying cryptosystem used to achieve security goals. For instance, the protocols using public key cryptography suffers from the constraint that an encrypted message does not provide guarantee for the identity of the sender of the message, or at least the originator of the message. The reason being obviously the fact that anyone can compose a message masquerading someone else by using public key of the intended recipient. That is why, protocols using public key cryptography typically rely on a challenge

response criterion in order to achieve authentication. In such protocols, the initiator usually dreams up a random number as a challenge and sends out this challenge after encrypting it with the public key of the recipient. The responder, upon receiving the challenge, tries to respond to the initiator often by including its own challenge in case if two-way authentication is desired. The symmetric key cryptography, on the other hand, has the advantage that an encrypted message contains some guarantees about the originator of the message. Therefore, there is no need to send an encrypted challenge, or any challenge at all. Rather, any random number or a time stamp can be sent to the intended recipient, even in an un-encrypted form. This ensures that the participants playing in the current session of the protocol do not become victim of replay attacks. Let us illustrate the point by giving an example protocol based on symmetric keys. The protocol below is designed simply for illustrative purpose and we do not intend to make any claim about its security.

$$
\begin{aligned}
&1.\ A \rightarrow B \ : \ T_1 AB \\
&2.\ B \rightarrow A \ : \ \{T_2 AB\}_{K_{AB}} \\
&3.\ A \rightarrow B \ : \ \{T_3 AB\}_{K_{AB}}
\end{aligned}
$$

Here T_1, T_2 and T_3 are time stamps, and K_{AB} is a symmetric key used by participants A and B. Notice that how utilizing different cryptographic schemes affects the design of a simple protocol.

3.5.6 The Notion of Security

As mentioned earlier, declaring a protocol secure is heavily contingent upon how the notion of security is defined. Different notions of security entails different design issues related to it. A

protocol declared secure according to one situation may be highly vulnerable in different situation. For instance, consider below the two party name stamp protocol discussed by Dolev and Yao in [DY83].

$$1.\ A \rightarrow B \quad : \quad A\{MA\}_{K_B}B$$

$$2.\ B \rightarrow A \quad : \quad B\{MB\}_{K_A}A$$

This protocol is claimed to be secure. However, its security is contingent upon the intention of the initiator of this protocol. It is clear that the protocol does not achieve two-way guarantees about any secrets in it. The participant B has no way of assuring that the secret M is indeed sent by the participant A because anyone can construct the first message using the public key of the recipient.

3.6 The Penetrator

An important parameter worth mentioning in the design of a security protocol is how to model the penetrator. The penetrator may be modeled explicitly or a formal system may capture the capabilities of a penetrator implicitly. The way a penetrator's capabilities are assessed and analyzed greatly affects the design of a secure protocol. Plethora of security leaks in some well known protocols resulted from misunderstanding the penetrator and his abilities. For instance, in the logic of Burrows, Abadi, and Needham, called BAN logic [BAN90], the authors made the assumption of trusted participants in the protocol. This was the reason that BAN overlooked the hidden flaw while analyzing the Needham-Schroeder public-key protocol for its authentication property. Lowe's attack on NS protocol [Low96] greatly motivated the researchers towards giving a clear semantics

on the capabilities of the penetrator. Understanding the capabilities of a penetrator greatly helps design a secure protocol. A stringent approach may assume all participants of the protocol to be the penetrators until and unless they complete their part of the protocol run exactly as specified by the protocol. To flip the coin, an equivalent strategy is to assume all penetrators to be the legitimate participants of the protocol. Notice that the above mentioned definition of the penetrator is quite strict, but this conservativeness seems essential for the design of a safe protocol.

3.6.1 Capabilities of a Penetrator

From the notion of a penetrator given above, we can describe the set of operation that a penetrator can perform on messages he receives from the network. The penetrator may also apply any combination of these operations in order to find a successful attack in a protocol.

Generation A penetrator is generally assumed to generate messages from the set of messages he possesses. He may generate new messages using some operations performed on this set.

Forwarding Forwarding is defined as sending a message after receiving it from some participant. A penetrator is equipped with the ability to forward any message in the network to any participant he likes.

Copying Message copying can be done by simply duplicating the data item even though the ingredients of the data items are unknown to a participant. The notion of a penetrator possess this property that he can copy any message to create it duplicates as many times as he wants in order to attack a protocol.

Deleting A penetrator can obtain any message from the network and then can flush it out from the

network. This is similar to deleting of messages from a public network channel.

Concatenation A penetrator is assumed to be able to perform concatenation on two messages he possesses.

Encryption A penetrator is assumed to able to encrypt any data item it possess with any key he possesses. This can result in infinite number of messages that a penetrator can generate by simply encrypting a message over and over using a key. However, in a realistic scenario, only a bounded number of encryption suffices the analysis.

Decryption Similar to encryption, a penetrator is capable of decrypting a message using a key he possesses. The key may have been obtained by any successful attack on a protocol, or he may have performed cryptanalysis on any old session key to obtain the key.

CHAPTER 4
OUR WORK

Since our work focuses on developing a proof-based method from distributed temporal protocol logic, we first briefly describe the the Distributed Temporal Protocol Logic. From this chapter onward, we will be using the set of symbols given in Table 4.1.

4.1 The Distributed Temporal Protocol Logic

We briefly introduce the distributed temporal protocol logic of [CVB05a], a version of the distributed temporal logic DTL, to reason about protocols and their properties. In DTPL, a distributed system is viewed as a collection of communicating sequential objects (principals or agents) that interact by exchanging messages through an insecure channel *Ch*. A security protocol comprises of a sequence of messages sent and received by two or more principals in a distributed system. DTPL represents a protocol by capturing the temporal aspect of the sequence of actions of each principal executing the protocol. We usually represent by **A** the set of messages that principals communicate where we refer to the elements of **A** as *terms* or *messages*. Moreover, we assume *algebra freeness* in which **A** is *freely* generated from two disjoint sets, T (for texts, e.g, nonces or principal ids) and K (for keys). It implies that two syntactically different terms can not represent the same message. In particular if M, M', N, N' represent terms and k and k' are keys, then according to the algebra freeness assumption:

1) $\{M\}_k = \{M'\}_{k'} \Rightarrow M = M' \wedge k = k'$,

2) $MN = M'N' \Rightarrow M = M' \wedge N = N'$,

3) $MN \neq \{M'\}_k$.

Table 4.1: Symbol Table for DTPL

Symbol	Meaning	Symbol	Meaning
$\top$	True	$\bot$	False
k	Encryption (public) key	k^{-1}	Decryption (private) key
$\{M\}_k$	Encrypting message M with key k	MN	Concatenating messages M and N
Ch	Channel	$\mathbf{A}$	Set of messages
T	Text message	K	Set of keys
M, M', N, N'	Terms	k, k'	Keys
Act_A	Actions of principal A	$Prop_A$	State propositions of Participant A
$A \overset{\psi}{\mapsto} k$	k is public-encryption key of principal A	$@_i[\phi]$	ϕ holds at the current local state of principal i
$A \overset{\sigma}{\mapsto} k$	k is public-signature verification key	$A \overset{\delta}{\mapsto} k$	k is public key-agreement key of principal A
	of principal A	$A \overset{k}{\leftrightarrow} B$	k is shared key b/w A and B
$\Rightarrow$	Conditional	$M_X, X \sqsubset M$	Message M contains term X
$j : \phi$	Principal i has communicated with	λ	Distributed life-cycle
	principal j for whom ϕ held	Ev_i	Set of events of principal i
L	Global language	L_i	Local language of principal i
$\wp(Prop_i)$	Local state propositions of principal i	$\Vdash$	Global satisfaction relation
act	action	$\Vdash$	Local satisfaction relation
μ	Interpretation structure	ψ, ϕ	Logical formula
$\xi\|_i$	$\xi \cap Ev_i$	Γ	Set of formulas
ξ_i	Local configuration of principal i	α_i	Associates a local action to each local event
$\rightarrow_i$	Local successor relation b/w events in Ev_i	Ξ_i	Set of local configurations of principal i
π_i	Associates a set of local state	ξ	global configuration
	propositions to each local configuration	$e\downarrow$	$\{e' \in Ev \| e' \rightarrow^* e\}$

Local alphabet of each principal A comprises of actions Act_A and state propositions $Prop_A$. Act_A includes operations such as sending a message M to principal B, $send(M, B)$, receiving a message M, $rec(M)$, spying a message M, $spy(M)$, generating a nonce N, $nonce(N)$, and generating a key k, $key(k)$ whereas $Prop_A$ includes only knowing a message M, $knows(M)$. Channel's actions Act_{Ch} include receiving a message M from principal A, $in(M, A)$, sending a message M to principal A, $out(M, A)$, and leaking any message, *leak* and there is no state proposition associated with the channel.

DTPL captures all the local actions and local state propositions of a principal i using its *local language* L_i. Local languages of all principals and interactions among them is captured by means of *global language L.* The global language and the local language of the logic are defined by the grammar[1]:

$$L ::= @_i[L_i] \mid \bot \mid L \Rightarrow L$$

$$L_i ::= Act_i \mid Prop_i \mid \bot \mid L_i \Rightarrow L_i \mid L_i \,\mathsf{U}\, L_i \mid L_i \,\mathsf{S}\, L_i \mid j : L_j$$

where,

- $i, j \in Princ$, a set of principal's ids.
- U and S are temporal operators *until* and *since*,
- $@_i[\phi]$ means that ϕ holds at the current local state of principal i, and
- $j : \phi$ appearing inside a formula in L_i is called a *communication formula*. It means that principal i has just communicated with principal j for whom ϕ held.

[1]DTPL uses $\Rightarrow$ to represent the conditional instead of $\rightarrow$ used in the previous section in other logics. We follow the DTPL's notation in order to avoid confusion since DTPL uses $\rightarrow$ for other purposes.

Due to the concurrent nature of the distributed system, event structures are used instead of Kripke structures as the interpretation structures in DTPL. In DTPL, λ, a prime event structure without conflict built form Ev_i, is called a distributed life-cycle. The interpretation structure $\mu = \langle \lambda, \alpha, \pi \rangle$ of L are suitably labeled distributed life-cycles, built upon a simplified form of Winskel's *event structures* [Win87]. If Ev_i represents a discrete, linearly ordered, set of events for each principal $i \in Id$:

- λ: is a distributed life-cycle.
- $\alpha_i : Ev_i \rightarrow Act_i$ associates a local action to each local event.
- $\pi_i : \Xi_i \rightarrow \wp(Prop_i)$ associates a set of local state propositions to each *local configuration* ξ_i in the set of local configurations Ξ_i.

Here, *local configuration* of a principal i means a collection of all the local events that have occurred up to a given point. In other words, local configuration of a principal i is a finite set $\xi_i \subseteq Ev_i$ closed under local causality. That is, if $\rightarrow_i$ represents the local successor relation between the events in Ev_i, $e \rightarrow_i^* e'$, and $e' \in \xi_i$ then also $e \in \xi_i$. Every non-empty local configuration ξ_i is reached by the occurrence of an event $last(\xi_i)$ from the local configuration $\xi_i \backslash last(\xi_i)$. A *global configuration* is a finite set $\xi \subseteq Ev$ closed under global causality, that is, if $e \rightarrow^* e'$, and $e' \in \xi$ then also $e \in \xi$. Every global configuration ξ includes the local configuration $\xi|_i = \xi \cap Ev_i$ of each principal i. Moreover, given $e \in Ev$, $e \downarrow = \{e' \in Ev | e' \rightarrow^* e\}$ is always a global configuration. The distributed life-cycle of three principals A, B, and C is shown in Fig. 4.1 where dotted vertical line represents communication point. The progress of principal A in shown in Fig. 4.2.

Using the interpretation structure defined above, the global satisfaction relation at a global

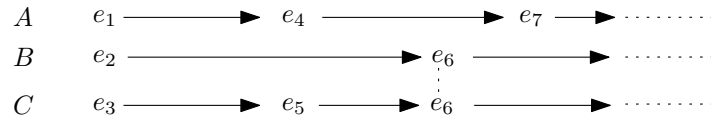

Figure 4.1: A Distributed Life-cycle for Principals A, B, and C

$$\pi_A(\emptyset) \xrightarrow{\alpha_A(e_1)} \pi_A(\{e_1\}) \xrightarrow{\alpha_A(e_4)} \pi_A(\{e_1, e_4\}) \xrightarrow{\alpha_A(e_7)} \pi_A(\{e_1, e_4, e_7\}) \cdots\cdots$$

Figure 4.2: The Progress of Principal A

configuration ξ of μ can be defined as:

- $\mu, \xi \Vdash @_i[\phi]$ if $\mu, \xi|_i \Vdash_i \phi$,
- $\mu, \xi \nVdash \bot$,
- $\mu, \xi \Vdash \gamma \Rightarrow \delta$ if $\mu, \xi \nVdash \gamma$ or $\mu, \xi \Vdash \delta$,

where the local satisfaction relations $\Vdash_i$ at local configurations are defined as:

- $\mu, \xi_i \Vdash_i act$ if $\xi_i \neq \emptyset$ and $\alpha_i(last(\xi_i)) = act$,
- $\mu, \xi_i \Vdash_i p$ if $p \in \pi_i(\xi_i)$,
- $\mu, \xi_i \nVdash_i \bot$,
- $\mu, \xi_i \Vdash_i \varphi \Rightarrow \psi$ if $\mu, \xi_i \nVdash_i \varphi$ or $\mu, \xi_i \Vdash_i \psi$,
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{U}\, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i \subsetneq \xi_i''$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \phi$ for every $\xi_i' \in \Xi_i$ with $\xi_i \subsetneq \xi_i' \subsetneq \xi_i''$,
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{S}\, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \phi$ for every $\xi_i' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i' \subsetneq \xi_i$, and
- $\mu, \xi_i \Vdash_i j : \phi$ if $\xi_i \neq \emptyset$, $last(\xi_i) \in Ev_j$ and $\mu, (last(\xi_i)\downarrow)|_j \Vdash_j \varphi$.

Table 4.2: Temporal Operators

Operator	Meaning	Operator	Meaning
$\mathsf{X}\,\varphi \equiv \bot\,\mathsf{U}\,\varphi$	next	$\dagger \equiv \neg\,\mathsf{X}\,\top$	in the end
$\mathsf{Y}\,\varphi \equiv \bot\,\mathsf{S}\,\varphi$	previous	$* \equiv \neg\,\mathsf{Y}\,\top$	in the beginning
$\mathsf{F}\,\varphi \equiv \top\,\mathsf{U}\,\varphi$	sometime in the future	$\mathsf{F}_o\,\varphi \equiv \varphi \vee \mathsf{F}\,\varphi$	now or sometime in the future
$\mathsf{P}\,\varphi \equiv \top\,\mathsf{S}\,\varphi$	sometime in the past	$\mathsf{P}_o\,\varphi \equiv \varphi \vee \mathsf{P}\,\varphi$	now or sometime in the past
$\mathsf{G}\,\varphi \equiv \neg\,\mathsf{F}\,\neg\varphi$	always in the future	$\mathsf{G}_o\,\varphi \equiv \varphi \wedge \mathsf{G}\,\varphi$	now and always in the future
$\mathsf{H}\,\varphi \equiv \neg\,\mathsf{P}\,\neg\varphi$	always in the past	$\mathsf{H}_o\,\varphi \equiv \varphi \wedge \mathsf{H}\,\varphi$	now and always in the past

The interpretation structure μ is called a *model* of $\Gamma \subseteq L$ if $\mu, \xi \Vdash \gamma$ for every global configuration ξ of μ and every $\gamma \in \Gamma$. Other operators are defined in Table 4.2.

In DTPL, the principals are equipped with two functions *synth* and *analz* to compose (through concatenation[2] and encryption) and decompose (through separation and decryption) messages respectively. In particular, if S is a set of messages, then:

The function $synth(S)$ is the smallest set containing S such that:

If $M \in synth(S)$ and $k \in synth(S)$, then $\{M\}_k \in synth(S)$.

If $M_1 \in synth(S)$ and $M_2 \in synth(S)$, then $M_1 M_2 \in synth(S)$.

Similarly, $analz(S)$ is the smallest set containing S such that:

If $\{M\}_k \in analz(S)$ and $k^{-1} \in analz(S)$, then $M \in analz(S)$.

If $M_1 M_2 \in analz(S)$ then $M_1 \in analz(S)$ and $M_2 \in analz(S)$.

In [CVB05b], a number of axiom schemas have been proposed to represent the specifications of the communication network. The following axiom schemas represent the notion of perfect cryptography. That is, knowledge of each principal only depends on his initial knowledge and on the actions that have occurred.

[2]The DTPL uses ; to represent message concatenation. For simplicity, we write AB to represent the concatenation of the two terms A and B.

(K1) $@_A[knows(M_1M_2) \Leftrightarrow (knows(M_1) \wedge knows(M_2))]$,

(K2) $@_A[knows(M) \wedge knows(k) \Rightarrow knows(\{M\}_k)]$,

(K3) $@_A[knows(\{M\}_k) \wedge knows(k^{-1}) \Rightarrow knows(M)]$,

(K4) $@_A[knows(M) \Rightarrow \mathsf{G}_o\ knows(M)]$,

(K5) $@_A[rec(M) \Rightarrow knows(M)]$,

(K6) $@_A[spy(M) \Rightarrow knows(M)]$,

(K7) $@_A[nonce(N) \Rightarrow knows(N)]$, and

(K8) $@_A[key(k) \Rightarrow knows(k)]$.

The first three axiom schemas state that a principal can separate a concatenated message (K1) and encrypt or decrypt a message with the known keys (K2, K3). K4 simply says that a principal does not forget its known messages. K5 through K8 say that a principal knows its received, spied, or generated (nonces, keys) messages.

The behavior of the channel and the way principals communicate with each other are captured in terms of the following axiom schemas.

(C1) $@_{Ch}[in(M, A) \Rightarrow \bigvee_{B \in Princ} B : send(M, A)]$,

(C2) $@_{Ch}[out(M, A) \Rightarrow \mathsf{P}\ in(M, A)]$,

(C3) $@_{Ch}[out(M, A) \Rightarrow A : rec(M)]$,

(P1) $@_A[send(M, B) \Rightarrow \mathsf{Y}(knows(M) \wedge knows(B))]$,

(P2) $@_A[send(M, B) \Rightarrow Ch : in(M, B)]$,

(P3) $@_A[rec(M) \Rightarrow Ch : out(M, A)]$[3],

(P4) $@_A[spy(M) \Rightarrow Ch : (leak \wedge \mathsf{P} \bigvee_{B \in Name} in(M, B))]$,

[3]In [CVB05a], the channel may output M to any possible aliases used by a principal.

(P5) $@_A[\bigwedge_{B \in Princ \setminus \{A\}} \neg B : \top]$,

(P6) $@_A[nonce(N) \Rightarrow \neg Ch : \top]$, and

(P7) $@_A[key(k) \Rightarrow \neg Ch : \top]$.

C1 through C3 state that a message arrives at a channel only if it is sent by some principal, that the channel delivers a message only if it was previously arrived, and that a principal receives a message if channel delivers it. The axiom schemas P1 through P7 state that a principal must know the message and the recipient of a sending message (P1), that a sending or a receiving message must go through the channel (P2, P3), that a spied messages must have been received and leaked by a channel (P4), that principals do not interact directly (P5), and that nonce and key are not channel events (P6, P7). Finally, the following axiom schemas capture the freshness and uniqueness of the nonces respectively:

(N1) $@_A[nonce(N) \Rightarrow \mathsf{Y}\neg knows(M_N)]$, and

(N2) $@_A[nonce(N)] \Rightarrow \bigwedge_{B \in Princ \setminus \{A\}} @B[\neg knows(M_N)]$,

where M_N ranges over all the messages containing the nonce N. N1 captures freshness of N as a principal does not know any message containing N before it generates N using *nonce* action. N2 captures uniqueness of N as at the time N is generated, no principal knows any message containing N except the one who generated it.

4.2 Developing a Proof-based Verification Framework

In this section, we first describe some of the similarities and the differences in the language of messages and formulae in SVO and DTPL. We also describe how the messages and formulae in SVO can be represented in DTPL. Finally, we demonstrate how to benefit from the SVO axioms in

order to develop a framework in DTPL that can be used to verify authentication protocols. In doing so, we have tried to refrain from introducing new symbols and constructs into DTPL and used the existing framework. However, we do introduce some new notions, and the associated symbols and constructs, either if it was lacking in DTPL or if it considerably simplifies the protocol analysis. Throughout the discussion, we also give comparative comments at appropriate points on how our work differs from the SVO [MFG07b].

4.2.1 The language of Messages and formulae

Both approaches use similar primitives to define their language of messages of security protocols. In particular, SVO assumes the existence of a set T of primitive terms for principal's names, their keys and constants. Whereas, DTPL uses finite sets *Princ* for principal names, *Nonce* for random numbers, and *Key* for keys. Moreover, in the SVO, messages and formulae are built by mutual induction, whereas DTPL treats messages and atomic propositions separately. In both approaches, concatenation and encryption on existing messages introduce new messages.

SVO uses primitive proposition constants, some standard operators, and higher-level constructs (like *believe*, *sees*, *has*) to define formulae. In the DTPL, the principals as well as the channel are associated with local actions (Act_A and Act_{Ch}) and local state propositions ($Prop_A$ and $Prop_{Ch}$) that contribute towards formula. DTPL also uses temporal operators given in Table 4.2 besides standard operators. Below we describe how the logical constructs of SVO can be represented in the DTPL.

- *A received X*: In SVO, it includes all the received messages plus all the messages that can be analyzed using X by a principal A. We do not call the analyzed sub-messages of a received

message X to have been received in DTPL. For example, if $\{M\}_k$ is received by a principal who possesses k^{-1}, then we do not call M to be a received message. Instead, we use the notion of *knows* along with *rec* to represent knowing a received message X and extend *knows* to all the sub-messages of X, by applying the function *analz*, that can be analyzed using X. It seems more intuitive as A did not explicitly receive any of the sub-messages of X, but knows them by applying the function *analz*.

- A *sees* X: It includes all the messages that are received, newly generated, or initially available to A plus all the messages that A can produce from these messages. In effect, the DTPL's actions *rec*, *spy*, *nonce*, *key* and proposition *knows* can be collectively used to define the SVO's *sees* as axiomatized in the axioms K1 through K8 excluding K4. Obviously, *sees* does not imply future assertions and hence K4 is not catered in *sees*. If X is restricted to only keys k, then A *has* k in SVO can be defined in the similar way.

- A *said* X: It simply represents that A has sent X at some time in the past. Since DTPL has a rich set of operators to capture various temporal aspects, it can easily represent such constructs. In particular, the DTPL's past time operator P_0 on its action *send* provides similar effect. Moreover, instead of SVO's two constructs *said* and *says*, we can choose relevant temporal operator on *send* in order to provide a precise interpretation of when a message was exactly said in DTPL.

Now we introduce the following logical constructs into the language of the DTPL.

- $fresh(X)$: Freshness condition in SVO represents what has not been said prior to the current run of a protocol. Notice that the definition of freshness in SVO provides sufficient condition

which not only caters the freshness of nonces, but it can also be applied, in general, on any term. For example, principal's ids are not generally considered fresh because they were transmitted to their owners or to other principals at some time in the past (and the SVO's definition of freshness holds). However, we argue that this definition of freshness can not be generalized for any message. Rather, it should be restricted to only atomic terms (like nonces). For example, according to this definition, provided that an id A and a key k are not fresh, the message $\{A\}_k$ will still be called fresh just because if nobody ever sent this message in the past. Since freshness is used to capture, generally with the help of a nonce or a timestamp, that a message is communicated recently in a protocol, calling $\{A\}_k$ fresh violates the purpose. Of course, $\{A\}_k$ does not guarantee that a message is fresh. Using the similar idea, DTPL restricts its definition of freshness to the nonces and adopts a more conservative approach by assuming that none of the principals know a fresh message before the time when it was generated.

- $A \overset{k}{\longleftrightarrow} B$, $\mathbf{PK}(A, k)$: These key association constructs of SVO provide useful information about a principal's association of certain keys. $A \overset{k}{\longleftrightarrow} B$ indicates that k is a key exclusively shared between A and B whereas $\mathbf{PK}(A, k)$ indicates that k is the public-key of A. We extend the DTPL message structure by adopting key association constructs of SVO. However, we change the SVO notation as follows[4]. $\mathbf{PK}_{\psi}(A, k)$ is changed to $A \overset{\psi}{\mapsto} k$ (for encryption keys), $\mathbf{PK}_{\sigma}(A, k)$ is changed to $A \overset{\sigma}{\mapsto} k$ (for signature keys), and $\mathbf{PK}_{\delta}(A, k)$ is changed to $A \overset{\delta}{\mapsto} k$ (for key-agreement keys). Moreover, due to the specific nature of these constructs, special

[4]It is simply a matter of taste. Instead of using an entirely different notation for public keys (i.e., **PK**), we find our modified notation for public-keys $\mapsto$ to be closer to the corresponding notation for symmetric-key $\longleftrightarrow$. BAN also uses $\mapsto$ for public-keys.

meaning can be assigned to them. Specifically, $A \overset{k}{\longleftrightarrow} B$ or $A \overset{x}{\mapsto} k$ also represent a proposition which can be assigned truth values. That is why, they represent both, messages and formulae, the proper use of which can be easily understood from the context.

- *A controls X*: It represents *A*'s jurisdiction over *X*. Since it is normally used to mention authority of a key-server to generate trusted keys, it represents an essential part of authentication protocols. We introduce a proposition $controls(X)$ to represent this feature in DTPL.

4.2.2 The Axioms of the Framework

First, we modify the way DTPL models the communication channel as axiomatized in C1 through C3. DTPL models a lossy channel which can lose the data without any intruder intervention. Since our focus is on guaranteeing the security of a protocol with respect to the intruder, we can restrict ourself to a reliable channel. That is, in the absence of an intruder, messages will reach to their intended destination. We add the following axioms to capture the reliability of the communication channel.

(C4) $@_{Ch}[in(M, A) \Rightarrow \mathsf{F}(leak \vee out(M, A))]$

(C5) $@_{Ch}[leak] \Rightarrow @_{A}[spy(M)]$

By P2, C4, C5, C3, now it is easy to see the following:

(P8) $@_{A}[send(M, B)] \Rightarrow \bigvee_{P \in Princ} @_{P}[\mathsf{F}\ (spy(M) \vee rec(M))]$

The above axioms state that a channel always transfer the data to a principal, legitimate or otherwise.

DTPL utilizes temporal dimension of knowledge of a principal and does not use the notion of 'belief', thereby avoiding confusion caused by different interpretations of the notion of belief in

previous logics [AT91]. In fact, one of the main contributions of GNY [GNY90] and its successors (like SVO) was to separate the notion of belief from other notions, like knowing that a shared-key exists between two principals A and B in $A \overset{k}{\longleftrightarrow} B$. Below we describe how we take advantage from SVO in order to design axioms for our verification framework of DTPL. We have changed the order in which SVO axioms appear in Section 2.4 in order to obtain continuity of presentation in this section. We use modus ponens as used by SVO as the only inference rule.

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

Since DTPL does not utilize the epistemic properties of its knowledge [CVB05a], the necessitation rule and the belief axioms 1 through 4 of SVO have not been used in our exposition. Moreover, due to their importance, the *source association* axioms of SVO are treated separately in the next subsection.

4.2.2.1 Possession:

Since the notion of *knows* in DTPL represents possession in the SVO, all the possession axioms can be seen in terms of knowledge axioms already described in the previous section. In particular, Axiom 12 of SVO can be represented by Axiom K5 of DTPL and Axiom 13 of SVO can be easily derived using Axiom K1 of DTPL as follows.

(K1a) $@_A[knows(X_1 \ldots X_n) \Rightarrow knows(X_i)]$ for $i = 1, \ldots, n$.

Similarly, Axiom 14 of SVO is the generalized form of K2 and K3 of DTPL given as follows.

(K2a) $@_A[knows(X_1) \wedge \ldots \wedge knows(X_n) \Rightarrow knows(F(X_1, \ldots, X_n))]$

where F represents any function computable by A, e.g. encryption, signature etc.

4.2.2.2 Receiving:

All the receiving axioms of the SVO can be translated into DTPL using knowledge Axioms K5, K1 and K3. K5 along with K1a represent Axiom 9 as follows.

$@_A[rec(X_1 \ldots X_n) \Rightarrow knows(X_1 \ldots X_n)]$

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

Note that DTPL's equivalent of SVO's Axiom 9 is given in terms of knowing the concatenates of a received concatenated message instead of receiving it. Similarly, receiving an encrypted term such that a principal holds the corresponding decryption-key means that the principal knows the contents of the encrypted message. Since we have adopted a richer representation of key association from SVO, we can break down K3 of DTPL for symmetric-key encryption, public-key encryption, and signatures as follows.

(K3a) $@_A[knows(\{M\}_{k_{AS}}) \wedge knows(A \overset{k_{AS}}{\longleftrightarrow} S) \Rightarrow knows(M)]$,

(K3b) $@_A[knows(\{M\}_k) \wedge knows(A \overset{\psi}{\mapsto} k) \Rightarrow knows(M)]$,

(K3c) $@_A[knows(\{M\}_{k^{-1}}) \wedge knows(B \overset{\sigma}{\mapsto} k) \Rightarrow knows(M)]$,

Notice that SVO's notion of key association not only captures different notions of encryption, but also provides a clear understanding of knowing the relevant key associated to a principal. For example, knowing $A \overset{\psi}{\mapsto} k$ not only implies the possession of k but also binds it with A. The Axiom K5 along with K3(a,b,c) give the DTPL representation of Axioms 10 and 11 of the SVO. Although SVO represents the sub-messages of a received message as being received, we believe that our adherence to the notion of received messages to only those which are actually received during communication is more intuitive.

4.2.2.3 Comprehension and Saying:

The way DTPL models knowledge (K1 through K8), it can not be claimed that a principal A knows X if he knows $F(X)$. For example, a principal knows a message if he receives it (K5), but he may not be able to invert $F(X)$ to obtain X. As also mentioned in SVO, its Axiom 15 does not imply that F is invertible by A. Therefore, we do not use the comprehension axiom of SVO in DTPL. Moreover, since we do not hold a principal responsible for saying all the concatenates of a concatenated message, we intentionally remove the saying Axiom 16 in our work. However, the formula P1 partially represents Axiom 16 in which a principal must know what it says. We do not have two different temporal constructs for sending a message like SVO's *says* and *said*. Since we can use past time temporal operators to capture several past time activities, we do away with Axiom 17 in our work.

4.2.2.4 Freshness:

Since a nonce action $nonce(N)$ in DTPL represents the notion of freshness, we use this action to introduce a new proposition $fresh(N)$ similar to SVO. In particular, we introduce:

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

The rules N1 and N2 clarify the freshness concept in the DTPL. As used in SVO, a term remains fresh in the current run of a protocol. That is,

(F1a) $@_A[fresh(N) \Rightarrow \mathsf{X}\, fresh(N)]$

Now, we can use Axiom 18 of SVO as follows.

(F2) $@_A[fresh(X_i) \Rightarrow fresh(X_1 \ldots X_n)]$, for $i = 1, \ldots, n$.

It should be noted that $fresh(X_1 \ldots X_n)$ means that the entire message $X_1 \ldots X_n$ is fresh. It may be

because any of the X_i or all of the X_i are fresh. So given only $fresh(X_1 \ldots X_n)$, one can not tell with certainty that which sub-message is fresh. Similarly, any computable function by A of a fresh term is also fresh as given by SVO's Axiom 19.

(F3) $@_A[fresh(X_1 \ldots X_n) \Rightarrow fresh(F(X_1, \ldots, X_n))]$

For the same reason mentioned above, function F must depend on all of its arguments in order to guarantee fresh output.

4.2.2.5 Jurisdiction:

Like SVO's Axiom 20, the following axiom captures the jurisdiction of a principal for generating keys in DTPL.

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

That is, if a principal 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. Of course, the receiver does need to make sure that S sent M_k before it concludes. By P8, someone will receive M_k and apply J1 to know φ_k. In order to represent a principal's jurisdiction over generating keys, we restrict φ_k to be of type $A \overset{k}{\longleftrightarrow} B$ or $A \overset{x}{\mapsto} k$ (x is the key type in public-key system).

4.2.2.6 Nonce Verification:

The nonce verification axiom of SVO is used to transform the past (*said*) into present (*says*) using the notion of freshness (*fresh*). DTPL can clearly pin point a past time event using its explicit notion of *configuration*. That is why, DTPL introduces only a single action *send* to represent sending a message and benefits from its temporal operators to capture the exact timing of the

event. Therefore, we do not use the nonce verification axiom of SVO.

4.2.2.7 Key Agreement:

These axioms of SVO specifically target the protocols involving a key agreement such as Diffie-Hellman key exchange. We use the key agreement axiom of SVO in the following.

(K9) $@_A[knows(A \overset{\delta}{\mapsto} k_A) \wedge knows(B \overset{\delta}{\mapsto} k_B) \Rightarrow knows(A \overset{F_0(k_A,k_B)}{\longleftrightarrow} B)]$

Here, F_0 is some key-agreement function such that $F_0(k_A, k_B) = F_0(k_B, k_A) = k_{AB}$.

4.2.2.8 Symmetric Goodness:

We simply use the symmetric goodness axiom of SVO as follows:

(G1) $@_C[knows(A \overset{k}{\longleftrightarrow} B) \Leftrightarrow knows(B \overset{k}{\longleftrightarrow} A)]$

4.2.3 Originators of the Received Messages

This subsection is entirely devoted to the *source association axioms* of SVO because of the key role they play in verifying authentication in a protocol. Basically source association pertains to claiming that a message can only be bound to certain source who is responsible for originating that message. Before we actually translate the source association axioms, we introduce the notion of 'origination'. 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 notion of origination is not new, but has also been mentioned in the work of strand spaces in [FHG99]. Message origination can be captured with the help of following axiom.

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

That is, A originates N in M by sending M such that it never communicated any message M' that contained N.

Authentication protocols can be categorized according to different underlying cryptography used for their implementation. They also utilize different mechanisms to achieve authentication. The source association axioms in SVO and in others (like BAN, AT, GNY etc.) utilize the same categorization of authentication protocols. In particular, SVO ascribes two axioms for source association, one for a received message encrypted by a shared-key of two principals, and other for a received signed message. We briefly mention how authentication protocols can be broadly classified into three categories. We also present our formulation of the source association axioms according to each category in the following.

4.2.3.1 Symmetric-key Protocols:

Cryptographic protocols often make use of the symmetric-key encryption in order to achieve their authentication goals. The symmetric-key cryptography has the advantage that an encrypted message contains guarantee about the originator of a received message. That is, an encrypted message can originate only from the principals having access to the encryption key with which the message was encrypted. This is a direct consequence of the assumption of ideal cryptography[5]. 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. We axiomatize the source association notion of SVO (Axiom 5) as

[5]Most of the work in this field is based on this assumption. It simply says that a principal can not encrypt or decrypt a message without having the proper encryption or decryption key.

follows.

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

Since A knows the key k, generally A is either P or Q in the above axiom. Moreover, if a server generated the key k for P and Q, then A may also represent the server. Since a server never encrypts a message with the shared key of principals, B does not represent the server. Notice that the above formula captures the possibility of a message originated by both principals possessing the key k. This distinguishes our formalization of the notion of source association from others. SVO assumes that a principal can recognize any message that it has seen before using its notion of recognizability in $\langle X \rangle_{*C}$. Therefore, it restricts its attention to only those messages that are known to be coming form a principal other than him by adding $from$ with the received message in its axiom, that is, $C\ received\{X\ from\ B\}_k$ in Axiom 5. We make no such assumptions. Our axiomatization captures those situations also in which a principal's own originated message is redirected back to him in order to launch an attack. See [GF02] for such an attack on Woo-Lam one-way authentication protocol [WL92]. However, we do assume that A is able to distinguish X from any garbage value. This is necessary, otherwise, after decrypting any garbage value Y ($Y = \{X\}_k$), A would not know if the content of Y, i.e. X, was really encrypted with k or was just a random bitstream. Moreover, note that we do not need to include DTPL equivalent of $B\ has\ X$ (i.e. $knows(X)$) in the consequence of the above rule as done in Axiom 5 of SVO, because according to P1, B already knows what it sends.

4.2.3.2 Asymmetric-key Protocols

In asymmetric-key cryptography, a signed message originates from a principal who has access to the private-key with which the message was signed. We assume the private-key of a principal to be its safe secret. Since principal's *ids* are unique and their mapping to the private-keys are injective in asymmetric cryptography, signing a message by a principal's private-key assures the originator of the signed message. The asymmetric-key equivalent of the SVO source association Axiom 6 can be given as:

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

We have avoided using the extra notation introduced in SVO as in the above axiom it is assumed that applying k to $\{X\}_{k^{-1}}$ confirms that $\{X\}_{k^{-1}}$ is the results of signing X with the private-key k^{-1}. Similarly, we avoided using separate notations for encryption and signing as it can be simply understood from the syntax ($\{X\}_k$ represents an encrypted message whereas $\{X\}_{k^{-1}}$ represents a signed message).

4.2.3.3 Challenge-response Protocols

This is the last category that we use to identify the originator of a received message. None of the methods provide any axiom for this category in the entire BAN suite. This is because in the protocols using public-key cryptography, an encrypted message does not provide any guarantee in the identity of its sender. The reason being the obvious fact that anyone can encrypt a message masquerading someone else by using public-key of the intended recipient. That is why, we use

a challenge-response strategy that public-key protocols typically rely on in which they use an encrypted secret in their challenge messages in order to achieve authentication (for example, the Needham-Schroeder public-key protocol [NS78]). If a nonce or a pre-shared secret in an encrypted message is used as a challenge, the correct response may be to generate a reply containing that nonce or the secret. It ensures that the challenge is not only received by the intended principal but it also opened (decrypted) the challenge message in order to retrieve the secret and composed a reply. The challenger may need to check the structure of reply message to ensure that an intruder has not simply forwarded the challenge message back to the challenger [MFG06b].

Since public-key encryption does not provide any information regarding source association, $B \overset{\psi}{\mapsto} k$ can not be directly used in the source association axiom. We use the above challenge-response idea to identify the originator of a received message in public-key protocols as follows. If N is a fresh secret uniquely originated in an encrypted challenge message $\{M_N\}_k$ such that only a principal B has the corresponding private-key to obtain N from $\{M_N\}_k$, then the reception of a message having N in any form other than $\{M_N\}_k$ ensures that B knows $\{M_N\}_k$, decrypted it and released N in any form other than $\{M_N\}_k$. 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 has been released by B. The idea of treating security protocols as challenge-response protocols is not new. For example, similar idea has been presented, not in the logic-based setting though, in the authentication tests of strand spaces in [GF02].

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

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 $\{M_N\}_k$. The above axiom is the generalized form of the challenge-response based on public-key encryption. The condition that N must exist in M''_N in a form other than $\{M_N\}_k$ guards against the attack where an intruder simply forwards $\{M_N\}_k$ back to A without involving B at all. Therefore, if this condition is met, B must have received and decrypted $\{M_N\}_k$ and released N in any form other than $\{M_N\}_k$ in M'''_N.

So far, we have presented the extended DTPL framework that can be used to analyze authentication protocols in a proof-based environment. Next, we present how to apply the proposed framework in order to analyze a protocol.

CHAPTER 5
APPLICATIONS

5.1 Analyzing the Needham-Schroeder Secret-Key Protocol

The Needham-Schroeder shared-key protocol [BAN90] is a well-known protocol that has influenced the design of many authentication protocols (for example, Kerberos protocol developed at MIT was based on it [MNS]). We analyze its authentication property using the proposed axioms.

5.1.1 The Protocol Description

The Needham-Schroeder shared-key protocol is depicted in Fig 5.1. The goal of this protocol is to distribute a session key k_{AB} from a trusted server S to two principals A and B. Principals already possess secret shared-keys (k_{AS} and k_{BS}) with the server. N_a and N_b are the nonces of A and B respectively. The protocol begins by A, the initiator, sending its request to S in the first message. Upon receiving the first message, the server generates a session key k_{AB} and sends the second message to A. The initiator A extracts the relevant information and forwards the encrypted sub-message of its message to B. Upon receiving the message, B, the responder, generates a nonce

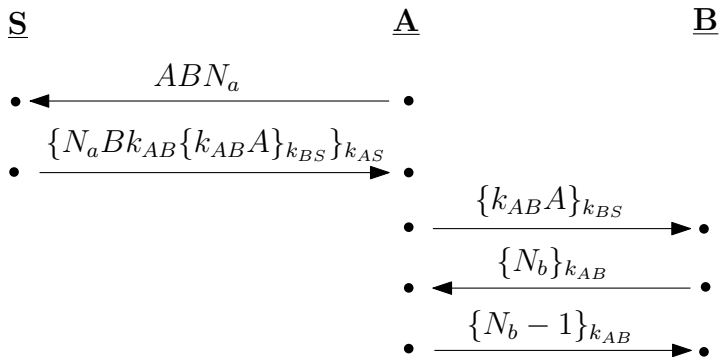

Figure 5.1: Needham Schroeder Shared-key (NSSK) Protocol.

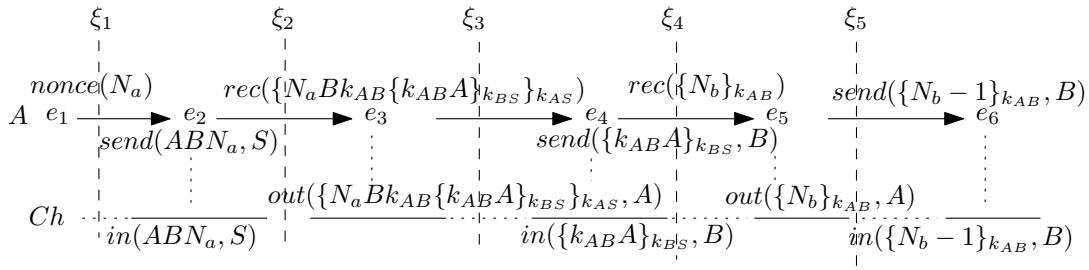

Figure 5.2: Life-cycle of Principal A in NSSK Protocol.

and encrypts it with the received session key and sends it to A. Finally, A replies B back in the last message. A subtracts 1 from B's nonce in order to differentiate the last two messages.

5.1.2 The Initiator's Perspective

We begin by analyzing the protocol from the initiator's perspective. The initiator's sequence of messages can be represented in terms of DTPL as follows.

1. $@_A[send(\{n_B-1\}_{k_{AB}},B) \wedge \mathsf{P}\,(rec(\{n_B\}_{k_{AB}}) \wedge \mathsf{P}\,(send(\{k_{AB}A\}_{k_{BS}},B)$

$\wedge\, \mathsf{P}\,(rec(\{N_ABk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}}) \wedge \mathsf{P}\,(send(ABN_A,S) \wedge \mathsf{P}\, nonce(N_a)))))]$

In the above, Y could be used instead of the past time operator P, but we stick to the the representation given in [CVB05a]. The life-cycle of initiator A is depicted in Fig. 5.2. Notice the vertical dashed lines in the figure indicating different configuration points in the run of the initiator. Each event e_i corresponding to each action of the initiator of the protocol changes A's configuration from ξ_{i-1} to ξ_i.

The initial set of assumptions related to principal A is as follows:

$@_A[* \Rightarrow knows(A \overset{k_{AS}}{\longleftrightarrow} S)]$

$@_S[* \Rightarrow knows(A \overset{k_{AS}}{\longleftrightarrow} S)]$

where $*$ (see Table 4.2) captures initial configuration of a principal for this protocol. Knowledge is

treated in DTPL as non-decreasing as formulated by K4. This monotonicity has been adopted by several researchers, such as BAN and its successors, where they treated belief as non-decreasing. We apply MP and K4 and use the above assumptions to get the following results at any configuration.

A1. $@_A[knows(A \overset{k_{AS}}{\longleftrightarrow} S)]$

A2. $@_S[knows(A \overset{k_{AS}}{\longleftrightarrow} S)]$

Similarly, we also assume the server's authority for generating session keys.

A3. $@_S[controls(A \overset{k}{\longleftrightarrow} B)]$

Applying O2, A1, A2, 1, and MP at configuration ξ_3:

2. $@_Q[\mathsf{P}\ Orig(\{N_a Bk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$

where $Q \in \{A, S\}$ originates the above message at $\xi \subset \xi_3$ (capturing 'sometime in the past'). Since a principal originating a message sends that message to some principal C, using O1, MP, and 2 at ξ:

3. $@_Q[send(\{N_a Bk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}}, C)]$

For this protocol to work properly, notice that Q should be actually the server S and not the principal A. But in our case, $Q \in \{A, S\}$. So in order to check the possibility if $Q = A$, we use a fairly straight forward method. We generalize the message by focusing on abstract message structure in the protocol without instantiating any message variable to any particular value. That is, Q sends $\{NYk\{kX\}_{k_1}\}_{k_2}$ (here, N is a nonce, X and Y are principal ids, and k, k_1, and k_2 are keys). We simply analyze all the sending events of principal A. We do not check the server because that what we are trying to prove. Examining all the actions corresponding to sending events and using the assumption of message *algebra freeness*, it is easy to realize that none of the sending action has a message

corresponding to $\{NYk\{kX\}_{k_1}\}_{k_2}$. Therefore, we remove the possibility that a message of this form is sent by any principal except S. Therefore, Q is in fact equal to S. From now on, we replace Q by S.

Since S must know its sending message before sending it. According to P1, MP, and 3 at a configuration $\xi' = \xi \backslash last(\xi)$ (capturing 'previous' Y in P1).

4. $@_S[knows(\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$

Therefore, S having the decryption key also knows the contents of the encrypted term. Using K3a, A2, MP, and 4 at ξ':

5. $@_S[knows(N_aBk_{AB}\{k_{AB}A\}_{k_{BS}})]$

Since A received $\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}}$ at ξ_3 (from 1) and knows the corresponding key (from A1), it knows the encrypted message as well as its contents. From K5, 1, and MP at ξ_3:

6. $@_A[knows(\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$

From K3a, A1, MP and 6 at ξ_3:

7. $@_A[knows(N_aBk_{AB}\{k_{AB}A\}_{k_{BS}})]$

Using K1a, 7 and MP:

8. $@_A[knows(\{k_{AB}A\}_{k_{BS}})]$

Note that A's knowing $\{k_{AB}A\}_{k_{BS}}$ does not mean knowing its content because A does not possess the key k_{BS}. Therefore, $\{k_{AB}A\}_{k_{BS}}$ appears as arbitrary term to A. From the analysis so far, it can be deduced that the server is familiar with the initiator's nonce and the responder's id and generated k_{AB}. Now using F1, 1, and MP at ξ_1:

9. $@_A[nonce(N_a) \Rightarrow fresh(N_a)]$

From N1 and N2 at ξ_1, we can say that no principal ever knew any message before ξ_1 containing

N_a as its subterm. Using F2, 9, and MP:

10. $@_A[fresh(N_aBk_{AB}\{k_{AB}A\}_{k_{BS}})]$

and from F3, 10, and MP:

11. $@_A[fresh(\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$

Therefore, from A's perspective, the server not only generated k_{AB}, but it generated it freshly. Furthermore, A also deems k_{AB} as the right session key to be used between him and B as stated by J1, 3, A3, and MP:

12. $@_A[knows(A \overset{k_{AB}}{\longleftrightarrow} B)]$

So far we have established A's understanding from the message exchanges between him and the server. Now we examine the interaction between principals A and B from the initiator's perspective. A sends $\{k_{AB}A\}_{k_{BS}}$ to B at ξ_4 and receives $\{N_b\}_{k_{AB}}$ at ξ_5. Using R2, 1 and MP at ξ_5:

13. $@_A[knows(\{N_b\}_{k_{AB}})]$

Notice that here we can not apply the origination formula for symmetric keys because even if A decrypts $\{N_b\}_{k_{AB}}$, it can not recognize N_b, violating the assumption in O2. From K3a, 13, 12, and MP at ξ_5:

14. $@_A[knows(N_b)]$

Since A had no knowledge of N_b before this point in time and there is no extra information in the message $\{N_b\}_{k_{AB}}$ that may help A recognize the correctness of this message, A can not conclude anything further. That is, even though A possesses k_{AB} and is able to extract N_b from its received message $\{N_b\}_{k_{AB}}$, it can not tell if the nonce N_b is actually the responder's nonce or any garbage value. This suggests a possible weakness in the protocol where an intruder could simply replace $\{N_b\}_{k_{AB}}$ by any random value X. As long as A does not recognize X (using any other means) as

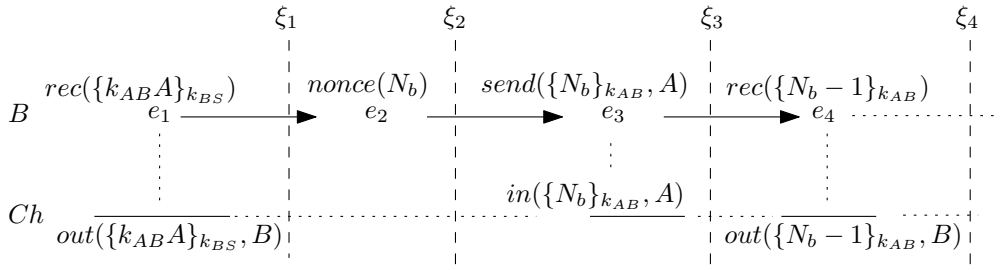

Figure 5.3: Life-cycle of Principal B in NSSK Protocol.

an improper message, the intruder can successfully make A feel that it has completed the protocol with B whereas B may not have been involved in the protocol at all. This is because we have not assumed that a principal can tell where a message is coming from by just looking at the structure of the message. That is why, A could not recognize N_b from the message $\{N_b\}_{k_{AB}}$. It should be obvious by now that this problem can be resolved by simply adding in the second last message something that A can recognize, such as B's id. This problem and the corresponding solution was first suggested in [GNY90]. This concludes our analysis of the protocol from the initiator's side. Next, we analyze the protocol from the responder's perspective.

5.1.3 The Responder's Perspective

The responder's life cycle is depicted in Fig. 5.3. The responder's sequence of actions can be written as:

1. $@_B[rec(\{N_B - 1\}_{k_{AB}}) \wedge \mathsf{P}(send(\{N_B\}_{k_{AB}}, A) \wedge \mathsf{P}(nonce(N_b) \wedge \mathsf{P}\ rec(\{k_{AB}A\}_{k_{BS}})))]$

The initial set of assumptions is as follows:

$@_B[* \Rightarrow knows(B \overset{k_{BS}}{\longleftrightarrow} S)]$ and $@_S[* \Rightarrow knows(B \overset{k_{BS}}{\longleftrightarrow} S)]$.

By MP, K4, and the above assumptions:

A1. $@_B[knows(B \overset{k_{BS}}{\longleftrightarrow} S)]$

A2. $@_S[knows(B \overset{k_{BS}}{\longleftrightarrow} S)]$

A3. $@_S[controls(A \overset{k}{\longleftrightarrow} B)]$

From O2, A1, A2, 1, and MP at ξ_1:

2. $@_Q[\mathsf{P}\ Orig(\{k_{AB}A\}_{k_{BS}})]$

Here, $Q \in \{B, S\}$. Using the similar reason as given before and assuming that the message algebra is freely generated, it can be seen that none of the B's sending actions correspond to a message of the form $\{kX\}_{k'}$. Therefore, from O1, MP, and 2 at $\xi \subset \xi_1$:

3. $@_S[send(\{k_{AB}A\}_{k_{BS}}, C)]$

From P1, MP, and 3 at $\xi' = \xi \backslash last(\xi)$

4. $@_S[knows(\{k_{AB}A\}_{k_{BS}})]$

From K3a, A2, MP, and 4 at ξ':

5. $@_S[knows(k_{AB}A)]$

From K5, 1, and MP at ξ_1:

6. $@_B[knows(\{k_{AB}A\}_{k_{BS}})]$

From K3a, A1, MP and 6 at ξ_1:

7. $@_B[knows(k_{AB}A)]$

Therefore, the responder is aware of the session key originated by the server. Note that unlike the initiator, B can not gain any assurance in the freshness of its received message. Therefore, from B's perspective, although the server has generated k_{AB}, it may not have generated it freshly. Therefore, examining the protocol from the responder's perspective also reveals a vulnerability to a known replay attack. If an attacker records one run of this protocol and subsequently learns the key k_{AB}, he can replay the message $\{k_{AB}A\}_{k_{BS}}$ to B. Being unable to tell that the key k_{AB} is not fresh,

B will accept it as a legitimate request to initiate a session using that key. The authors of BAN in [BAN90] mentioned this vulnerability in the protocol. That is why, BAN had to resort to the dubious assumption that $B\ believes\ fresh\ A \overset{k_{AB}}{\longleftrightarrow} B$ in order to attain authentication.

B also considers k_{AB} as the right session key to be used between him and A as stated by J1, 3, A3, and MP:

8. $@_B[knows(A \overset{k_{AB}}{\longleftrightarrow} B)]$

By F1, 1, and MP at ξ_2:

9. $@_B[nonce(N_b) \Rightarrow fresh(N_b)]$

By F3, 9, and MP:

10. $@_B[fresh(\{N_b\}_{k_{AB}})]$

11. $@_B[fresh(\{N_b - 1\}_{k_{AB}})]$

B receives $\{N_b - 1\}_{k_{AB}}$ at ξ_4. Since A can recognize $N_b - 1$, by O2, 8, 1, and MP at $\xi \subset \xi_4$:

12. $@_A[Orig(\{N_b - 1\}_{k_{AB}})]$

Since $N_b \neq N_b - 1$ (free message algebra), B could not have originated the above message. Therefore, assuming that the session key k_{AB} is fresh, the responder of the protocol guarantees that the initiator shares the same session key with him.

The purpose of the protocol was to distribute a session key between both principals. But analyzing the protocol from the initiator's perspective reveals that the initiator is not sure if the responder also possesses the session key at the end of the protocol. Furthermore, the responder is not sure that it shares the fresh session-key with the initiator until it assumes that k_{AB} is always freshly generated.

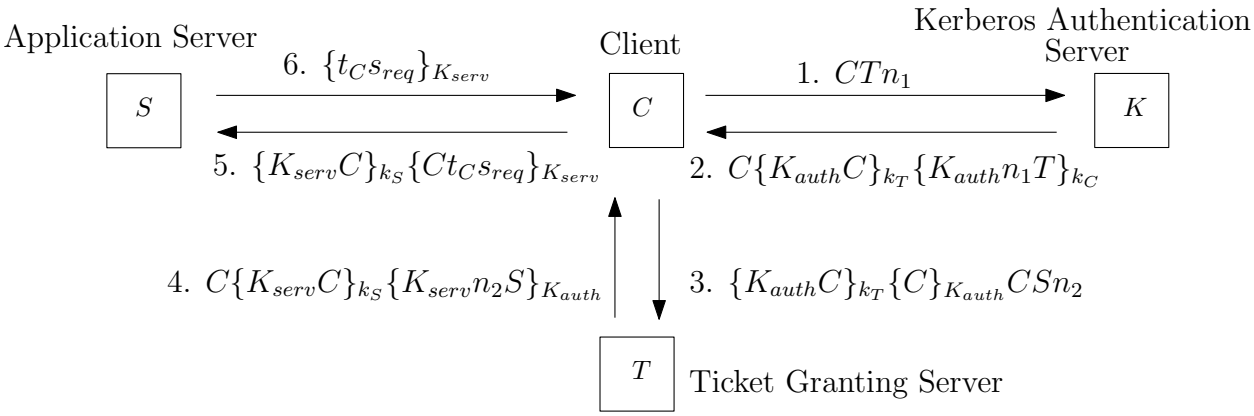

Figure 5.4: Message Exchanges Between the Client C and the Servers K, T, and S in the Kerberos protocol.

5.2 Analyzing Public-Key Extension of Kerberos-5

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

5.2.1 The Protocol Description

Kerberos [NYH05], [NT94] 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 (TGT)* from *kerberos authentication server (KAS) K*. *C* then presents *TGT* to *ticket granting server (TGS) T* and obtains a *service ticket (ST)*. Finally *C* uses the service ticket to authenticate itself to an *application server S*. Kerberos message exchanges are depicted in Fig. 5.4. For simplicity, we omit some of the message ingredients from the protocol that essentially do not affect the analysis at hand.

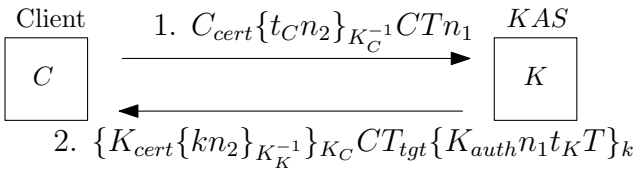

Figure 5.5: The First Round of Message Exchanges Between a Client C and the Kerberos Authentication Server K in PKINIT Protocol.

In Figure 5.4, $\{K_{auth}C\}_{k_T}$ = 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 [IET05] 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 Fig. 5.5.

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 the 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.

5.2.2 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 [MFG07a]. 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 $\{kn_2\}_{k_K^{-1}}$. The C's and K's runs of the first phase of the protocol are

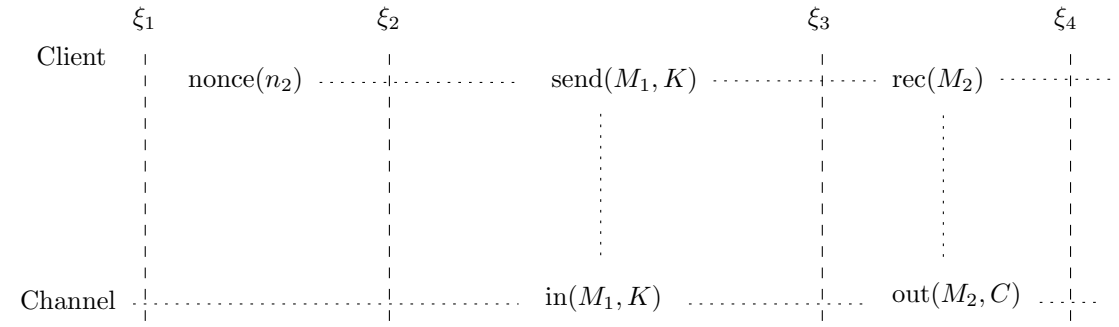

Figure 5.6: First Pass of the Client's Run in PKINIT Using DTPL (The client C sends its challenge n_2 and expects a message containing n_2 singed by the private-key of K.)

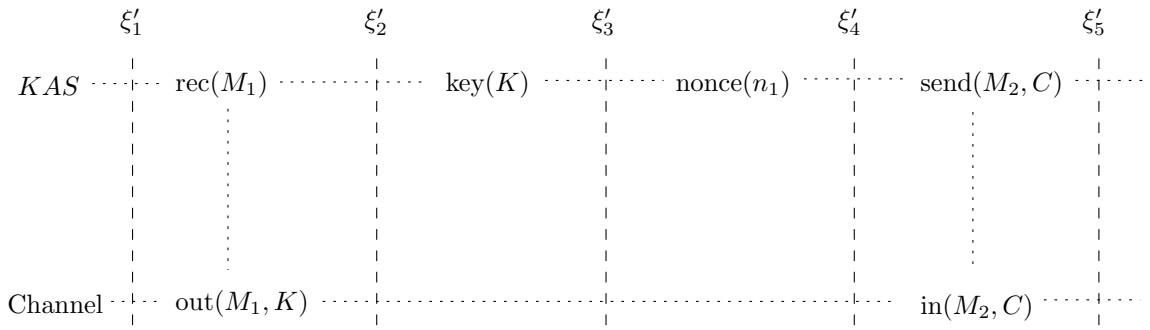

Figure 5.7: First Pass of the *KAS*'s Run in PKINIT Using DTPL (The server K responds to the client's challenge by sending a singed message containing n_2 along with a session key k.)

depicted in terms of DTPL in Fig. 5.6 and Fig. 5.7 respectively.

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

1. $@_C[rec(M_2) \wedge P\ (send(M_1,\ K) \wedge P\ nonce(n_2))]$

2.$@_K[send(M_2,\ C) \wedge\ P\ (nonce(n_1) \wedge\ P(key(k) \wedge\ Prec(M_1)))]$

Where, $M_1 = C_{cert}\{t_C n_2\}_{k_C^{-1}} CT n_1$ and $M_2 = \{K_{cert}\{kn_2\}_{k_K^{-1}}\}_{k_C} CT_{tgt}\{K_{auth} n_1 t_K T\}_k$.

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:

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

Where, * 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. 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. $@_K[rec(\{t_C n_2\}^{-1}_{k_C})]$ 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. $@_C[P\ Orig(\{t_C n_2\}_{k_C^{-1}})]$ by 3, O3, A2 and MP at ξ'_2.

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

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

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

10. $@_C[rec(\{kn_2\}_{k_v^{-1}})]$ by 9, C1and MP at ξ_4.

11. $@_C[rec(kn_2)]$ by 10, C4 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. $@_K[P\ Orig(\{kn_2\}_{k_v^{-1}})]$ 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, 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 $\{kn_2\}_{k_K^{-1}}$ 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*.

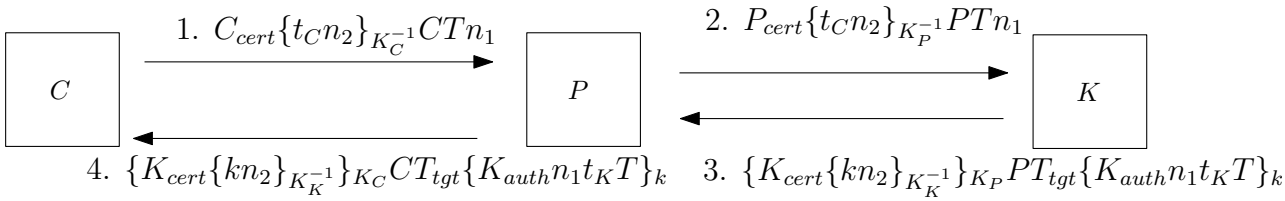

Figure 5.8: Attack on PKINIT in Which a Penetrator Plays Man-in-the-middle Between *C* and *K*.

5.2.3 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 [**?**] were the first to mention this attack on PKINIT-26. The attack, somewhat similar to that on the Needham-Schroeder public-key protocol in [**?**], exploits the above-mentioned weakness in the protocol in which ids of the principals are not tightly bound with the messages. Fig. 5.8 shows how it works.

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 $\{kn_2\}_{K_K^{-1}}$, 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.

CHAPTER 6
EXTENDING THE DISTRIBUTED TEMPORAL PROTOCOL LOGIC

After a protocol is represented in terms of DTPL, we investigate a participant's life-cycle using the challenge-response strategy. The analysis proceeds by first identifying the challenge generated by a participant in its life-cycle and then assuring that the reply message correctly represents the response to that challenge.

In order to achieve authentication goal, a variety of authentication protocols have been proposed in the literature. They can be broadly categorized according to the cryptographic approach taken (symmetric-key or public-key), numbers of trusted third parties to carry out some agreed function, and the type of authentication (one-way or two-way authentication) desired [CJ97]. We posit that irrespective of their particular implementation, authentication protocols follow a common challenge-response criterion in which a participant generates a challenge and waits for the proper reply. A correct reply convinces the challenger that authentication is achieved from its side of the protocol.

Authentication protocols employ different underlying cryptography and utilize several mechanisms to achieve authentication. For instance, some protocols use a nonce as a challenge (for example, the Needham-Schroeder public-key [NS78], the Andrew Secure RPC Handshake [Sat87], and the Otway-Rees [OR87] protocols) whereas others use a pre-shared secret (for example, the 802.11i protocol [IEE04]) to generate a challenge. If a nonce or a pre-shared secret in an encrypted message is used as a challenge, the correct response may be to generate a reply containing that nonce or the secret. It ensures that the challenge is not only received by the intended participant but it also opened (decrypted) the challenge message in order to retrieve the secret and composed a

reply. The challenger may need to check the structure of reply message to ensure that a penetrator has not simply forwarded its challenge message back to him. We will further elaborate on this issue in the rest of the document. In other cases where the nonce is not encrypted in the challenge message, the correct reply may be to construct a message containing the nonce and encrypted by the key only possessed by the legitimate responder. Nonces, timestamps, or sequence numbers also serve as a notion of freshness in many protocols [MFG06a].

The type of cryptography used by a protocol also heavily influence how the protocol implements its challenge-response strategy. For instance, in the protocols using public-key cryptography, an encrypted message does not provide any guarantee in the identity of its sender. The reason being the obvious fact that anyone can compose a message masquerading someone else by using public-key of the intended recipient. That is why, public-key protocols typically rely on an encrypted secret in their challenge messages in order to achieve authentication (for example, the Needham-Schroeder public-key protocol). Nevertheless, a signed message in public-key system ensures the originator of the message because the private-key of a participant is assumed to be its safe secret (for example, the Diffie-Helman key agreement in TLS protocol [DA99]). In the similar vein, the symmetric-key cryptography has the advantage that an encrypted message contains guarantee about the originator of the message. Therefore, the challenge does not have to be a secret any more. Rather, any random number or a timestamp can be sent as a challenge to the intended recipient, even in an un-encrypted form (for example, the Neuman-Stubblebine [NS93] and the Woo-Lam [WL92] protocols).[1] The correct reply should be a message encrypted under the symmetric-key containing the fresh challenge. Existence of a freshly generated random number or

[1]This is not always the case though. For instance, the Otway-Rees protocol uses symmetric-key cryptography and still encrypts its challenge messages.

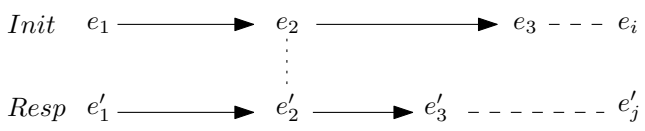

Figure 6.1: An Abstract Authentication Protocol

a timestamp simply ensures that the participants do not become victims of replay attacks in which a penetrator reuses a message from a previous run of the protocol [MFG06b].

6.1 The Analysis Steps

So far we have given a brief account on how security protocols can be seen in terms of a high-level challenge-response protocols. Now we use the above-mentioned concepts and describe our proposed strategy for analyzing cryptographic protocols. We take an abstract protocol and describe the important phases involved in its analysis.

Phase 1. Protocol Representation: As mentioned before, we use the DTPL's event structures to represent a protocol. Authentication protocols often comprise of two or more participants sending/receiving messages to/from each other. An abstract view of an authentication protocol is shown in the Fig. 6.1. Notice that each participant's life cycle can have a different length. That is, *i* is not necessarily equal to *j* in e_i and e'_j in Fig. 6.1. Moreover, events correspond to the actions from *send*, *rec*, *spy*, *nonce*, and *key*. The communication events such as (e_2, e'_2) represent (*send*, *rec*) pair. After a protocol is represented in terms of distributed life-cycles, each participant is analyzed separately.

Phase 2. Identifying the Challenge and Response: In this step, we focus on a participant's life-

cycle and identify the parameter n used as a challenge by that participant. Lets assume that the participant is playing as an initiator *Init* in the protocol. Assuming that n is fresh, we ascertain if n is originating at *Init*. To ensure that n is originating in a message M_c, the following conditions need to be checked:

- *Init* contains an event such that $\alpha(e_i) = send(M_c, A)$.
- $n \in subterm(M_c)$.
- $n \notin subterm(M)$ such that $\alpha(e_k) \in \{send(M, A), rec(M), spy(M)\}$ lies in the local life-cycle of *Init*, where $1 \leq k \leq i - 1$.

Assuming n to be uniquely originating (that is, it does not originate at any participant except *Init*), identify the expected response of the protocol at *Init*. In addition to having the intended structure of the response message, correct response M_r should observe the following conditions:

- *Init* contains an event e_j such that $j > i$.
- $\alpha(e_j) = rec(M_r)$.
- $n \in subterm(M_r)$.
- n lies as a new component in M_r.

Generally either the challenge message M_c contains n in the encrypted form discernable to only legitimate participants, or the response message M_r is in a form that can only be constructed by a legitimate participant. In either case, we need that the last condition hold. That is, n lies as a new component in M_r to assure that the response message is not a result of simply replaying the

challenge message.

Phase 3. Finding the Originator of the Response: Once the challenge and the response at a participant are identified, we investigate the originator of the response. In order to find the true originator of a response, following observations are made:

- In asymmetric-key cryptography, a signed message originates from the participant who has access to the private-key with which the message was signed. We assume the private-key of a participant to be its safe secret. Since participant's *ids* are unique and their mapping to the private-keys are injective in asymmetric cryptography, signing a message by a participant's private-key assures the originator of the signed message.

- In symmetric-key cryptography, an encrypted message can originate only from the participants having access to the encryption key with which the message was encrypted. This is a direct consequence of the assumption of ideal cryptography. Since in symmetric-key cryptography, a key is assumed to be a participant's safe secret, encrypting a message under symmetric-key ensures the possession of the key, and hence the origination of the message by a participant having that key.

- If n is a fresh secret uniquely originated in an encrypted challenge message M_c at a participant such that only a participant P has the key to decrypt M_c and obtain n, and M_r is the correct response, then the reception of M_r ensures its origination at P.

In the above, we have simply discussed the rationale behind the observations. We give formal representation for finding the originator of a message in terms of distributed temporal protocol logic

in the next section. Most of the formal frameworks use similar reasoning to find the originator of a received message, see [BAN90, GNY90, MFG06c, FHG99] for example. Observing the aforementioned challenge-response strategy followed by a broad range of cryptographic protocols, notice that these points suffice our purpose of analyzing the protocols at a higher level of abstraction. We elaborate this notion further along with an example in the rest of the document.

As a result of applying the above-mentioned observations, if only the intended participant specified by the protocol is identified as the originator of the response message, we proceed with the next step. But if the response message could have been originated by any participant P_i other than the one specified by the protocol, we examine the life-cycle of each such participant P_i in order to find out the event responsible for originating the response M. In particular, for each P_i:

- For all the events in the life-cycle of P_i, find an event e_i such that $\alpha(e_i) = send(M_r, B)$, $M \in subterm(M_r)$ and $M \notin subterm(M')$ in an event e_k, $\alpha(e_k) \in \{send(M', A), rec(M'), spy(M')\}$, where $1 \leq k \leq i - 1$.

- Existence of such event suggests possible attack on the protocol.

Phase 4. Matching the Parameters: Successful identification of the originator of the response message ensures the reception of the challenge message at the responder. This results into *agreement* in the parameters of the challenge message between the challenger and the responder. As suggested by Lowe [Low97], we check if the agreeing parameters include critical values such as the *ids*, nonces, and other secret variables used in the protocol. Lack of parameter matching in any of the critical values suggests possible security hole resulting into successful man-in-the-middle (MITM) attack.

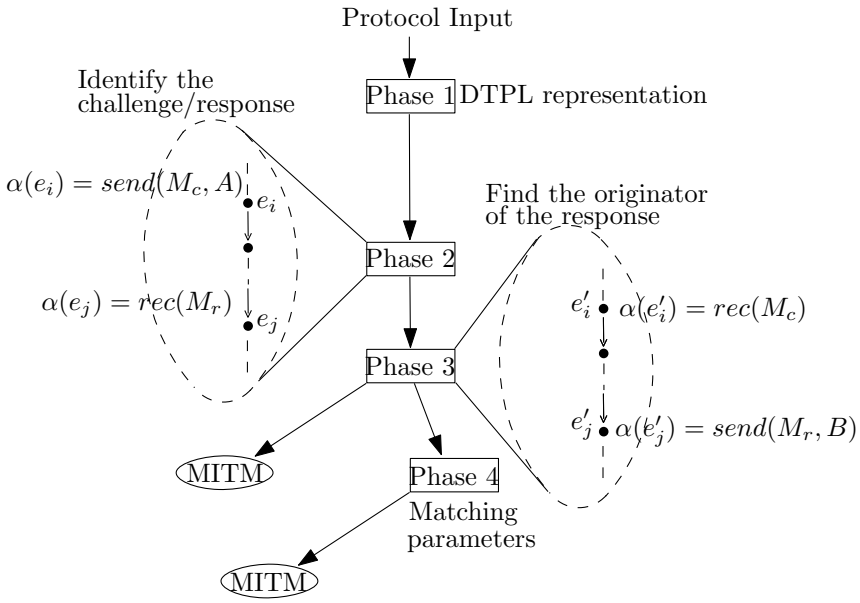

Figure 6.2: Phases of Our Proposed Method (Here the challenge term $n \in subterm(M_c)$ and $n \in subterm(M_r)$.)

An abstract view of the phases of overall analysis is shown in Fig. 6.2. As mentioned before, finding the unintended originator of a received message either directly at phase 3 or through discovering the mismatched parameter at phase 4 results into a successful MITM attack.

6.2 Originator of a Received Message

Based on different categories of authentication protocols described earlier, the authors in [GF02] proposed three different tests. These tests perceive security protocols as an implementation of a general challenge-response protocols. Depending upon the structure of the challenge and the response messages, an analyzer can choose any combination of these tests to either prove that the protocol achieves authentication or use them as a heuristic for finding attacks against incorrect protocols.

In order to translate the authentication tests in the DTPL framework, we need to modify the

existing framework. In this section, we first state some notions that are either defined in [GF02] or define the translation of it in the DTPL framework. Later we will use these definitions to construct higher-level authentication formulas. Moreover, we define some concepts such as *life span*, a basic ingredient behind authentication tests, and give new characteristics to the distributed channel. The overall idea behind the authentication tests is to trace the originating point of a received message in order to investigate who could have originated the messages of certain forms.

6.2.1 Message Components

Informally speaking, either the atomic terms or encrypted terms are called components of a message. If an encrypted term contains another encrypted term inside, then the outermost encrypted term is called component. Let t be a component of a message M, then $t \sqsubset M$[2], which we represent by M_t, i.e. t is a subterm of M. Let $C_{M_t} = \{send(M_t, B'), rec(M_t), spy(M_t)\}$. At the local configuration ξ_i of a participant, a component t is *new* in a message M if $t \sqsubset M$, $last(\xi_i) = e$, $\alpha(e) \in C_{M_t}$, and $\forall e' \in \xi_i \backslash e, \alpha(e') \notin C_{M'_t}$. That is, the participant never communicated a message containing that component previously.

$$@_A[send(M_t, B') \vee rec(M_t) \vee spy(M_t) \Rightarrow \mathsf{H}(\neg send(M'_t, B') \wedge \neg rec(M'_t) \wedge \neg spy(M'_t))]$$

Note that a component t in a message $M_t = abtcd$ is new even though if a participant communicated a message $\{t\}_K$ before.

[2] $\sqsubset$ represents the relation *subterm*. Roughly speaking, it represents the contents of a term.

6.2.2 Message Origination

If a participant sends a term N inside a message M, $N \sqsubset M$, such that he never communicated that term inside any message M' in the past then he originates the term in its sending message. Let $Comm_N = \{send(M'_N, B'), rec(M'_N), spy(M'_N)\}$. At the local configuration ξ_i of a participant, a term N originates in a message M if $N \sqsubset M$, $last(\xi_i) = e$, $\alpha(e) = send(M_N, B')$, and $\forall e' \in \xi_i \backslash e$, $e' \notin Comm_N$. That is, the participant never communicated any message containing N previously. We say the term N originates in the message M at ξ_A if it satisfies the following condition. We call it (O1)

$$\mu, \xi_A \Vdash_A send(M_N, B') \wedge \mathsf{H}(\neg send(M'_N, C') \wedge \neg rec(M'_N) \wedge \neg spy(M'_N))$$

6.2.3 Life Span

A portion of the local life-cycle of a participant is called its life span represented by $\langle Ev', \rightarrow' \rangle$, $Ev' \subset Ev_i, \rightarrow' \subset \rightarrow_i$ such that Ev' is backward closed under local causality $\rightarrow'^{+}$ until we reach an event e. That is, if $e' = last(\xi_i)$, $e' \in Ev'$ and $e'' \rightarrow'^{+} e'$, then $e'' \in Ev'$ until we encounter an event $e \in Ev'$. We also represent life span as $e \hookrightarrow e'$ where e and $e' \in Ev'$ are the first and the last events in the life span of a participant.

6.2.4 Path

Since upon receiving a message, the SSM identifies a unique participant sending that message whereas we find some amount of indetermination in the DTPL models. In the DTPL, many choices exist when tracing back the origin of a received message. Therefore, in order to derive the DTPL equivalent of the SSM inter-strand communication construct $\rightarrow$, we need to modify the channel.

We constrain the DTPL channel by modifying it as follows:

(C1′) $@_{Ch}[in(M, A') \Rightarrow B : send(M, A')]$,

(P3′) $@_A[rec(M) \Rightarrow Ch : out(M, A')]$,

(P4′) $@_A[spy(M) \Rightarrow Ch : (leak \wedge \mathsf{P}\ in(M, B'))]$

Now we prove the following communication formulas

(C4) $@_A[rec(M)] \Rightarrow @_B[\mathsf{P}\ send(M, A')]$

Proof. Its proof is a direct consequence of applying P3′, C2 and C1′. □

(C5) $@_A[spy(M)] \Rightarrow @_B[\mathsf{P}\ send(M, C')]$

Proof. Its proof is a direct consequence of applying P4′ and C1′. □

Notice that given a message is received by a participant, C4 and C5 essentially provide a shortcut by not considering the details of the communication through channel.

The life span of a participant in DTPL can emulate the arrow $\Rightarrow^+$ in SSM (see $run2str(u)$ in [CVB05b], for example) except that for honest participants, the local events corresponding to the actions $nonce(N)$ and $key(K)$ in DTPL have no explicit role in SSM. Life span together with the communication formulas C4 and C5 define a path. A path p in a model μ and configuration ξ is any finite sequence of events and edges $e_1 \mapsto e_2 \mapsto \cdots \mapsto e_k$ such that:

- $\bigcup_{x=1}^{k} e_x \subseteq \xi$
- $\alpha_i(e_1)$ is $send(M, A)$ and $\alpha_j(e_k)$ is $rec(M')$.
- $e \mapsto e'$ means

– Either life span $e \hookrightarrow e'$ with $\alpha_i(e) = rec(M)$ or $spy(M)$ and $\alpha_i(e') = send(M', A)$, or else

– $\alpha_i(e) = send(M, A)$ and $\alpha_j(e') = rec(M)$ (or $spy(M)$ if $e' \neq e_k$). The participants i and j are such that the messages in the *rec* and *send* actions are compatible and $e' \downarrow \supset e \downarrow$.

That is, a path starts from a sending event and terminates at the receiving event of an honest participant. Honesty is due to the assumption that $\alpha_j(e_k) \neq spy(M')$. It is worth mentioning that a participant is deemed honest if he plays by the rules of the protocol since nobody knows the intruder beforehand. If a message destined for participant B is received by B, it can terminate the *path* even if B intends to use the message for illegitimate activity in the future. Since some honest participant has sent the message to B, the latter will be considered legitimate in that activity. However, even though a spy event can lie within the path, it can not terminate the path. In general, only either the local sequence of events of a participant, or the common communication event specified by C4 or C5 constitutes a path. We use $|p|$ to represent the length and $l(p)$ to represent the last event of a path. For example, in the above path p, $|p| = k$ and $l(p) = e_k$.

6.2.5 Transformed Life

The life span $e \hookrightarrow e'$ of a participant i is called a transformed life for $a \in \mathbf{A}$ if

- $\alpha_i(e) = send(M, A)$ and $\alpha_i(e') = rec(M')$.
- $a \sqsubset M$.
- there is a new component t in M' such that $a \sqsubset t$.

We can formulate the above definition for a model μ and local configuration ξ_i as follows:

$$\mu, \xi_i \Vdash_i rec(M'_{t_a^{new}}) \wedge \mathsf{P}\ send(M_a, A)$$

where M_{t_n} represents a message M with t as its component such that a term n is a subterm of t. Moreover, t^{new} represents a new component. Since honest participants use the transformed life to authenticate other participants, no event corresponds to the $spy(M)$ action.

6.2.6 Transforming Life

The life span $e \hookrightarrow e'$ of a participant i is called a transforming life for $a \in \mathbf{A}$ if

- $\alpha_i(e) = rec(M)$ or $spy(M)$ and $\alpha_i(e') = send(M', A)$.
- $a \sqsubset M$.
- there is a new component t in M' such that $a \sqsubset t$.

For a model μ and local configuration ξ_i, transforming life satisfies the following:

$$\mu, \xi_i \Vdash_i send(M'_{t_a^{new}}, A) \wedge \mathsf{P}\ (rec(M_a) \vee spy(M_a))$$

Note that for honest participants, the transforming life can be easily recognized in DTPL as honest principals strictly follow the protocol rules. However, due to the absence of explicit intruder behavior in DTPL, an illegitimate participant takes part in transforming life only if it can construct an M'-producing S-bundle where S is the resource available to the intruder. In other words, $M' \in synth(analz(S))$ [CVB05b]. We will further elaborate this issue in the later part of this

section. [CVB05b] defines an M-producing S-bundle as follows. A bundle is M-producing if it contains a node labeled with $+M$ which is not connected via $\rightarrow$. An S-bundle is any bundle-like structure with only intruder strands, but excluding **M** and **K** strands, where receiving nodes $-M$ may not be connected via $\rightarrow$ provided that $M \in S$ where S is a set of messages. So an S-bundle which is M-producing is called an M-producing S-bundle..

6.2.7 Transformation Path

Let (e_i, t_i) be a pair where e_i is an event and t_i is the component of the message involved in $\alpha(e_i)$. Then the transformation path is a *path* in which if (e_i, t_i) then (e_{i+1}, t_i) unless a life span $e_i \hookrightarrow e_{i+1}$ occurs in the life-cycle of a participant X and t_{i+1} is new at ξ_X. Transformation path does not have to terminate at this point.

Note that in a transformation path if $t_i \neq t_{i+1}$, $a \sqsubset t_i$, and $a \sqsubset t_{i+1}$, then $e_i \hookrightarrow e_{i+1}$ is a transforming life for a.

6.2.8 Finding the Originator of a Message

Given a DTPL model μ and configuration ξ such that $e_1 \in \xi$ in (e_1, t_1), $\alpha(e_1) = rec(M_{t_1})$ and $a \sqsubset t_1$.

Then there is a transformation path p in μ, ξ such that a originates at p_1, $l(p) = e_1$, $t_{|p|} = t_1$, and $\forall i, a \sqsubset t_i$.

Proof. Let us consider the following transformation path p (indexed in reverse order):

$$e_{k+1} \mapsto e_k \mapsto \cdots \mapsto e_1$$

where each event e_i is paired with a component t_i in (e_i, t_i) such that $\forall j \in e_j, a \sqsubset t_j$. If a originates at t_{k+1} in (e_{k+1},t_{k+1}) then p is complete. So suppose that a does not originate at t_{k+1}.

- If $\alpha_A(e_{k+1}) = rec(M)$, then using C4, $B : send(M, A)$. Extend p backward to (e_{k+2}, c_{k+2}).
- If $\alpha_A(e_{k+1}) = spy(M)$, then using C5, $B : send(M, C)$. Extend p backward to (e_{k+2}, c_{k+2}).
- If $\alpha_A(e_{k+1}) = send(M, B)$, then
  - If t_{k+1} is new, using the definition of origination, there exists $e_{k+2} \hookrightarrow e_{k+1}$ such that $a \sqsubset t_{k+2}$ and $\alpha_A(e_{k+2}) = rec(M')$ or $spy(M')$, since a does not originate at t_{k+1}. Extend p backward to contain such (e_{k+2}, t_{k+2}).
  - If t_{k+1} is not new, then there exists $e_{k+2} \hookrightarrow e_{k+1}$ such that $\alpha_A(e_{k+2}) = rec(M')$ or $spy(M')$ and M' has a component t_{k+1}. Extend p backward to contain such (e_{k+2}, t_{k+1}).

Since a distributed life-cycle $\rightarrow^*$ defines a partial order of global causality on the set $Ev = \bigcup_{i \in Id} Ev_i$ of all events, the set Ξ of all global configurations constitutes a lattice, under inclusion, and has $\emptyset$ as the minimal element [CVB05a]. Therefore, following the path in the reverse order as described above, we will reach a point (e_j,t_j) where a originates. □

6.2.9 Restricted Intruder

We can put some bounds on the penetrator using the observations made in [CJM98]. These observations show how we can benefit by eliminating redundant operations while constructing new messages from a set of existing messages. For example, considering freeness assumption on message algebra, decrypting a message $\{m\}_K$ after encrypting m using the key K is deemed redundant.

Similarly, separating a message m_1m_2 after concatenating two messages m_1 and m_2 is a redundant operation. In its natural deduction style message derivation, [CJM98] calls it *normalized derivation*. It is observed that "any derivation tree for a message m using some assumptions can be transformed into a normalized derivation tree under the same assumptions." In fact, following the idea of normalized derivation, [CJM98] puts more restriction on the operations of an intruder by observing that "no introduction rule appears above an elimination rule in a normalized derivation tree." Here, introduction rules are covered by concatenation and encryption operations and elimination rules are covered by separation and decryption operations. The overall idea is simply that if we remove all the redundant penetrator operations in the construction of a message, then once a penetrator encrypts or concatenates a message, it can not gain any advantage by decrypting or separating the message afterwards.

Since all the penetrator operations can be transformed into an equivalent normalized form, we restrict our attention to only those network models in which all penetrator actions are restricted to its normalized activity. SSM uses the same idea in its *Normal Bundles* [GF02]. It simply asserts that in a bundle with no redundancy, every destructive (decryption or separation strand) edge precedes every constructive (encryption or concatenation strand) edge.

6.2.10 Honest Transformation

Since DTPL treats an intruder almost the same way as it treats an honest participant, the normal form for intruder strands in protocol bundles obtained in [GF02] does not have a DTPL counterpart [CVB05b]. Unlike DTPL, the SSM strictly specifies how an intruder constructs a message. The DTPL leaves the task up to the *analz* and *synth* functions. But we can benefit from the back-and-

forth translation between the capabilities of an intruder in the DTPL model and that in the SSM model given in [CVB05b] as follows:

> "If S be a set of messages. Then $M \in synth(analz(S))$ if and only if there exists an M-producing S-bundle."

If p is a transformation path in which p_i represents a pair (e_i, t_i) such that

- $\forall i, a \sqsubset t_i$.
- p_1 and $p_{|p|}$ lie at honest participants.
- $t_1 \neq t_{|p|}$.
- t_1 is of the form $\{h_1\}_{K_1}$.
- t_1 is not a proper subterm of any honest component.
- $K_1^{-1} \notin K_P$.

If α is the smallest index such that $t_\alpha \neq t_{\alpha+1}$, then p_α lies at honest participant and $p_\alpha \hookrightarrow p_{\alpha+1}$ is a transforming life for a.

Proof. (Sketch) If p_α lies at an intruder then according to the above conditions, the intruder produces new component $t_{\alpha+1}$ in a message M from its available resources S using the functions *synth* and *analz*. Note that *synth* and *analz* precisely cover the intruder attacks caused by the strands **C**, **S**, **E**, and **D** in SSM. The only intruder strand that can produce a new component is either **E** or **D** because **C** and **S** simply use previous components and do not introduce new one. Given $K_1^{-1} \notin K_P$, we are left with only **E** strand to construct M-producing S-bundle. With the help of only **E** strand,

suppose a penetrator constructs an M-producing S-bundle such that $t_{\alpha+1} \sqsubset M$. Now it is easy to see that no matter how many times the penetrator tries to construct an M-producing S-bundle, $t_\alpha \sqsubset t_{\alpha+1} \sqsubset t_{\alpha+2}$ and so on. So we reach a point where t_α is a proper subterm of $t_{|p|}$ resulting into contradiction. As mentioned above, the restricted class of intruder gets no benefit by using redundant operations like **E-D** or **C-S** which can always be eliminated as shown in [GF02]. In other words, after using redundancy elimination and applying the freeness assumption on message algebra, once a penetrator uses a **E** or **C** strand, it can not gain any benefit by using **D** or **S** strand afterwards. Therefore, a penetrator can not produce M and p_α can not lie at the intruder, but must lie at a legitimate participant. □

Similarly, if p is a transformation path in which p_i represents a pair (e_i, t_i) such that

- $\forall i, a \sqsubset t_i$.
- p_1 and $p_{|p|}$ lie at honest participants.
- $t_1 \neq t_{|p|}$.
- $t_{|p|}$ is of the form $\{h_1\}_{K_1}$.
- $t_{|p|}$ is not a proper subterm of any honest component.
- $K_1^{-1} \notin K_P$.

If α is the largest index such that $t_\alpha \neq t_{\alpha-1}$, then p_α is honest and $p_{\alpha-1} \rightarrow^+ p_\alpha$ is a transforming life for a. Its proof is very similar to the previous proof.

6.2.11 Authentication Tests

In this section, we use the above-mentioned concepts to derive authentication tests in DTPL equivalent to the one established in [GF02] using SSM.

6.2.11.1 Outgoing Test for the DTPL Model Configurations

The life span $e_0 \hookrightarrow e_1$ of a participant is an outgoing test for $a \in \mathbf{A}$ in $t = \{h\}_K$, $a \sqsubset t$ if:

- $\alpha(e_0) = send(M, B)$ and $\alpha(e_1) = rec(M')$.
- t is a component of M.
- The term a uniquely originates at e_0 in M.
- a does not occur in any component of M other than t.
- The term t is not a proper subterm of a component of any message in the local life-cycle of an honest participant.
- There is a new component t' of M' such that $a \sqsubset t'$.
- $K^{-1} \notin K_P$.

Let μ be a DTPL model and ξ be a configuration with $e_1 \in \xi$, and let $e_0 \hookrightarrow e_1$ be an outgoing test for $a \in \mathbf{A}$ in t, then

- There exist an honest participant i with the life span $m \hookrightarrow m'$, $(m, m' \in \xi)$, such that $\alpha_i(m) = rec(M'')$ and $\alpha_i(m') = send(\mathtt{M}, C)$.
- $a \sqsubset M''$ and t is a component of M''.

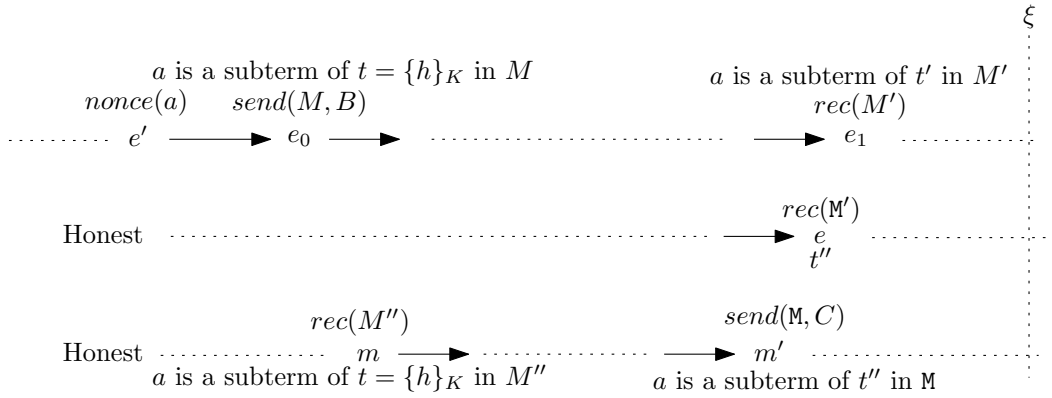

Figure 6.3: The Outgoing Test

- There is a new component t'' of $\mathtt{M}$ such that $a \sqsubset t''$.

Moreover, if:

- a occurs only in component $t'' = \{h'\}_{K'}$ of $\mathtt{M}$.
- The term t'' is not a proper subterm of any honest component.
- $K'^{-1} \notin K_P$.

then, there is an event e in the local life-cycle of an honest participant such that $\alpha(e) = rec(\mathtt{M}')$ and t'' is a component of $\mathtt{M}'$. The distributed life-cycle for the outgoing test at ξ is shown in Fig. 6.3.

Proof. (Sketch) Note that the life span $e_0 \hookrightarrow e_1$ represents a transformed life for a. We can benefit from finding the originator of a message presented in Section 6.2.8 in order to trace the transformation path p such that p in μ, ξ, a originates at p_1, $l(p) = e_1$, $t_{|p|} = t'$, and $\forall i, a \sqsubset t_i$. Since t' is new in e_1, $t \neq t'$. Moreover, since a does not occur in any component of M other than $t = \{h\}_K$, $t_1 = t$ and $t_1 \neq t_{|p|}$. Using the idea of honest transformation introduced in the previous section, the smallest index such that $t_\alpha \neq t_{\alpha+1}$ exists in which p_α is honest. Also note that according to the definition, $p_\alpha \hookrightarrow p_{\alpha+1}$ is a transforming life. Therefore, $t = t_1 = t_\alpha$ is a component of M'' and

$m = p_\alpha$. Now we have $m' = p_{\alpha+1}$, $\alpha_i(m') = send(\mathtt{M}, C)$, t'' is a new component of $\mathtt{M}$, $a \sqsubset t''$, and a occurs only in component $t'' = \{h'\}_{K'}$ of $\mathtt{M}$. Therefore, $t_{\alpha+1} = t''$. Now either $t' = t''$ or use the concept of honest transformation again to reach the conclusion. □

Now we represent the outgoing test in terms of a formula in DTPL.

$$@_A[rec(M'_{t_a'^{new}}) \wedge \mathsf{P}\ (send(M_{t_a}, B'))] \Rightarrow \bigvee_B @_B[\mathsf{P}\ (send(\mathtt{M}_{t_a''^{new}}, C') \wedge \mathsf{P}\ (rec(M''_{t_a})))]$$

where $B \in Princ\backslash\{A\}$ and m_{t_a} represents a message m with t as its component such that a term a is a subterm of t. Moreover, t^{new} represents a new component. In the above, $t_a = \{h\}_K$ in M_{t_a} such that $K^{-1} \notin K_P$ and a uniquely originates in M_{t_a}. Since a represents a term such as a nonce, the action $nonce(N)$ combined with the axioms N1 and N2 impose the condition of unique origination in DTPL. As the general requirements for the authentication test to hold, the term N does not occur in any component of M_{t_a} other than t and the term t is not a proper subterm of any honest component.

Finally, if a occurs only in the component $t_a''^{new}$ in $\mathtt{M}$ above, $t_a''^{new}$ is not a proper subterm of a component of any message of an honest participant, and $t_a''^{new} = \{h'\}_{K'}$ such that $K'^{-1} \notin K_P$, then there exists an honest event $rec(\mathtt{M}'_{t''})$ with t'' as a component.

6.2.11.2 Incoming Test for the DTPL Model Configurations

The local life-cycle $e_0 \rightarrow^+ e_1$ is an incoming test for $a \in \mathbf{A}$ in $t = \{h\}_K$ if:

- The term a uniquely originates in M at e_0.
- $\alpha(e_0) = send(M, B)$ and $\alpha(e_1) = rec(M')$.

- The term t is not a proper subterm of a component of any message in the local life-cycle of an honest participant.
- t is a new component of M' such that $a \sqsubset t$.
- $K \notin K_P$.

Let μ be a DTPL model and ξ be a configuration with $e_1 \in \xi$, and let $e_0 \rightarrow^+ e_1$ be an incoming test for $a \in \mathbf{A}$ in t, then

- There exist an honest participant i with the life span $m \hookrightarrow m'$, $(m, m' \in \xi)$, such that $\alpha_i(m) = rec(M'')$ and $\alpha_i(m') = send(\mathtt{M}, C)$.
- $a \sqsubset M''$ and t is a new component of $\mathtt{M}$ in which $a \sqsubset t$.

Proof. (Sketch) We trace the transformation path p as we did in the outgoing test such that p in μ, ξ, a originates at $p_1 = e_0$, $l(p) = e_1$, $t_{|p|} = t$, and $\forall i, a \sqsubset t_i$. Since t is new in e_1, $t_1 \neq t_{|p|}$. Using the result of honest transformation, the largest index α such that $t_\alpha \neq t_{\alpha-1}$ has $t_{\alpha-1}$ honest and $p_{\alpha-1} \hookrightarrow p_\alpha$ is a transforming life. Therefore, $t_{|p|} = t = t_\alpha$ is a component of $p_\alpha = m'$ where $\alpha(m') = send(\mathtt{M}, C)$. □

Let μ is a DTPL model of a protocol such that ξ is a configuration of μ equivalent to the bundle C mentioned in the above incoming test. Then, we can represent the incoming test as shown in Fig. 6.4.

The following DTPL formula expresses the incoming test.

$$@_A[rec(M'_{t_a^{new}}) \wedge \mathsf{P}\ (send(M_a, B'))] \Rightarrow \bigvee_B @_B[\mathsf{P}\ (send(\mathtt{M}_{t_a^{new}}, C') \wedge \mathsf{P}\ (rec(M''_a)))]$$

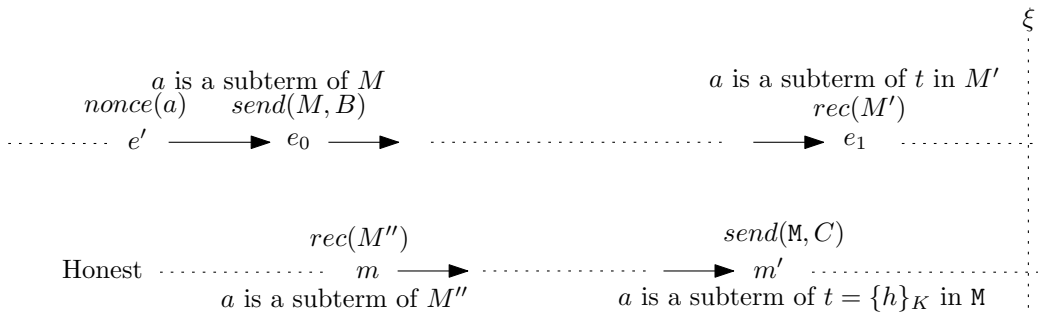

Figure 6.4: The Incoming Test

Here, $B \in Princ \backslash \{A\}$ and $t_a^{new} = \{h\}_K$ such that $K \notin K_P$ and a uniquely originates in M_a. Moreover, the term t is not a proper subterm of any honest component. As obvious, we can use the nonce N for the value of a.

6.2.11.3 Unsolicited Test for the DTPL Model Configurations

An event e such that $\alpha(e) = rec(M)$ is an unsolicited test for $t = \{h\}_K$ if for any a in M:

- $a \sqsubset t$ and t is a component of M.
- t is not a proper subterm of a component of any message in the local life-cycle of an honest participant.
- $K \notin K_P$.

Let μ be a DTPL model and ξ be a configuration with $e \in \xi$ such that e is an unsolicited test for $t = \{h\}_K$, then there exists an honest participant with event $m \in \xi$, $\alpha(m) = send(M', C')$ such that t is a component of M'.

Proof. (Sketch) In order to find the originator of message M, we trace the transformation path p in μ, ξ such that a originates at p_1, $l(p) = e$, $t_{|p|} = t$, and $\forall i, a \sqsubset t_i$ in (e_i, t_i). It is easy to see that the

penetrator can originate a message only by using its $send(M, B')$ action in which the originating term $t \sqsubset M$ can be obtained only by using *nonce* or *key* actions or by applying *analz* and *synth* functions on the set of messages available to the penetrator. Obviously *nonce* or *key* actions can not produce an encrypted term t. So we are left with the *analz* and *synth* functions in which *analz* can not originate a message (see the definition of message origination) whereas *synth* can originate a message t only by encrypting a message m with any key available to the penetrator K_P. Since $K \notin K_P$, using the assumption of free message algebra, the penetrator can not produce $t = \{h\}_K$. Moreover, since t is not a proper subterm of any regular component, it is a component of M'. □

The following DTPL axiomatization captures the unsolicited test.

$$@_A[rec(M_{t_a})] \Rightarrow \bigvee_{B \in Princ \setminus \{A\}} @_B[\mathsf{P}\ send(M'_{t_a}, C')]$$

where, $t_a = \{h\}_K$, $K \notin K_P$, and t is not a proper subterm of any honest component.

6.3 Verifying Authentication in the Needham-Schroeder Public-Key Protocol

We take the Needham-Schroeder public-Key protocol [NS78] as an example to illustrate the difference in the existing DTPL approach with the one presented in this thesis. The famous Needham-Schroeder public-key protocol is given as follows:

1. $A \rightarrow B : \{N_1 A\}_{K_B}$

2. $B \rightarrow A : \{N_1 N_2\}_{K_A}$

3. $A \rightarrow B : \{N_2\}_{K_B}$

6.3.1 Analyzing the Protocol Using Existing DTPL

In order to analyze a protocol, DTPL formalizes it by defining a sequence of actions (*send, rec, nonce*, and *key*) taken by honest participants executing the protocol. A protocol instantiation is a variable substitution σ defined for participant's ids, their nonces, and their keys. By instantiating a participant's role in a protocol, we get a concrete sequence of actions to be executed by that participant in a run of the protocol. In this way, we can obtain all possible concrete runs of each participant involved in any role in a protocol.

Assuming that $\sigma(a_i) = A$, $\sigma(a_j) = B'$, $\sigma(n_i) = N_1$ and $\sigma(n_j) = N_2$, if the honest principal A plays the role of an initiator:

$run_A^{Init}(A, B', N_1, N_2) =$

$\langle nonce(N_1).send(\{N_1 A\}_{K_{B'}}, B').rec(\{N_1 N_2\}_{K_A}).send(\{N_2\}_{K_{B'}}, B')\rangle$

or if A plays as a responder:

$run_A^{Resp}(B', A, N_1, N_2) =$

$\langle rec(\{N_1 B'\}_{K_A}).nonce(N_2).send(\{N_1 N_2\}_{K_{B'}}, B').rec(\{N_2\}_{K_A})\rangle$

All of the runs of principal A as an initiator or a responder can be expressed as

$Runs_A^{Init} = \bigcup_{\sigma \in Inst}\{run_A^{Init}(A, \sigma(b), \sigma(n_A), \sigma(n_b))\}$ or

$Runs_A^{Resp} = \bigcup_{\sigma \in Inst}\{run_A^{Resp}(\sigma(b), A, \sigma(n_b), \sigma(n_A))\}$

Finally, all of A's runs in any role are given as

$Runs_A = Runs_A^{Init} \bigcup Runs_A^{Resp}$, where no two $Runs_A^{Init}$ can have the same N_1, no two $Runs_A^{Resp}$ can have the same N_2, and N_1 of an initiator run must be different from the N_2 of any responder run.

Since models of a protocol are those network models in which honest principals follow the

protocol rules, A's sequence of actions can be obtained by an interleaving of prefixes of sequences in $Runs_A$. To model an intruder (dishonest principal), it has been shown that a single intruder is enough as it can be used to emulate multiple intruders [CVB05a]. While all honest participants strictly follow the rules of a protocol, there is, in general, no restriction imposed on an intruder. He can use any of the above-mentioned actions in any order along with an additional action *spy* to attack a protocol. Models of a protocol are obtained by an interleaving of prefixes of sequences of all the possible concrete runs of honest participants in any role of the protocol.

DTPL defines security properties like secrecy and authentication by the corresponding formula γ. In order for authentication property to hold for NS public-key protocol, the following formula should hold for all configurations ξ of the protocol model μ. If A as an initiator authenticates a responder at step 2 of the protocol, then using $auth_{A,B}^{Init,Resp,2}(A, B', N_1, N_2)$:

$$@_A[role_A^{Init}(A, B', N_1, N_2)] \Rightarrow @_B[\mathsf{P}_0\ send(\{N_1 N_2\}_{K_A}, A)]$$

where, $role_A^i(\sigma)$ can be obtained by $act_n \wedge \mathsf{P}(act_{n-1} \wedge \mathsf{P}(\ldots \wedge \mathsf{P}\ act_1)\ldots)$ if $run_A^i(\sigma)$ is $act_1 \ldots act_n$. In the above formula, the protocol step 2 requires that the responder send the message $\{N_1 N_2\}_{K_A}$ to A.

Similarly B acting as a responder authenticates an initiator at step 3 of the protocol, then using $auth_{B,A}^{Resp,Init,3}(A', B, N_1, N_2)$:

$$@_B[role_B^{Resp}(A', B, N_1, N_2)] \Rightarrow @_A[\mathsf{P}_0\ send(\{N_2\}_{K_B}, B)]$$

Here, the protocol step 3 requires that the initiator send the message $\{N_2\}_{K_B}$ to B. An attack on

a protocol is defined by any protocol model μ and configuration ξ for which the security formula does not hold, i.e. $\mu, \xi \nVdash \gamma$.

6.3.2 Analyzing the Protocol Using DTPL Tests

Now we present how a protocol can be analyzed at a higher level of abstraction using the authentication tests presented in Section 6.2. The principal A's run as an initiator and B's as a responder are given as:

$run_A^{Init}(A, B', N_1, N_2) =$

$\langle nonce(N_1).send(\{N_1 A\}_{K_{B'}}, B').rec(\{N_1 N_2\}_{K_A}).send(\{N_2\}_{K_{B'}}, B')\rangle$

$run_B^{Resp}(A', B, N_1, N_2) =$

$\langle rec(\{N_1 A'\}_{K_B}).nonce(N_2).send(\{N_1 N_2\}_{K_{A'}}, A').rec(\{N_2\}_{K_B})\rangle$

From the honest runs of both participants, it can be easily seen that the life cycle $e_0 \hookrightarrow e_1$ of A is an outgoing test for N_1 in $\{N_1 A\}_{K_{B'}}$ such that $\alpha(e_0) = send(\{N_1 A\}_{K_{B'}}, B')$ and $\alpha(e_1) = rec(\{N_1 N_2\}_{K_A})$. Similarly, the portion of the local life-cycle $e'_0 \hookrightarrow e'_1$ of B is an outgoing test for N_2 in $\{N_1 N_2\}_{K_{A'}}$ such that $\alpha(e'_0) = send(\{N_1 N_2\}_{K_{A'}}, A')$ and $\alpha(e'_1) = rec(\{N_2\}_{K_B})$.

Using the outgoing test for the responder:

$@_B[rec(\{N_2\}_{K_B}) \wedge \mathsf{P}\ (send(\{N_1 N_2\}_{K_{A'}}, A'))] \Rightarrow$

$\bigvee_{A \in Princ \setminus \{B\}} @_A[\mathsf{P}\ (send(M'_{t'_{N_2}}, C') \wedge \mathsf{P}\ (rec(M_{t_{N_2}})))]$

We say that there exist an honest participant A with the life span $m \hookrightarrow m'$ such that $\alpha_i(m) = rec(M)$, $\{N_1 N_2\}_{K_{A'}}$ is a component of M and $m \hookrightarrow m'$ is a transforming life for N_b. Since the only honest principal having $\alpha_i(m) = rec(M)$ and containing $\{N_1 N_2\}_{K_{A'}}$ as a component of M is an initiator with $run_A^{Init}(A, C', N_1, N_2)$. Therefore, the transforming life $m \hookrightarrow m'$ lies in $run_A^{Init}(A, C', N_1, N_2)$

such that $\alpha(m) = rec(\{N_1 N_2\}_{K_{A'}})$ and $\alpha(m') = send(\{N_2\}_{K_{C'}}, C')$. Notice that the variable C' could not be instantiated from the message $\{N_1 N_2\}_{K_{A'}}$ and hence, the outgoing test could not guarantee that the initiator of the protocol is aware of the corresponding responder B. This was the flaw discover by Lowe in [Low96]. As suggested by the analysis, Lowe fixed the flaw in the protocol by including the responder's id B in the message $\{N_1 N_2\}_{K_{A'}}$. The resulting protocol, named the Needham-Schroeder-Lowe (NSL) protocol is given as:

1. $A \rightarrow B : \{N_1 A\}_{K_B}$

2. $B \rightarrow A : \{N_1 N_2 B\}_{K_A}$

3. $A \rightarrow B : \{N_2\}_{K_B}$

The resulting NSL protocol has been proven secure against attacks by Dolev-Yao intruder using many other formal methods. As shown in [GF02], we can easily use the outgoing test to establish that the agreement property between the principals is fully satisfied in NSL. We do not repeat the analysis, but instead focus on one another issue. While analyzing the NS protocol, it was assumed that the principals already possessed the correct public-keys of their counterparts. Now we show that this assumption is not totally true. We demonstrate how the authentication tests can be used to find another flaw in the key-distribution part of the NS protocol. Below is the key-distribution portion of the NS protocol suggested in [NS78].

1. $A \rightarrow S : AB$

2. $S \rightarrow A : \{K_B B\}_{K_S^{-1}}$

3. $B \rightarrow S : BA$

4. $S \rightarrow B : \{K_A A\}_{K_S^{-1}}$

In the above protocol, both principals A and B request the public-key of their counterparts by a server S. The server provides the required keys after signing it with its private-key K_S^{-1}. Using the unsolicited test for principal A (the test can be applied on principal B in exactly the same way):

$$@_A[rec(M_{t_a})] \Rightarrow \bigvee_{B \in Princ \setminus \{A\}} @_B[\mathsf{P}\ send(M'_{t_a}, C')]$$

Here, $t_a = \{K_B B\}_{K_S^{-1}}$, $K_S^{-1} \notin K_P$. Assuming K_S^{-1} to be a safe secret of the server S, it can be easily seen that $B = S$. Notice that even though the server S must have generated the reply $\{K_B B\}_{K_S^{-1}}$, it can not be guaranteed that this message is freshly generated. We can use the incoming test to find the missing condition in which this guarantee could be achieved. That is, if A uniquely originates a term a in its sending message M_a and expects it back in its received message M'_{t_a}. It assures the freshness of the communicating messages. Since in the above protocol, principal A uses a non-uniquely originating data AB, the corresponding guarantees of the incoming test can not be assured. Therefore, this protocol is vulnerable to a replay attack based on reusing old keys. Other formal methods have also been used to highlight this weakness such as [OC02].

CHAPTER 7
CONCLUSION

The correctness of a protocol involves specifying various properties of the protocol and analyzing a set of known threats under some assumptions. We have presented a proof-based framework for the verification of security protocols using distributed temporal protocol logic. For this purpose, we have utilized the framework of SVO, which unifies four of its famous predecessors in a sound way. We have demonstrated how the similarities between both methods helped us utilize the existing SVO axioms in DTPL. We have also clarified some of the notions and extended the framework of SVO in our work. We have categorized authentication protocols as symmetric-key, asymmetric-key and challenge-response protocols. We have formalized the source association axioms of SVO in terms of these categories. Our work contains the advantages of both, the conciseness of the proof-based techniques as well as the clear representation of the temporal aspects of a protocol run of DTPL.

We have taken our formalization for protocol analysis at another level of abstraction. More specifically, we treat security protocols as a general challenge-response protocols in which each participant investigates the originator of its received messages by applying some tests. Successful identification of the originators of a received message guarantees that the critical parameters of the participants match with each other. This results into agreement between participants of a protocol that eventually leads towards successful authentication. On the other hand, lack of parameter matching among the participants of a protocol can be used as a guide towards finding a successful attack on the protocol.

Our work utilizes the authentication tests of strand spaces and formulates the corresponding

tests for distributed temporal protocol logic. Thus, we reap the benefits from the advantages of both formalisms. On one hand, DTPL not only has the power to represent a distributed system in terms of simple formulas, its capability to clearly represent different temporal activities of an agent makes it a better choice for our purpose. On the other hand, the existing DTPL framework provides an environment suitable for model-checking type of approach for the analysis of security protocols. Instead of examining all the possible attacks by an intruder in a model-checker, our formulation of authentication tests for DTPL makes it possible to analyze a protocol at a higher level of abstraction. The comparative analysis of a protocol presented in this thesis reflects the improvement in the DTPL framework and the corresponding ease with which protocol analysis can be carried out.

LIST OF REFERENCES

[Aba99] M. Abadi. "Secrecy by typing in security protocols." *Journal of the ACM*, **46**:749–786, 1999.

[AG97] Martín Abadi and Andrew D. Gordon. "A Calculus for Cryptographic Protocols: the Spi Calculus." In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 129–137, Zurich, Switzerland, April 1–4, 1997.

[AT91] Martín Abadi and M. Tuttle. "A Semantics for a Logic of Authentication." In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pp. 201–216, 1991.

[BAN90] Michael Burrows, Martín Abadi, and Roger Needham. "A Logic of Authentication." *ACM Transactions on Computer Systems*, **8**(1):18–36, February 1990.

[BB94] P. Bieber and N. Boulahia-Cuppens. "Formal development of authentication protocols." In *BCS-FACS Sixth Refinement Workshop*, 1994.

[CDF03] T. Coffey, R. Dojen, and T. Flanagan. "Formal verification: an imperative step in the design of security protocols." *Computer Networks*, **43**:601–618, 2003.

[CDL99] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. "A Meta-Notation for Protocol Analysis." In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, June 1999.

[Che80] Brian F. Chellas. "Modal Logic: An Introduction." Cambridge University Press, 1980.

[CJ97] J. Clark and J. Jacob. "A survey of authentication protocol literature: Version 1.0." Technical report, 1997.

[CJM98] Edmund M. Clarke, Somesh Jha, and Will Marrero. "Using State Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols." In *Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[CJM00] E. M. Clarke, S. Jha, and W. Marrero. "Verifying security protocols with Brutus." *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **9**(4):443–487, 2000.

[Coh00] E. Cohen. "TAPS:a first-order verifier for cryptographic protocols." In *13th IEEE Computer Security Foundation Workshop*, pp. 144–158, 2000.

[CVB05a] Carlos Caleiro, Luca Vigano, and David Basin. "Metareasoning about security protocols using distributed temporal logic." *Electronic Notes in Theoretical Computer Science*, **125**(1):67–89, 2005.

[CVB05b] Carlos Caleiro, Luca Vigano, and David Basin. "Relating strand spaces and distributed temporal logic for security protocol analysis." *Logic Journal of IGPL*, **13**(6):637–663, 2005.

[DA99] T. Dierks and C. Allen. "The TLS Protocol, Version 1.0.", January 1999. RFC 2246.

[DH76] W. Diffie and M. Hellman. "New Directions in Cryptography." *IEEE Transactions on Information Theory*, **22**(6):644–654, 1976.

[DH99] Naganand Doraswamy and Dan Harkins. "IPSEC: The New Security Standard for the Internet, Intranets, and Virtual Private Networks." Prentice Hall, 1999.

[DY83] D. Dolev and A. C. Yao. "On the Security of Public Key Protocols." *IEEE Transactions on Information Theory*, **29**:198–208, March 1983.

[EG83] S. Even and O. Goldreich. "On the security of multi-party ping-pong protocols." In *Proceedings of the 24th IEEE Symposium on the Foundations of Computer Science*, pp. 34–39, 1983.

[FG98] J. Herzog F. Thayer Fabrega and J. Guttman. "Strand spaces: Why is a security protocol correct?" In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pp. 160–171, May 1998.

[FHG99] F. Thayer Fabrega, J. Herzog, and J. Guttman. "Strand Spaces: Proving security protocols correct." *Journal of Computer Security*, **7**(1):191–230, 1999.

[GF02] Joshua D. Guttman and F. Javier Thayer Fábrega. "Authentication tests and the structure of bundles." *Theoretical Computer Science*, **283**:333–380, June 2002.

[GNY90] Li Gong, Roger Needham, and R. Yahalom. "Reasoning About Belief in Cryptographic Protocols." In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 234–248, May 1990.

[HS00] J. Heather and S. Schneider. "Towards automatic verification of authentication protocols on an unbounded network." In *13th IEEE Computer Security Foundation Workshop*, 2000.

[HT96] Nevin Heintze and J. Tygar. "A Model for Secure Protocols and Their Compositions." *IEEE Transactions on Software Engineering*, **22**(1):16–30, January 1996.

[IEE04] "IEEE P802.11i/D10.0. Medium Access Control (MAC) security enhancements,a mendment 6 to IEEE Standard for local and metropolitan area networks part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications.", April 2004.

[IET05] IETF. "Public Key Cryptography for Initial Authentication in Kerberos.", 1996–2005. INTERNET DRAFT.

[KD97] R. Kemmerer and Z. Dang. "Using the ASTRAL model checker for cryptographic protocol analysis." In *DIMACS Workshop on Design and Formal Veification of Security Protocols*, 1997.

[Kem89] R. Kemmerer. "Using formal methods to analyze encryption protocols." *IEEE Journal on Selected Areas in Communications*, **7**(4):448–457, 1989.

[KMM94] R. A. Kemmerer, Catherine A. Meadows, and J. Millan. "Three Systems for Cryptographic Protocol Analysis." *Journal of Cryptology*, **7**(2):79–130, 1994.

[Low96] Gavin Lowe. "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR." In Tiziana Margaria and Berhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 2nd International Workshop TACAS'96*, LNCS 1055, pp. 147–166. Springer Verlag, March 27–29, 1996.

[Low97] Gavin Lowe. "A Hierarchy of Authentication Specification." In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 31–43, 1997.

[LR92] D. Longley and S. Rigby. "An automatic search for security flaws in key management schemes." *Computers and Security*, **11**(1):75–90, 1992.

[Ltd93] Formal Systems (Europe) Ltd. "Failures Divergence Refinement-User Manual and Tutorial, version 1.3.", 1993.

[MCJ97] Will Marrero, Edmund Clarke, and Somesh Jha. "A Model Checker for Authentication Protocols." In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, p. 19 pages, September 3–5, 1997.

[Mea96] Catherine A. Meadows. "Analyzing the Needham-Schroeder Public Key Protocol: A Comparison of Two Approaches." In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *4rd European Symposium on Research in Computer Security, ESORICS'96*, LNCS 1146, pp. 351–364, September 25–27, 1996.

[Mea03] Catherine Meadows. "Formal methods for cryptographic protocol analysis: Emerging issues and trends." *IEEE Journal on Selected Areas in Communications*, **21**(1):44–54, January 2003.

[MFG06a] S. Muhammad, Z. Furqan, and R. K. Guha. "Designing Authentication Protocols: Trends and Issues." In *Proceedings of the 5th International Conference on Networking (ICN), IEEE Computer Society Press*, April 2006.

[MFG06b] S. Muhammad, Z. Furqan, and R. K. Guha. "Understanding the Intruder through Attacks on Cryptographic Protocols." In *Proceedings of the 44th ACM Southeast Conference (ACMSE2006)*, pp. 667–672, March 2006.

[MFG06c] Shahabuddin Muhammad, Zeeshan Furqan, and Ratan Kumar Guha. "Logic-Based Formal Analysis of Cryptographic Protocols." In *Proceedings of the 14th IEEE International Conference on Networks (ICON)*, September 2006.

[MFG07a] Shahabuddin Muhammad, Zeeshan Furqan, and Ratan Kumar Guha. "Analyzing Authentication in Kerberos-5 using Distributed Temporal Protocol Logic." *Journal of Information and Computing Science*, 2007.

[MFG07b] Shahabuddin Muhammad, Zeeshan Furqan, and Ratan Kumar Guha. "A Logic Based Verification Framework for Authentication Protocols." *International Journal of Internet Technology and Secured Transactions*, 2007.

[MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern. "Automated Analysis of Cryptographic Protocols Using Murϕ." In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp. 141–151, May 1997.

[MNS] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. "Kerberos Authentication and Authorization System." Project Athena Technical Plan, Section E.2.1.

[NS78] Roger M. Needham and M. Schroeder. "Using Encryption for authentication in large networks of computers." *Communications of the ACM*, **21**(12):993–999, December 1978.

[NS93] B. Clifford Neuman and Stuart G. Stubblebine. "A Note on the Use of Timestamps as Nonces." *ACM Operating Systems Review*, **27**(2):10–14, April 1993.

[NT94] C. Neuman and T. Tso. "Kerberos: An Authentication Service for Computer Networks." *IEEE Communications*, **32**(9):33–38, September 1994.

[NYH05] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. "The Kerberos Network Authentication Service (V5).", 2005.

[OC02] Susan Older and Shiu-Kai Chin. "Formal Methods for Assuring Security of Protocols." *The Computer Journal*, **45**(1):46–54, 2002.

[Oor93] Paul C. van Oorschot. "Extending cryptographic logics of belief to key agreement protocols." In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pp. 233–243, November 1993.

[OR87] D. Otway and O. Rees. "Efficient and timely mutual authentication." *ACM Operating Systems Review*, **21**:8–10, January 1987.

[Pau98] Lawrence C. Paulson. "The Inductive Approach to Verifying Cryptographic Protocols." *Journal of Computer Security*, **6**:85–128, 1998.

[PL85] H. Kirchner P. Rety, C. Kirchner and P. Lescanne. "NARROWER: A new algorithm for unification and its applications to logic programming." *Proceedings of RTA'85 Lecture Notes in Computer Science*, **202**:141–157, 1985.

[Sat87] M. Satyanarayanan. "Integrating Security in a Large Distributed System." Technical Report CMU-CS-87-179, CMU, 1987.

[SBP01] D. Song, S. Berezin, and A. Perrig. "Athena: a novel approach to efficient automatic security protocol analysis." *Journal of Computer Security*, **9**:47–74, 2001.

[SC01] Paul Syverson and Iliano Cervesato. "The Logic of Authentication Protocols." In *Foundations of Security Analysis and Design*, LNCS 2171, pp. 63–136. Springer Verlag, 2001.

[SO94] P. Syverson and P. Van Oorschot. "On Unifying Some Cryptographic Protocol Logics." In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 14–28, 1994.

[SV93] M. Shand and J. Vuillemin. "Fast Implementations of RSA Cryptography." In *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pp. 252–259, 1993.

[Win87] Glynn Winskel. "Event structures." In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Applications and relationships to other models of concurrency*, LNCS 255, pp. 325–392. Springer Verlag, 1987.

[WL92] Thomas Y. C. Woo and Simon S. Lam. "Authentication for Distributed Systems." *Computer*, **25**(1):39–52, January 1992.

[WL93] Thomas Y. C. Woo and Simon S. Lam. "A Semantic Model for Authentication Protocols." In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 178–194, 1993.