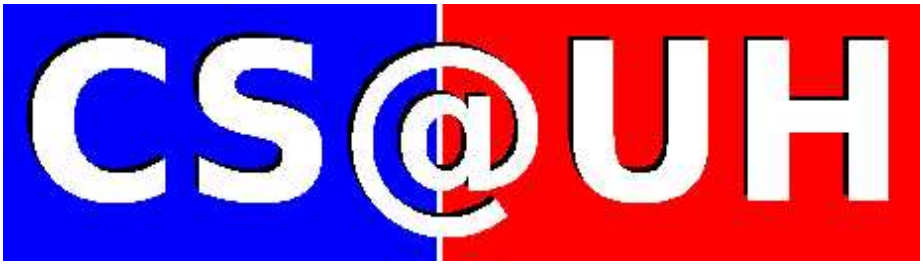

A Natural Logic for Checking Old Message Attacks [1]

Zhiyao Liang, Rakesh M. Verma

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report Number UH-CS-05-02

February 15, 2005

Keywords: Security, Protocol, Authentication, Freshness

Abstract

Freshness is the central issue of authentication in communication security protocols. The attacker can let an honest principal accept old message components sent in previous sessions. This kind of attack, called *old message attack*, is common and simple for the attacker, but nontrivial to check. We define a model to describe the honest principals' requirement, called labeled strand, which can work as a framework for reasoning about general kind of attacks. We also define a logic that includes the rules to describe the labeled strand, and the rules currently designed for reasoning about old message attack. This logic is sound and complete for checking old message attacks, and it has polynomial time implementation. The algorithm is simply to compute the closure of the rules and has efficient features.

[1] Research partially supported by NSF grant CCF 0306475

A Natural Logic for Checking Old Message Attacks[1]

Zhiyao Liang Rakesh M. Verma

Abstract

Freshness is the central issue of authentication in communication security protocols. The attacker can let an honest principal accept old message components sent in previous sessions. This kind of attack, called *old message attack*, is common and simple for the attacker, but nontrivial to check. We define a model to describe the honest principals' requirement, called labeled strand, which can work as a framework for reasoning about general kind of attacks. We also define a logic that includes the rules to describe the labeled strand, and the rules currently designed for reasoning about old message attack. This logic is sound and complete for checking old message attacks, and it has polynomial time implementation. The algorithm is simply to compute the closure of the rules and has efficient features

Index Terms

Security, Protocol, Authentication, Freshness, Logic, Old message, Completeness, Soundness, Polynomial time

1. Introduction

Security of communication protocols is important in this age when computer communication is ubiquitous. An important research direction in verifying communication protocols is checking attacks while assuming perfect encryption. The attacker can work as a middleman to play with different sessions and different principals. This problem is surprisingly hard. The simple and famous attack on the Needham-Schroeder protocol was found 17 years after publication by Lowe [21, 16, 17]. Researches have shown that to verify the security of communication protocols in general is undecidable [13, 9], but for finite sessions it is decidable and NP-complete [2, 25]. Many researchers follow the attacker model developed by Dolev and Yao [8]. There are approaches that use general verification tools, for example, Paulson uses the theorem prover Isabelle [26]. General model checkers are also used such as FDR [17, 15] and Murφ [14]. Automata are also used as a verification theoretic tool [20]. There are also special purpose model checkers to verify security protocols, such as Athena [28, 29], Brutus [6], and OFMC [3]. NRL protocol analyzer is written in Prolog as special-purpose tool [18]. The problem of checking attacks can also be reduced to solving constraints [19]. The common feature of these approaches is to enumerate all possible behaviors of the attacker (interleaving and constructing messages) to find the scenarios in which the attacker can succeed.

1.1 Motivation 1: Complex Freshness Tracking

Freshness is the central and fundamental issue in authentication of communicating protocols [11]. The purpose of using the fresh nonces in protocols is to let a principal A make sure that the other principals A is talking with are active in the current session and that A will not receive old messages. The basic form of freshness challenge from a principal A is to let A to send out a message containing some fresh items, then A expects to receive the fresh items back on the reply. However freshness tracking can be complex and nontrivial to understand without mechanical help. The following two protocols are examples to show this phenomenon (notations are explained in section 2).

Protocol 1: In this protocol X, M, N_A and W are all supposed to be fresh message components. Why A and B will only accept fresh components?

1. $A \rightarrow B: A$
2. $B \rightarrow A: \{X, M\}^{\rightarrow}K_A$
3. $A \rightarrow B: \{X, N_A\}^{\rightarrow}K_B$

[1] Research partially supported by NSF grant CCF 0306475

4. B → A: N_A, $\{M, W\}^{\rightarrow}K_A$

Protocol 2: This is the well known Yahalom protocol (cited in [4] as personal communication 1988, due to Yahalom). Why A and B can only accept fresh components (if they are supposed to be fresh) while S can accept old N_A and N_B? Then if the second message changes to 2. B → S: B, $\{A, N_A\}^{\leftrightarrow}K_{BS}$, $\{B, N_B\}^{\leftrightarrow}K_{BS}$, A can accept an old N_B.

1. A → B: A, N_A
2. B → S: B, $\{A, N_A, N_B\}^{\leftrightarrow}K_{BS}$
3. S → A: $\{B, K_{AB}, N_A, N_B\}^{\leftrightarrow}K_{AS}$, $\{A, K_{AB}\}^{\leftrightarrow}K_{BS}$
4. A → B: $\{A, K_{AB}\}^{\leftrightarrow}K_{BS}$, $\{N_B\}^{\leftrightarrow}K_{AS}$

In the above examples we are not considering the trick cases when a sophisticated attacker tries to interleave messages of parallel sessions and steal secrecies then decrypt messages. We are considering the most intuitive cases where the attacker just sends an old message component in the same position in some old session as a new one. And the attacker does not need to break encryptions, he can use the whole encrypted terms as a building blocks to build the faked messages.

1.2 Definition of OMA - Old Message Attack

We define **Old Message Attack** (OMA) by describing the capability and incapability of the attacker.

- The attacker only breaks a message by separating the elements of the unencrypted lists in the message. The attacker does not do decryption. If a list contains another list, the attacker can continue to break the inside list. The smallest term that the attacker can achieve by breaking a message (as a list) is called a **block.** For example, in the message of [A, N_A, [X, Y], $\{G, H\}^{\rightarrow}K_B$], there are 5 blocks: A, N_A, X, Y, and $\{G, H\}^{\rightarrow}K_B$. Positions can be assigned to these 5 blocks sequentially.
- A block is the smallest building unit for the attacker to construct a fake message. The attacker can construct a fake message by using some blocks sent in the previous sessions together with some new blocks. If several old blocks are used together in a message, then those old blocks may come from different old sessions.
- The new blocks can not change its supposed position in the faked message. For example if the 5^{th} block in the new message 3 can only be used by the attacker as the 5^{th} block in the faked message 3.
- Similarly, the old block used by the attacker must be in the same position in the protocol as the new one. E.g., the 5^{th} block of an old message 3 can only be used as the 5^{th} block of the faked message 3.
- The old session where the old block comes from must have the participants playing the same roles as in the new session. For example, if an old block is used by the attacker in the second message sent from A to B in the new session, then in the old message, which must be a second message, where the old block comes from, the sender is A and the receiver is B.
- There are no parallel current sessions. The only concepts about time epochs are the many old sessions, and the single new and current session. So the attacker can not interleave messages of different current parallel sessions.
- The attacker can not be a regular participant in the protocol.
- The attacker can intercept any messages sent by regular principals. And the attacker can possibly prevent a regular principal from receiving a message sent to it.
- A regular principal needs to finish its role (all the sending and receiving tasks, or its strand, defined later) defined in the protocol in order to finally accept any value. (This property is common for any kind of authentication attacks. For secrecy attack, the attacker can steal the secrecy and does not need to wait for the regular principal to finish its role in the current session.)
- If a regular principal P receives and sees a bit string that P can not recognize or understand, e.g., an encrypted term that P can not decrypt, P will not care about it. So if P receives such bit strings and they are supposed to be equal or different, P will not bother to do this kind of checking.
- If in the protocol, a regular principal P accepts (receives and sees) an old atomic term X that supposed to be fresh (defined as SDC in section 2.1), we can say that there is an OMA of X to P.

Observations. In OMA, encryption using some secret keys is not important to show the identification, since the attacker will not try to construct an encryption, and the old blocks in the fake message will come from the supposed sender. The only thing matters is whether the block is fresh or not. If a principal receives a fresh block, then the block must come from the supposed honest sender in the current session. In OMA, freshness (of a block) means honesty (sent by the supposed sender in the current session). In an old block, nothing is fresh. And a block containing a fresh subterm

[1] Research partially supported by NSF grant CCF 0306475

must be a fresh block. A fresh block may contain some old subterms.

1.3 Motivation 2: OMA Can Be Severe

OMA will cause honest principals to accept faked values. Further more, the attacker can take the advantage of the weakness of the protocol to launch Denial-Of-Service (DOS) attacks by repeatedly sending old messages components. For example, in Yahalom protocol, the server will always accept the second message and do the computation, no matter the message is honest or faked.

1.4 Motivation 3: OMA is Nontrivial to Check

Even with current successful algorithms, OMA can be nontrivial to check. The reason is that those algorithms usually will enumerate all of the possible behaviors of the attacker. In OMA, when the attacker tries to construct a fake message, for each block in the message, the attacker can have two choices: using the honest new block sent in the current session, or the old one in some old session. Suppose a message has N blocks, then there can be 2^N possible ways. The exponential number of possibilities will be more severe for a long message which is a list of many items.

Consider the algorithms of MS constraints [19] and Athena [29] as examples. Suppose MS constraints is customized for OMA, so there is only one interleaving to consider: messages of the new session after the messages of the old session (only one old session needs to be considered, proved in lemma 1), and only pair rule and unification rule is used. Every time the unification rule is applied, there can be two terms chosen from the knowledge in the history to do the unification, the old block and the new block. Similarly for Athena, suppose Athena has been customized with OMA such that there are only two sets of regular strands included in the state, the set of strands from an old session, and a set of strands from the current session, then in the next state function, a goal which is a block, can have two possible goal bindings, one is the old block sent from an R penetrator strand, the other is the new block also sent from an R penetrator strand. There can be many guesses and attempts before the result can be computed, especially for the cases where the freshness challenge is checked later than a message component is received. For example, in Protocol 1, A will not know X and M are fresh until message 4. If we need to check the possible OMA for different principals, then MS constraints and Athena will need to run separately for each principal with different inputs of formulas or goals, which will cost more computation time.

1.5 Motivation 4: Belief Logic

BAN logic [4] was published by Burrows, Abadi and Needham in 1989, which is the influential work among the first attempts to formalize the description and analysis of authentication protocols. BAN logic takes the form of beliefs in the sense like every regular principal is an intelligent party and will do analysis and checking of the messages. This can naturally describe our intuition. Many papers are published in the 90s following BAN logic's style [12, 1, 5, 31, 30, 10, 27]. What seems to be missing is a logic that can integrate both the natural intuitions to describe the behavior of the real world of protocol executions, and some rigor computation model to achieve preciseness and confidence.

However, BAN logic has been criticized in the literature [23, 24]. And we do not consider BAN logic is a very practical solution mainly due to the following reasons:

1) BAN logic has to generate the idealized version of the protocol before it can be analyzed. But the way to generate the idealized version depends on peoples understanding and intuition of the meaning of the protocol, which can sometimes be subjective with different people and vague to define. The following is the simplified version of Yahalom [18] (Protocol 2), based on its idealized version.

Protocol 3: The simplified version of Yahalom by BAN Logic [4]

1. $A \rightarrow B: A, N_A$
2. $B \rightarrow S: B, N_B, \{A, N_A\}^{\leftrightarrow}K_{BS}$
3. $S \rightarrow A: N_B, \{B, K_{AB}, N_A\}^{\leftrightarrow}K_{AS}, \{A, K_{AB}, N_B\}^{\leftrightarrow}K_{BS}$
4. $A \rightarrow B: \{A, K_{AB}, N_B\}^{\leftrightarrow}K_{BS}, \{N_B\}^{\leftrightarrow}K_{AB}$

We can see that K_{AB} is encrypted together with N_B in message 3, and forwarded from A to B in message 4, in order to show the intuitive meaning of the freshness challenge by B, which is easier to understand than the original Yahalom protocol. But the challenge is that how to deal with the cases where the freshness is not obvious to track, such as the original version of Yahalom. In stead of change the protocol code for the analysis, we should change the analysis for the protocol code.

2) BAN logic dos not have a computation model to prove its completeness, in terms of power to check attacks, even

with those simple attackers. We consider BAN logic is not enough to track freshness. The following example, protocol 4, is the Needham-Schroeder protocol with shared key [22]. BAN logic needs to have the special assumption that B believes K_{AB} is fresh in order to do the reasoning. The authors claim that the reason for this assumption is that the designer of the protocol did not realize this assumption, and that has also been criticized by the timestamp paper [7]. However, the timestamp paper argues about this problem only in the case when K_{AB} is mysteriously stolen, and that is why timestamp is needed. Intuitively K_{AB} has to be fresh to B, because B can verify by the nonce N_B that A is active, and if A is active A must have verified that K_{AB} is fresh. The excuse in BAN logic paper is not a reasonable one. The real reason is that BAN logic does not capture the behavior of the freshness challenge that a term is verified fresh later than the time when the term is received. Tracking freshness is the key motivation for BAN logic, on the first page of the paper [4]: "…, and it is particularly necessary to take precautions against confusion caused by the reissue of old messages. ". But BAN logic is not powerful enough for freshness tracking.

Protocol 4:

1. $A \rightarrow S: A, B, N_A$
2. $S \rightarrow A: \{N_A, B, K_{AB}, \{K_{AB}, A\}^{\leftrightarrow}K_{BS}\}^{\leftrightarrow}K_{AS}$
3. $A \rightarrow B: \{K_{AB}, A\}^{\leftrightarrow}K_{BS}$
4. $B \rightarrow A: \{N_B\}^{\leftrightarrow}K_{AB}$
5. $A \rightarrow B: \{N_B - 1\}^{\leftrightarrow}K_{AB}$

We like the approaches of belief logics, which is based on intuition (intuition can lead to simple solutions), and they can describe naturally what happens in the real world. We tackle the first weakness of BAN logic by defining a logic directly based on the protocol code. We tackle the second weakness by integrating the strand model [32, 33, 34] in this logic, and we build a model called labeled strand based on it. Labeled strand works as the framework of this logic, and it supports the proof of soundness and completeness of this logic. BAN logic has the limitation that it assumes simple attacking scenarios, no message interleaving is allowed. This is similar to the scenario of OMA. We do not provide the ambitious solution in this paper to handle all possible tricky attacks, but concentrate on OMA which is common enough to be important and non-trivial enough to be interesting.

In the following sections, we will describe the behavior of running a protocol in the *real world*, and the logic rules in the *logic world*. The goal is to reflect the behavior of real world into the logic world correctly and sufficiently, which is proved by the soundness and completeness of this paper.

2. Modeling a Protocol Run (the real world)

2.1 Terms and messages.

A **term** is defined recursively

- A character string without internal structure is a term, which can be a principal name, atomic key, nonce, plain text, etc. (In this paper, the atomic public key, private key and shared key are represented like K_A, K_A^{-1}, K_{AB}.)
- [T1, T2, …] is a term, which is a finite list of terms, if T1, T2, … are all terms.
- If T is a term, then T^{-1} is a term, which is the reverse key if T is used as a key using the public key encryption algorithm. When T is considered as a public key, T^{-1} is the corresponding private key. Similarly, if T is the private key, then T^{-1} is the corresponding public key. We consider " $^{-1}$ "as a notation, not a structure. We assume T^{-1-1} is T, and a sequence of this tag is always reduced to its normal form.
- If X and Y are terms, $\{X\}^{\rightarrow}Y$ is a term, which is encryption of X using public key encryption with Y as the Key.
- If X and Y are terms, $\{X\}^{\leftrightarrow}Y$ is a term, which is the encryption of X using symmetric key encryption with Y as the key.

We assume **free term algebra**, which means that the only way to construct a term. Every term is reduced to its normal form (no repeated decryption and encryption). For example, the only way to achieve the bit string of an encryption $\{X\}^{\rightarrow}Y$ or $\{X\}^{\rightarrow}Y$ is by encrypting the same X with the same Y.

Composite key is allowed (which means a key can be any term). This will allow the frame work of this paper to deal with more general attacks. No matter the key is atomic or composite, the reasoning in this paper about OMA will be the same.

A **message** is a term. When a message is a list, usually the top level [ ] is omitted. Also $\{X, Y, \ldots\}^{\rightarrow}K_A$ means the same as $\{[X, Y, \ldots]\}^{\rightarrow}K_A$. Same for symmetric key encryption.

Subterms are defined in the obvious way:

- T and M are the subterms of [T, M]

- X and Y are the subterms of {X}Y
- X is a subterm of Y, if Y is a subterm of Z.
- X is a subterm of X.

A term is a **Session Dependent Component** (**SDC**) if its value can be different from session to session. Usually public keys and private keys are non SDCs, while nonces and plain text are SDCs.

2.2 Event and Strand

Strand [32, 33, 34] has been developed as a model to represent the exact behavior of a principal when running a protocol. Athena [28, 29] extended strand and defined semi-bundle which represents a possible run of the protocol. An **event** of a principal is either sending a message or receiving a message. A strand of a principal P is the sequence of events that P needs to do, which is defined by the protocol. For example, the strand of A in the Yahalom protocol is the following sequence, we use + to represent the sending event, and – for receiving event.

A's strand in Yahalom Protocol

1. + : A, N_A
3. – : {B, K_{AB}, N_A, N_B}$^{\leftrightarrow}K_{AS}$, {A, K_{AB}}$^{\leftrightarrow}K_{BS}$
4. + : {A, K_{AB}}$^{\leftrightarrow}K_{BS}$, {N_B}$^{\leftrightarrow}K_{AS}$

We say that sending and receiving events are **external events.** A principal can also have **internal events**, such as comparing and verifying the equivalence of different message components, constructing messages, decryption, and so on. The algorithm will represent the internal events naturally. In the following, an event means an external event.

We assume a principal will do internal events as much as possible after receiving each message and before sending every (already constructed) message.

A term X is **originated** by P in a message M if P sends out M and X does not occur as a subterm in any earlier messages received by P. It is possible that X is originated by P several times in different messages sent by P. This definition is different from the one defined [29] to achieve simplicity and efficiency in our algorithm. When a term is originated by P, P must know its subterms in order to construct it.

Before running a session, a principal P has its **initial knowledge**, which should include all of the terms that the principal need to know to construct or decrypt messages. It can include the public keys, its private keys, the symmetric keys shared with other principals, the principal names, the SDCs that are generated by P in the current session, and so on. P's initial knowledge is represented a list, called **initial (P)**.

2.3 Labeled Strand

Labeled strand is this paper's extension of the strand model. The extended features include the definition of terms (in the strand model, only atomic key is allowed), the explicit recording the locations of terms and the equivalence between terms, and the principals' behavior of **sensing** terms (newly defined), and **seeing** terms at different locations.

2.3.1 locations

Locations of terms are recorded using the following scheme:

- The i^{th} message in the protocol, $1 \leq i$, has location
- If term {X}Y has the location L, then X is located at L.α, and Y is located at L.β.
- If term [X, Y, Z] has location L, then X has location L.1, Y has location L.2, and Z has the location L.3. The lists that are not included in any encryption are fully expanded and then locations are assigned sequentially. For example, in [X, [Y, Z], {A}$^{\rightarrow}$B], the location for X is 1, Y is 2, Z is 3, and {A}$^{\rightarrow}$B is 4.
- We say a natural number and a α and a β that appears in a location has length 1. Similarly α.1 has length 2. Because of this way of evening the top level nested lists, the block with the locations 1.1.1 and 1.1 will be same. So we can say the location of a block will always have length 2. The empty location has length 0.

Locations can be used in reasoning conveniently:

- We use a lower case character as a variable to stand for a non-empty location with length 1. For example, i can be the value 1, 2, α or β. An upper case character stands for a location with a sequence bits, possible empty. For example if i.L strands for the location 1.2.α, then i has the value of 1 and L has the value of 2.α. If L is empty, L.H is the same as H.L and H..

- The term located at `L.H` is a subterm of the one located at `L`. For example, i.L can be any location of a term in the i^{th} message. We can use the notation of term(L) to represent the term located at the location L. To say a term is located in a block, we use the notation `term(i.j.L)`, where `i.j` is the location of the block
- Two different variables represent that they can represent possibly different values. E.g. `term(i.H) == term(i.L)`, H and L are possibly different, but they can also be the same (without the additional specification that $H \neq L$). But the two terms must be in the same message msg^i.

2.3.2 Equivalence

The **equivalences** between terms are recorded using the following scheme:

- If `X` is located at location L, `term(L)== X`.
- If `X` is located both at L and H, `term(L)==term(H)`.
- The equivalence relation is transitive, reflexive and symmetric.

2.3.3 Sensing and Seeing

The *Sensing* and *Seeing* behavior is defined as the following:

- A principal P can only sense and see the terms located within the message sent or received by P.
- A principal P senses a term X at the location L in message M, if P knows or conscious that X is used as the subterm at the position L to construct the containing message. In other word, when P does its internal computation, P knows that there exists a term (the term is unique, by the free algebra assumption), name it X (according to the protocol), which is used as subterm at location L when the message is constructed. Or we can say, P can sense the existence of X at L, and P can name it according to the protocol code.
- Seeing is defined similarly as defined in BAN logic. If a principal P can **see** a term, then P can repeat that term and use it to construct new message.
- Sensing covers the behavior of Seeing. In BAN logic only seeing is defined. But sensing may not be able to repeat the term. For example after P decrypt $\{X\}K_A^{-1}$ using the published public key of A, K_A, P can sense that K_A^{-1} is used as the key of the encryption, but P can not know exactly what is K_A^{-1}, in other word, can not see K_A^{-1}. If P can see a term then P can also sense the term. Sensing is powerful enough to reason about the equivalence and freshness of terms.
- The only difference between sensing and seeing (with the current definition of terms in this paper) is in the case of decrypting an encryption using public key encryption algorithm (when the decryption key is available). Seeing can only see the decrypted text, not the encryption key, while sensing can sense both the text and the encryption key.
- For the shared key decryption, both the text and encryption key (same as the decryption key, which is required for the decryption) can be seen and sensed.
- If P senses a fresh X at location L, can P can not see an old X at L.

2.3.4 Rules for Labeled Strand

There is a precise and automatic scheme to reasoning about the sensing and seeing behavior of a principal P when P can perform smoothly in the protocol. There are four intuitions behind.

First, if P *originates* a term (the term is never been received before), then P must construct the term using its top level subterms, and so P must (know and) see them. This is justified by the assumption of free encryption.

Second, P is only supposed to do decryption when P receives terms. So when P send out some (encrypted) term that P has received already, then whether P will see or sense the subterms at the place where they are sent depends on whether they can be seen or sensed where they are received.

Third, P may not see or sense a term in a message until several messages later, since P can receive the decryption keys later. That is why in the labeled strand rules the message numbers of the beliefs (where the belief occurs) can be different from the message number of the location of the seeing or sensing behavior (where the term involved occur).

Fourth, the most important difference between seeing and sensing is that when a term is decrypted, using public key encryption algorithm, P will only see the decrypted text, not the key, but P will sense both the text and the key.

The following rules form the scheme. The seeing behavior and sensing behavior are the same in some cases, we will use see/sense to represent these cases. The following rules are directly translated to labeled strand rules LS1 to LS8, respectively.

1.) If P can see a term at a location L, then P can also sense it at L.

2.) If P can see/sense a list, then P can see/sense its members.
3.) If P receives and sees an encryption using public key algorithm, and P can see the reverse key, then P can see the encrypted text.
4.) If P receives and sense an encryption using public key algorithm, and P can sense the reverse key, then P can sense both the encrypted text and the encryption key.
5.) If P receives and see/sense an encrypted term using symmetric key algorithm, and If P can see/sense the symmetric key, then P will see/sense both the encrypted text and key at the corresponding location.
6.) If P originates an encrypted term using public key encryption, then P will see/sense both the text and the key. This is justified by the first intuition.
7.) If P originates an encrypted term using symmetric key encryption, then P will see/sense both the text and the key, similar to 6.
8.) If P sends out a term that has already been received. Then if P see/sense a subterm of it where it is received, then P also see/sense that subterm at where it is sent.

LS9 will simply change the representation of the locations of terms in a more convenient way, described in section 2.3.2.

Labeled strand is the natural and detailed representation of a principal's requirement during a legal run of the protocol. Since it provides more information than strand, it can be a flexible and convenient reasoning framework to analyze general kinds of security protocol problems, not just OMA.

2.4 A Run of the Protocol and Its Properties

A principal P can *fail* at an event if one of the following conditions is true:

- P has not finished its earlier events successfully, or P did not start or finish its earlier events.
- If in the event P will send out a message and P can not construct the supposed message
- If in the event P will receive a message, and if X is a subterm of the received message, `P` can *see* two different values of X at different locations in the received and sent message so far. If the two different X located in the same received message, we call it *horizontal conflict*. If one X appears in a message earlier than the received message, we call it *vertical conflict*.

Notice that if the initial knowledge covers all terms P supposed to know before running a session, and previous receiving events is successful, P will always be able to construct a message and send it out, and P will not fail at the second condition.

A *run* represent a sequence of successful events of executing the protocol. It is defined as the following:

- If an event `e` is in the `run` and that `e` is a sending or receiving event by a regular principal `P`, then all P's previous events in the session must be also included in the run earlier than `e`.
- If an event `e` is in the `run` and `e` is done by a regular principal `P`, then `P` will not *fail* at `e`.

The idea of a run is similar to what defined as a semi-bundle in Athena [28]. The difference is that a semi-bundle will represent a set of runs, and each run is a possible linearization of the events of a set of strands.

Observation. When an honest principal P can see at location L a fresh subterm X of a block, then there must be sequence of fresh blocks to deliver the X from the originating place of X to P at L. Call this sequence of fresh blocks as the *carrier sequence* of `X` to `L`. In this sequence, G_0 sends `block(`i_1`,` j_1`)` to G_1, and G_1 send `block(`i_2`,` j_2`)` to G_2, and so on. G_0, G_1… are the regular principals who must be active in the current session for P to receive the fresh X. Especially G_0 is the originator of X. Call them the *`active principals`* with `P`'s point of view. If X is originated from P at L, then the length of the carrier sequence of X to P at L is 0. The length of a carrier sequence is the number of blocks contained in it.

When considering OMA, if a fresh block T is sent by an honest principal X and is aimed to another honest principal Y, Y may not receive the fresh T, since the attacker can intercept the fresh one and send an old one to Y. We are especially interested in the case when a term must be sent by an honest principal and also is received by the supposed receiver in the current session, with the point of view of a regular principal `P`. Call this kind of terms the *ensured term* for `P`, `ET(P)`, for what is sent, what is received. Any subterm of a fresh block in one of the carrier sequences to P must be a member of `ET(P)`. We can observe that if a block is received and the block is fresh, then the block is an ensured term.

2.5 Causally Earlier Conditions

There are necessary conditions to let P to finish its role in the protocol. The following are the intuitive description:

1.) If that the term X located at i.j.L must be fresh and the term is an ensured term, is the necessary condition for P to finish its role, then that all of the blocks included in any possible carrier sequence to deliver X to i.j.L are fresh and ensured, must also be the necessary condition for P to finish its role, and all the principals who is the sender or the receiver of these fresh blocks must be active in the current session.

This can be proved by contradiction. Suppose one of the blocks described above is not fresh, then we prove by induction on the distance, in terms of number of blocks, between the block to i.j.L, following the carrier sequence, and then we can see that it is impossible for P to receive the fresh X at i.j.L.

2.) If P needs to have a principal Q to be active at message i, which means Q is either the sender or the receiver of message i, then all of the necessary condition for Q about which block must fresh and ensured (together with the knowledge of the subterms located in these blocks) and which principal must be active at which message, must also be necessary condition for P.

3.) If P needs to have a set of sequence of fresh blocks to be successfully delivered (ensured), and these blocks can form a carrier sequence of a SDC X from the originating place of X to i.j.L, then the ensured term located at i.j.L must be fresh (the fresh X must be passed to P at i.j.L.

The events of delivering those fresh blocks (described by the above necessary conditions), has the **causally precedence** relationship [34] with the last event in P's strand. And all the necessary conditions gathered can be considered causally earlier to the last event of P. The gathered knowledge is similar to the concept of goal binding and searching back defined in Athena [28]. The difference is that we are only interested in finding out the necessary conditions, not all of possible situations like the goal binding in Athena.

Originally P only knows that the SDC terms originated by P are fresh. To find the necessary conditions 1 and 2 is like *searching back* toward the origination place of a SDC, while for condition 3, it is like *searching forward* from the originating place of a SDC. The algorithm and the rules, which are defined later, will have this kind of two direction searching ability to find all necessary conditions

3. Notations and Meanings

Table 1: Notations used in Rules

P $\models_m$ facts	After P sends or receives the m^{th} message, P believes the facts. The facts in this notation means they are the knowledge only available to P, not shared. The set of facts believed by a single principal P can be grouped as {P $\models_m$ fact1, fact2, … } for clarity. Here m means the number of the message where P is involved. Variables inside has default quantifier of ∀.
P $\not\models_m$ fact	P cannot believe the fact, after trying all allowed rules at message m. We do not provide negation operators in the logic. Variables inside has default quantifier of ∀
,	Logical and, to separate the beliefs of a principal. It should be distinguished from the separator in a list, by the context.
;	Logical and, used to separate different kind of conditions, or different principals beliefs.
LHS ➔ RHS	A logic postulate. The LHS is the conditions. A condition can have three forms: 1.) Principal beliefs, in the form of P $\models_m$ facts; 2.) Shared knowledge, presented as facts without the preceding P $\models_m$. These facts includes who is the supposed sender or receiver, what is the supposed term located at the location L.; 3.) natural facts, like i ≤ j, P ≠ Q. Beliefs of different principals and different kind of conditions are separated by " ;", and it means the logic and of them.
term(L)	The term at location L.
< /@$_L$	See and sense can by applied exactly the same way, the action is applied to the term located at the location L.

P $<_L$ T	P see the term T at location L in the current session.
P $@_L$ T	P sense the term T at location L in the current session.
#Block(L)	This notation will make the belief that the block is ensured stand out
msg^i	The i^{th} message in the protocol (the code)
P < term(L)	P can see the term at location L in the current session.
P @ term(L)	P can sense the term at location L in the current session.
P $\models_i$ *fact	The fact is the causally earlier condition for P to be active at message i. The fact will also be causally earlier to other principals if they depend on P to be active at message i.
P < #X	P can see a fresh X, which is a SDC, in the current session. This notation is goal of the algorithm, and is generated by the conclusion rule.
$P^{active}{}_i$	P is active at the i^{th} message.
P $\rightarrow_i$	P is the supposed sender of the i^{th} message, i ≥ 1.
P $\leftarrow_i$	P is the supposed receiver of the i^{th} message, i ≥ 1.
…	The remaining list items. It also means the same thing can applied to the remaining list items.
EndP	The message number where P ends its role in the protocol

4.The OMA Checker and the Rules (the logic world)

The input of OMA checker is just the protocol code. The output will be the checking results of whether some OMA attack is possible to a principal. If there can be an attack, the details of that attack can be showed by constructing the evidence sequence.

The OMA checker has two stages, preparation stage, and reasoning stage.

4.1 Preparation stage.

The preparation stage will built the shared knowledge about the protocol and the initial knowledge of each principal, without applying any rules:

Preparation Steps:

1.) Add the following facts into SharedKnowledge**:**

- For every message, say message i, if S is the sender and R is receiver, add S $\rightarrow_i$ and R $\leftarrow_i$ into SharedKnowledge.
- For every term T appears in the protocol, if T appears at location L, add term(L) == T to SharedKnowledge.

2.) Then the following tasks will be done for every principal P.

- Add P $\models_0$ P $<_0$ inital(P) into knowledge(P*)*
- Add P $\models_0$ #X into knowledge(P) for every atomic SDC *X* that will be originated from P.
- If P is the sender or the of message i, add P $\models_i$ P $<_i$ msg^i into knowledge(P).

4.2 Reasoning Stage

The purpose of the reasoning stage is simply to compute the closure of all possible rules for all principals.

To make their meaning of the rules more clear, we separated them into default rules, labeled strand rules, and reasoning rules, showed in table 2, 3, and 4 respectively.

Default rules will be applied together with labeled strand rules and reasoning rules. E1 and E2 describe the symmetric and transitive properties of equivalence. X, Y and Z are variables stands for a regular term or a notation like term(L). The common reflexive property is removed since it is not needed and may make the OMA checker inefficient. The monotonic rule M1 shows that the knowledge of P is monotonically increasing.

The labeled strand rules (explained in section 2.3) is the direct implementation of labeled strand. Theses rule will not interfere with the reasoning rules. So one choice (not chosen here) of the reasoning stage is to divide it with two steps, step 1 to compute the closure of labeled strand rules, and step 2 for the closure of reasoning rules.

Reasoning Stage Steps:

- Compute the closure of all of the rules (A possible way to compute the closure is showed in the appendix 2). The beliefs of each principal P is saved in knowledge(P).

There are possible optimizations of the computation, showed in appendix 3.

Table 2. Default rules.

$P \models_m X == Y$ $\Rightarrow P \models_m Y == X$	(E1)
$P \models_m X == Y,\ Y == Z$ $\Rightarrow P \models_n X == Z$	(E2)
$P \models_m FACTS$; $m < n$ $\Rightarrow P \models_n FACTS$	(M1)

Table 3: Labeled Strand Rules

$P \models_m P <_L X$ $\Rightarrow P \models_m P @_L X$	(LS1)
$P \models_m P < /@_L [X, Y]$ $\Rightarrow P \models_m P < /@_{L.1} X,\ P < /@_{L.2} Y$	(LS2)
$\{ P \models_m P <_{n.L} \{X\}^{\rightarrow}Y,\ P <_H Y^{-1}\}$ $P \leftarrow_n$ $\Rightarrow P \models_m P <_{n.L.\alpha} X$	(LS3)
$\{ P \models_m P @_{n.L} \{X\}^{\rightarrow}Y,\ P @_H Y^{-1}\}$ $P \leftarrow_n$ $\Rightarrow P \models_m P @_{n.L.\alpha} X,\ P @_{n.L.\beta} Y$	(LS4)
$\{ P \models_m P < /@_{n.L} \{X\}^{\leftrightarrow}Y,\ P @_H Y\}$ $P \leftarrow_n$ $\Rightarrow P \models_m P < /@_{n.L.\alpha} X,\ P < /@_{n.L.\beta} Y$	(LS5)
$\{ P \models_m P < /@_{n.L} \{X\}^{\rightarrow}Y\}$; $\{ P \not\models_m P < /@_{f.H} \{X\}^{\rightarrow}Y \}$; $P \rightarrow_n,\ P \leftarrow_f$ $\Rightarrow P \models_m P < /@_{n.L.\alpha} X,\ P < /@_{n.L.\beta} Y$	(LS6)
$\{ P \models_m P < /@_{n.L} \{X\}^{\leftrightarrow}Y\}$; $\{ P \not\models_m P < /@_{f.H} \{X\}^{\leftrightarrow}Y \}$; $P \rightarrow_n,\ P \leftarrow_f$ $\Rightarrow P \models_m P < /@_{n.L.\alpha} X,\ P < /@_{n.L.\beta} Y$	(LS7)
$\{P \models_m P < /@_{n.L} X,\ P < /@_{n.L.T} Y,\ P < /@_{f.H} X\}$; $P \rightarrow_f$ $\Rightarrow P \models_m P < /@_{f.H.T} Y$	(LS8)
$P \models_m P < /@_L X$ $\Rightarrow P \models_m P < /@\ term(L),\ term(L) == X$	(LS9)

4.3 The Reasoning Rules

The reasoning rules will do the reasoning about the freshness of terms. The default rules will be applied together with the reasoning rules.

FT1 (fresh term) says that if a subterm of a block is fresh, then the containing term, which is a subterm of the block too, is also fresh. FT2 is obvious, the X and Y can be a regular term, or in the form of term(L). ET1, ET2 and ET3 (equivalent terms) say if two terms are equivalent, their corresponding subterms are also equivalent. A1 (Active Principal) shows that the sender of a fresh term must be active. A2 means that if a principal Q can sense a term (which

should be part of a fresh block), which implies that Q is either the sender or the receiver, then Q is active at that message. The learning rules from LN1 to LN6 will gather other principal's knowledge into P's knowledge. LN1 will let P learn whether Q can sense a smaller term, if P knows Q can sense a bigger term. LN2 is designed for the reason that a block is always sensible to Q, if Q can sense the smaller term. Combining LN1 and LN2, P can learn all of the subterms of a block that is sensible to Q. LN3 says if Q is active at message a, and P knows Q can sense a term, and Q at message a believes that the term is fresh, then P will learn that the term is fresh. LN4 is describing the searching back behavior. If P knows Q is active at message a, and a term T1 is fresh, Q can sense T1, and Q believes that Q has received another term T2 that is equivalent to T2, then P will learn that Q can sense T2 and T1 and T2 is equivalent. LN5 says that if P knows Q is active at message a, P knows that Q can sense two terms, and Q at message a believes that the two terms are equivalent, P will learn this equivalence. The causal earlier rules CE1 to CE5 will describe all the causally earlier conditions so they can be transferred to other dependent principals. CE1 to CE5 together with LN6 can be explained as one sentence: if P believes that Q is active at message a, then P will learn all of Q's belief at message a (as P's causally earlier conditions) about who is active, and the beliefs such that all of terms in them are ensured (located in some fresh block). The conclusion rule (C1) will make the final result stand out.

Table 4: Reasoning Stage Rules

$P \models_m$ #term(i.n.L.H) ➔ $P \models_m$ #term(i.n.L)	(FT1)
$P \models_m$ #X, X == Y ➔ $P \models_m$ #Y	(FT2)
$P \models_m$ term(L) == term(H); term(L) == [X, Y,…] ➔ $P \models_m$ term(L.1) == term(H.1), term(L.2) == term(H.2), …	(ET1)
$P \models_m$ term(L) == term(H) ; term(L) == {X}$^{\rightarrow}$Y ➔ $P \models_m$ term(L.α) == term(H.α), term(L.β) == term(H.β)	(ET2)
$P \models_m$ term(L) == term(H) ; term(L) == {X}$^{\leftrightarrow}$Y ➔ $P \models_m$ term(L.α) == term(H.α), term(L.β) == term(H.β)	(ET3)
{ $P \models_m$ #term(f.h), R @ term(f.h) }; { Q $\rightarrow_f$, R $\leftarrow_f$ }; P ≠ Q ➔ $P \models_m$ Q @ term(f.h), #block(f.h)	(A1)
$P \models_m$ Q @ term(f.h.L); P ≠ Q ➔ $P \models_m$ Q$^{active}{}_f$	(A2)
{ $P \models_m$ Q @ term(f.h), Q$^{active}{}_a$ } ; { $Q \models_a$ Q @ term(f.h.L)}; P ≠ Q ➔ $P \models_m$ Q @ term(f.h.L)	(LN1)
{ $P \models_m$ Q @ term(f.h.L) }; P ≠ Q ➔ $P \models_m$ Q @ term(f.h)	(LN2)
{$P \models_m$ Q @ term(f.h.L),Q$^{active}{}_a$ }; {$Q \models_a$ #term(f.h.L) }; P ≠ Q ➔ $P \models_m$ #term(f.h.L)	(LN3)
{$P \models_m$ #term(f.h.L), Q @ term(f.h.L), Q$^{active}{}_a$ } ; {$Q \models_m$ Q @ term(i.j.N), term(f.h.L) == term(i.j.N)} ; Q $\leftarrow_i$; P ≠ Q ➔ $P \models_m$ Q @ term(i.j.N), term(f.h.L) == term(i.j.N)	(LN4)

{ P $\models_m$ Q @ term(L), Q @ term(H), $Q^{active}{}_a$} ; { Q $\models_a$ term(L) == term(H) } ; P ≠ Q ➔ P $\models_m$ term(L) == term(H)	(LN5)
P $\models_m$ #block(f.h) ➔ P $\models_m$ *#block(f.h)	(CE1)
P $\models_m$ $Q^{active}{}_a$ ➔ P $\models_m$ *$Q^{active}{}_a$	(CE2)
P $\models_m$ #term(f.h.L), #block(f.h) ➔ P $\models_m$ *#term(f.h.L)	(CE3)
P $\models_m$ term(f.h.L) == term(a.b.N), #block(f.h), #block(a.b) ➔ P $\models_m$ *term(f.h.L) == term(a.b.N)	(CE4)
P $\models_m$ Q @ term(f.h.L), #block(f.h) ➔ P $\models_m$ *Q @ term(f.h.L)	(CE5)
{ P $\|=_m$ $Q^{active}{}_a$ } ; { Q $\|=_a$ *FACT } ; P ≠ Q ➔ P $\|=_m$ FACT, *FACT	(LN6)
P $\models_m$ P < term(L), #term(L), term(L) == X ➔ P $\models_m$ P < #X	(C1)

4.3 Result of OMA checker

After the running reasoning stage, if for a principal P, if P < #X ∉ knowledge(P), for a atomic SDC X, and P is supposed to see X, which means P < term(L) and term(L) == X □ knowledge(P), then there is an OMA attack to trick P to accept some old X. If P < #X □ knowledge(P), then an OMA to trick P to accept some old X is impossible.

If an attack is detected for P, then an *evidence sequence* can be constructed (described in the following section) to show the details of the attack.

4.4 Evidence Sequence

The evidence sequence is constructed to show the attack in detail, if an OMA is detected by the OMA checker. The design of constructing the evidence sequence is inspired by the observation that is proved by lemma 1. Lemma 1 can also be proven by the completeness of OMA checker. The following will give a more direct proof.

Lemma1:
If there is an OMA, call it attack1 for a protocol to let a principal P to accept an old value for a SDC X, then there is another OMA attack2 for the same protocol such that every faked message constructed by the attacker in attack2 is the same as the corresponding faked messaged message in attack1, except that all of the old blocks that appear in a faked message in attack1 (they may be chosen from different old sessions) are replaced with the corresponding old blocks coming from a single old session (with the same locations).

Proof: Suppose there is a run, called V1, which is a sequence of events. In V1 some message is faked by the attacker and contains some old blocks. We will do the old block replacing process message by message in V1, and the result sequence is called V2. We will prove that at each step the so far achieved prefix of V2 is a run. OldS is the full sequence (no dropped message) of events in an old session that is perfectly legal and has no attacks (OldS is also a run) from which the old blocks are chosen to replace the old blocks in V1.

Since sending events will always be successful if the previous events are all successful, we only need to show that the events of receiving the replaced message will not fail. Suppose the messages contained in V1 are msg^{m1}, msg^{m2}…, we do induction on msg^{mj}, **1 ≤ j**, until the end of V1. We call the receiver of msg^{mj} as Rmj.

Base case: msg^{m1}. We replace all of the old blocks in msg^{m1} with the corresponding blocks in the first message in OldS. There is no vertical conflict (two different values of a SDC X in two different messages are visible to the principal) in the replaced message since it is the first event of Rm1, otherwise V1 will not be a run. Suppose Rm1 find a horizontal conflict (two different value of X in the same message) of an atomic SDC X, in the replaced message, which means Rm1 can see two different X at the replaced msg^{m1}.

Case 1: the two X appears in two new blocks. It is impossible since if it is true, the same conflict appears also in the first message in V1 and thus V1 will not be a run.

Case 2: the two X appears in two old blocks in the new message, which means that the two old blocks come from OldS. This is impossible because it means that the same conflict occurred in OldS. But in OldS there is no conflict, contradictory.

Case 3: the two Xs appears in a single block. If the block is a new block, then V1 must have the same conflict, impossible. If the block is an old block, which comes from OldS, then the OldS must have the same conflict, impossible.

Case 4: one of the X locates at an old block, while the other locates at a new block. There can only be two values of X in the new message, the old one from OldS, and the new one created by the sender of msg^{m1}. Since in an old block everything is old, the new X must be located in the new block, and the old X is located at the old block. Suppose the old X appears at location L in the old block from OldS in V2, then Q must see an old X at location L in V1 also. Since Q will see the new X in V2, say at L2, all of the blocks in a possible carrier sequence of X to L2 must be fresh and included in V2. Then they are included in V1 too. So Q will also see a new X in V1 (new block has not been replaced, and the carrier sequence of X), there is a conflict in V1 too, impossible.

Induction case: Suppose the old block replacing process is successful at all of the messages before msg^{mj}, $1 < j \le n$, which means there are no vertical and horizontal conflict up to the msg^{j-1}, we want to prove that after replacing the old blocks in the msg^{mj}, the result is also a run. We prove by contradiction and suppose there is a conflict for receiver of msg^{jm}.

If the conflict is a horizontal conflict, it must occur at msg^{jm} since by the induction assumption it can not occur earlier. It is impossible by the identical proof of the base case.

Now the only possible conflict is a vertical conflict. Suppose Rmj sees an X at msg^{mj} and a different X at msg^{mi}, $1 \le i < j$. We can repeat the proofs of the case 1, 2 and 4 in the base case here to show that it is impossible. The reasoning of the case 1, 2 and 4 does not require it to be a horizontal conflict or a vertical one.

Now we have proved that V2 is a run. We only need to show if V1 is an OMA to trick P to see an old SDC X at location L, then in V2 is also an OMA to trick P to see some old X at L. P should not be the originator of X, otherwise P will always see the fresh X in every possible run. So P must firstly see X by receiving it. It is enough to just prove the case that P receives and sees X at L.

Suppose in V2, P will receive and see a fresh X. Then every possible block that is located in some carrier sequence of X to L must also be included as fresh blocks in V2. By the way V2 is constructed, all these blocks must also included in V1 as fresh blocks. So in V1 P will also see a fresh X at L. Contradictory.

4.4.1 Building Evidence Sequence

The idea to build evidence sequence is to include all of the events of an active principle, while replaces all blocks that P does not believe to be fresh with the corresponding old blocks from an old session.

Preparation step :

Define the following names for the principal P:

- P ends at the $endP^{th}$ message, $1 \le endP$.
- *NewBlocks*(P) = { block(i.j) | $P \models_{endP}$ #block(i.j) ∈ knowledge(P), for some i, j}.
- NewMessages(P) = { i | $P \models_{endP}$ #block(i.j) ∈ knowledge(P), for some i, j }.
- ActivePrincipals(P) = { Q | $P \models_{endP} Q^{active}_{i}$ ∈ knowledge(P), for some j } ∪ { P }.
- LatestAction(P,Q) = max({ m | $P \models_{endP} Q^{active}_{m}$ }). LatestAction(P, P) = endP.
- EV(P) is a sequence of events, initially empty.
- OldS is an old session.

Scanning step:

Starting from the first message to the $endP^{th}$ message, for each message, say the j^{th} message, $1 \le j \le endP$, suppose *Sj* is the sender and *Rj* is the receiver, do the following in sequence:

– If $Sj \in ActivePrincipals(P)$, and $j \le LatestAction(P, Sj)$, append the event to $EV(P)$: Sj constructs msg^j using its current knowledge and sends it out.
– If $Rj \in ActivePrincipals(P)$, and $j \le LatestAction(P, Rj)$, then
  - If $j \notin NewMessages(P)$, append the event to $EV(P)$: Rj receives the old j^{th} message in $OldS$ (it is sent by the attacker)
  - If $j \in NewMessage(P)$, append the event to $EV(P)$: Rj receives a faked message (constructed by the attacker), which is the same as the new msg^j sent by Sj (it must be true that $Sj \in ActivePrincipals(P)$), except that all of the blocks of the new msg^j that are not in $NewBlocks(P)$ are replaced by the corresponding old blocks in the old msg^m in $OldS$.
  – If any of the above condition is not true, do nothing.

Observation: When run the protocol with the evidence sequence, for a SDC X, X can only have two possible values, the new one generated by the supposed originator of X, and the old one chosen from a single old session.

5. Soundness

The following are obvious observations about running the OMA checker with a protocol. P is a principal.

1.) P can sense a term X after sending or receiving the j^{th} message, X has the location L, when running the protocol, iff $P \models_j P$ @ term(L) ∈ knowledge(P).
2.) P is the sender of the i^{th} message, iff $P \leftarrow_i \in$ SharedKnowledge. Similarly, P is the receiver of the i^{th} message, iff $P \rightarrow_i \in$ SharedKnowledge. A term X appear is used to construct the i^{th} message as a subterm in the location i.L, iff X == term(i.L) ∈ sharedKnowledge.
3.) After sending or receiving the j^{th} message, P can sense the term X at two different locations L and H, $P \models_j$ term(L) == term(H).
4.) If an atomic SDC X is originated from P, then for every location where P originates X and send X out, say location i.j.L, $P \models_i$ #term(i.j.L), P @ term(i.j.L).
5.) Among the leaning rules LN1 to LN6, only LN2 does not require a condition of Q^{active}_j, for some Q and j. But the condition of Q @ term(i.j.L) in LN2 immediately implies that Q^{active}_i, by A2.
6.) Whenever $P \models_m$ #block(i. j), it must also be true that $P \models_m$ #term(i. j), Si^{active}_i, Ri^{active}_i, Si @ term(i, j), Ri @ term(i, j). The reason is that only A1 and LN6 will produce $P \models_m$ #block(i. j). For A1, from the conditions and conclusions we know P have 3 of these 5 beliefs. The remaining 2 will come from applying A2 with the conditions of Si @ term(i, j), Ri @ term(i, j). LN6 will preserve this invariant, once this invariant is true.
7.) A1 must be the rule to apply in order to let P have the first belief of $P \models_m$ W @ term(L), because all other possible rules will depend directly or indirectly on the facts that Q @ term(H) and $P \neq Q$, for some Q and H. In A1, there is the condition of R @ term(m.n), R can be P. A1 reflects the behavior of the freshness challenge that P generates, sends out, and receives some fresh term. After receiving the fresh term, P knows that the sender must be active.
8.) For all of the learning rules, together with A1 and A2, if a term appears in the conditions or conclusions, say term(i.j.L), then this term is an *ensured* term for P, in the sense that the term is sent and received in the real world by the honest principals in the current session. In fact, P must have $P \models_m$ #block(i.j) for some m. This can be proved by induction. And for each rule, suppose this observation is true for the conditions, then it is also true for the conclusion.
9.) The only terms that P may believe to be fresh, but actually not ensured are those that are sent out by P. But these terms will not participate in the learning rules.
10.) If $P \models_m Q^{active}_i$, then in the real world, there must be a sequence of fresh blocks passing Q at the j^{th} message and reach to P no later than the m^{th} message. If we draw the branches of fresh blocks as a upside down tree, The node of P's strand at the m^{th} message is the root, than the node of Q at the j^{th} message is on a branch of the tree.
11.) Only sense is used in reasoning the causally earlier conditions.
12.) The beliefs have the temporal feature. The belief of $P \models_i$ fact will honest reflect P's behavior in the real world at the i^{th} message.
13.) In summary, the labeled strand rules will reflect the labeled strand correctly. The knowledge that P collects by using the reasoning rules about other strands and other principals must be causally earlier conditions about tracing the freshness, and P needs them to finish its strand.

Soundness Theorem: After running the OMA checker. For a principal P and an atomic SDC X, if $P \models_{endP} P < \#X$, then there must be no old message attack to trick P to accept some old value of X.

Proof: Intuitively soundness means that algorithm will only derive the necessary conditions for P to finish its strand successfully. The soundness is obvious. Detailed proof can show that each rule is sound. The observations also shows the behind reason of the design of the rules and its soundness. The labeled strand rules are sound, since it is just exact translation of the labeled strand. The reasoning rules are sound since they will only derive the causally earlier conditions for P with no confusion (the terms are all ensured). Since each belief will reflect the corresponding real world behavior correctly, no OMA of an atomic SDC X to P is possible, if $P \models_{endP} P < \#X \in$ knowledge(P).

6. Termination and Computation Time

The algorithm will terminate since the amount of beliefs is finite and the algorithm will not continue to compute new beliefs (the details in appendix 2).

The locations of beliefs like $P \models_i$ can be organized as a list of sets. And the monotonic rule can be implemented using a sequence of belief, without duplicating the same beliefs at different steps. The equivalence beliefs can be implemented using equivalence classes. A term can only belong to one equivalent class.

Suppose the length of the protocol is *L*, in terms of number of characters in the code. The total number of terms is $O(L)$ (easy to prove). The number of principals is *P*.

For each term say term(T), a principal Q will have the beliefs of #term(T), term(T) == X, W @ term(T), term(T) == term(H). The number of these beliefs are bounded by the belief number of W @ term(T) which is $O(P)$. Other beliefs like #(block(i.j), *fact do not need to mention since they have lower bound of belief numbers. The total number of beliefs for each principal is $O(P \times L)$. The total number of beliefs for all principals is *B*, $B = O(P^2 \times L)$.

The maximum number of independent conditions (the conditions the can decide other input conditions, defined in appendix 2) of all rules is 2. By the Rule Closure Complexity Theorem, the overall complexity is $O(R \times B^{2+1}) = O(R \times P^6 \times L^3)$.

This is the most pessimistic bound, but it is a polynomial time result. In computing the closure of OMA checker, many rules can be applied sequentially together, For example, A2 can always be applied after A1. And the real computation cost will be much lower than this bound.

There are potentials to make the computation much more efficient (see the appendix).

The framework provided in this paper provides several features supporting efficient implementation:

1.) The learning rules can be utilized to remove redundant computation as much as possible.

2.) The algorithm will compute all of the possible OMA for all principals with all SDC in just one run, and which is fully automatic.

3.) Third, the logic will only reason about the correct and necessary conditions, without wasting time on guessing and computing the possible but false cases, a feature of many other tools.

We consider our approach, with these features, a novel and practical solution. These novel features make the extension of this paper promising and interesting to deal with the very hard problem of checking the most general kinds of attacks.

Rule Closure Complexity Lemma

Rules is a set of rules, the total amount of rules is *R*. *Facts* is a set of facts, initially is not empty, call the initial facts as *Initial Facts*. Each rule $r_i \in$ Rules needs Num(r_i) facts as the input, and returns a set of facts. $1 \le i \le R$. Suppose the computation time to compute the output of every given signature is a constant. $A = \max\{ \text{Num}(r_i) \mid r_i \in R \}$. When a new fact is generated, the new fact will be used an input fact of the rules to generate further pieces of knowledge. When Facts is expanded to the extent that no more new facts can be added, we say the closure of Rules with Initial Facts is computed. There is an algorithm to compute this closure with the complexity of $O(R \times B^{A+1})$.

Proof: Facts and beliefs have the same meaning in this lemma.

We define a signature as a tuple of the form $[r_i, K_1, K_2, \ldots K_n]$, where $r_i \in$ Rules, and $K_1 \ldots K_n \in$ Facts. P(A, B) is the number of possible sequences using A elements from the total set of B elements. The total number of signatures is $O(R \times P(A, B))$. Since P(A, B) is $O(B^A)$, the total amount of signature is $O(R \times B^A)$.

Note that possibly different signatures will produce the same knowledge several times. For checking a signature is a new one, we can sort the facts set, and then each check will cost $O(log(B))$. But it will not change the polynomial time result. So this proof assume the simpler case where the check cost $O(B)$

There is an algorithm which can enumerate the entire possible signatures on the fly (together with the process of generating new facts), and each signature can be touched only once, as the following:

NewFacts is an empty sequence.

Initial step: enumerate and compute all of signatures using only facts from Initial Facts. Every time a new knowledge is generated (not a member of current Facts), append it to NewFacts.

Iteration Step:

- pop off the first element from NewFacts, call it *a*
- append *a* into Facts.
- Enumerate and compute all of the signatures where *a* is used as an input.
- If a fact *b* is generated, and *b* is a new fact ($b \notin$ Facts, and, $b \notin$ NewFacts) append *b* to NewFacts..
- Repeat Step 2 until NewFacts has to be empty. Then Facts is the computed closure.

So the complexity is $O(R \times B^A \times B) = O(R \times B^{A+1})$.

For the input facts of a signature (a permutation of beliefs), some facts can decide other beliefs, call them *independent facts*. For every r_j, we define the set number of independent facts as $I(r_i)$. Define $I = \max\{I(r_j) \mid r_j \in \text{Rules}\}$. The possible number of signature is $O(R \times B^I)$. This observation proves the following theorem.

Rule Closure Complexity Theorem:
There is an algorithm to compute the closure of the rules with complexity $O(R \times B^{I+1})$.

7. Completeness

Completeness theorem: For a protocol Pro, an atomic SDC X, and a principal P, if $P \models_{endP} P <$ #X $\notin$ knowledge(P), and P is supposed to see X ($P \models_{endP} P <$ term(L) $\in$ knowledge(P) , and term(L) == X) then there is an OMA that will make P to accept an old value of X.

The completeness theorem can be proved by showing that there is a run of the protocol, which is the evidence sequence.

Intuitively completeness means that rules are powerful enough to collect all of the necessary conditions for P to finish its strand. Lemma 2 essentially shows that the rules can search forward, in the sense that after a sequence of fresh blocks is found, P will discover the new knowledge about equivalence and freshness of the terms inside these blocks, and the searching direction will start from the originators, following the direction of the deliverance of the blocks. Lemma 3 essentially shows that the rules can search backwards, in the sense that if P senses that some term X is fresh at some location L, then P will find all of the possible carrier sequence of X to L, and knows that all of the blocks along those sequences are fresh and all of the principals involved are active, and the searching direction starts from L and will trace back toward the originator of X. Lemma 2 and 3 together will show that some properties must be true in the real world for P iff they are reflected in the logic world as P's knowledge. Lemma 4 will show EV(P) is a run. Then the proof of the completeness theorem is easily followed. When applying rules in the proofs some conditions are not mentioned if they are obvious. When a belief reflects the labeled strand, i.e., the rule is derived by the labeled strand rules and default rules, we may not present the rules to derive it since they are so obvious.
e, such that after the run, P will actually receive an old X.

Lemma 2 In EV(P), if a principal Q (Q can be P) receives and senses a fresh X, X is an atomic SDC, at position i.j.L, then $P \models_{endP}$ #term(i.j.L) , Q @ term(i.j.L) $\in$ knowledge(P), and $Q \leftarrow_i \in$ SharedKnowledge.

Proof: The conditions of this lemma are real world behaviors (no events in EV(P) will fail before Q receives X), and the conclusions is the logic world behavior. The proof of this lemma can demonstrate the searching forward ability of the rules.

Since Q receives the fresh X at position i.j.L, there must be a carrier sequence of X to i.j.L in EV(P), call it CSX, so that every block in CSX is included in EV(P) as a fresh block. We name the sender of the first block in CSX as G0, the

first block as term(i1.j1), similarly the m^{th} block as term(im, jm), and so on. Q is the same as Gn, suppose the length of CSX is n, $n \geq 1$. If $P \models_{endP}$ FACT $\in$ knowledge(P) , we say P has FACT.

$Q \leftarrow_i \in$ SharedKnowledge is obvious since Q is the receiver of the i^{th} message, by observation 2 of soundness.

Gm-1 and Gm are the sender and receiver of the m^{th} block, which is the same as term(im.jm). By the definition of EV(P), P has #block(im.jm). P will also have Gm-1 @ term(im.jm), Gm @ term(im.jm), #term(im.jm), $Gm\text{-}1^{active}{}_{im}$, and $Gm^{active}{}_{im}$. Call it ***result 1***. This is showed by observation 6 of soundness.

We should take special care to the cases where a principal R can not sense the passing X (but it can sense some term containing X) and R forwards a bigger term containing X, in the middle of CSX. Suppose Gm is one of the principles in the middle of CSX. Then there must be a smallest term t^{Gm} that t^{Gm} contains the passing X and Gm can sense t^{Gm} in both the received m-1^{th} block, say as term(im-1.jm-1.T^{Gm}m-1), and the sent m^{th} block, say as term(im.jm.T^{Gm}m). Then it must be true that Gm $\models_m$ term(im-1.jm-1.T^{Gm}m-1) == term(im.jm.T^{Gm}m), by the soundness of the rules to record the labeled strand. Let the passing X received by Gm and contained in t^{Gm} as term(im-1.jm-1.T^{Gm}m-1.L^{Gm}m-1), and it sent by Gm in the m+1^{th} block in CSX as term(im.jm.T^{Gm}m.L^{Gm}m). Note in the case when Gm can sense the passing X, X is t^{Gm}, and $T^{Gm}{}_m$ and $T^{Gm}{}_{m-1}$ are empty.

By result 1, P has G_m @ term(im-1.jm-1) and Gm @ term(im.jm), then by LN1, $P \models_{endP}$ Gm @ term(im-1.jm-1.T^{Gm}m-1), G_m@ term(im.jm.T^{Gm}m). By LN5, $P \models_{endP}$ term(im-1.jm-1.T^{Gm}m-1) == term(im.jm.T^{Gm}m). Call it ***result 2***. If P is the same as Gm, then the above result is trivially true without needing to apply LN1 and LN5.

Now we do induction to show that P will believe for each location H that X appears in CSX, P has #term(H).

***Step 1 (base step)*:** G0 must have a belief of G0 $\models_{i1}$ #term(i1.j1.T^{G0}1.L^{G0}1), G0 @ term(i1.j1.T^{G0}1.L^{G0}1). X is originated at i1.j1.T^{G0}1.L^{G0}1. G0 is the originator of X (actually T^{G0}1 is empty). This is true by the soundness of the algorithm to record the labeled strand into beliefs. If P = G0, then $P \models_{i1}$ #term(i1.j1.T^{G0}1.L^{G0}1) and also $P \models_{endP}$ #term(i1.j1.T^{G0}1.L^{G0}1) by M1. If $P \neq G0$, then since P has G0 @ term(i1.j1), P will have G0 @ term(i1.j1.T^{G0}1.L^{G0}1), by LN1. Then by LN3, P has #term(i1.j1.T^{G0}1.L^{G0}1).

***Step h (induction step)*:** Suppose for every previous principal in CSX, say Gm, $0 \leq m < h$, if Gm sends out a block containing the passing X as the term(im+1.jm+1.T^{Gm}m+1.L^{Gm}m+1), then P has #term(im+1.jm+1.T^{Gm}m+1.L^{Gm}m+1). It is obvious that P also has #term(im+1.jm+1.T^{Gm}m+1). We want to prove that if G_h sends out the h+1^{th} block in CSX containing the fresh X as term(ih+1.jh+1.T^{Gh}h+1.L^{Gh}h+1), P will have #term(ih+1.jh+1.T^{Gh}h+1.L^{Gh}h+1).

By induction hypothesis, P has #term(ih.jh.T^{Gh-1}h.L^{Gh-1}h). P has term(ih.jh.T^{Gh}h) == term(ih+1.jh+1.T^{Gh}h+1) by result 2. And by ET2 or ET3 P has term(ih.jh.T^{Gh}h.L^{Gh}h) == term(ih+1.jh+1.T^{Gh}h+1.L^{Gh}h+1). Notice that the X passed in the h^{th} block is represented as both term(ih.jh.T^{Gh-1}h.L^{Gh-1}h) and term(ih.jh.T^{Gh}h.L^{Gh}h). Although the two locations are the same, they are named differently because T^{Gh} and T^{Gh-1} can be different. So P has #term(ih.jh.T^{Gh}h.L^{Gh}h). And by FT2, P has #term(ih+1.jh+1.T^{Gh}h+1.L^{Gh}h+1). Induction step is done.

Since Q can sense the fresh X at i.j.L in the last block in CSX, $Q \models_i$ Q @ term(i.j.L). P has Q @ term(i.j), showed by result 1. Then by LN1, P has Q @ term(i.j.L). If P = Q, it is trivially true.

We can show that Gn-1 is the sender and Q is the receiver of the n^{th} block in CSX. By applying the above induction, we get $P \models_{endP}$ #term(in.jn.T^{Gn-1}n.L^{Gn-1}n). Since in.jn.T^{Gn-1}n.L^{Gn-1}n is the same as i.j.L, $P \models_{endP}$ #term(i.j.L). lemma 2 is proved.

Lemma 3 If $P \models_h$ #term(i.j.L), Q @ term(i.j.L) $\in$ knowledge(P), and $Q \leftarrow_i \in$ SharedKnowledge, for some h, $h \leq$ endP, and some i.j.L, and some principal Q (Q can be P), X is an atomic SDC that appears at i.j.L, then for every block that is in a possible carrier sequence of X to i.j.L, say term(m.n), $P \models_h$ #block(m.n).

Proof: The conditions are logic world behavior. Since the EV(P) is constructed by the P's beliefs of fresh blocks, the results of this lemma will reflect the real world behavior. The proof of this lemma can demonstrate the searching backward ability of the rules.

Suppose there is a possible carrier sequence of X to i.j.L, call it CSX with length (the number of blocks) n. We name the blocks and principals in CSX the same way as in the proof of lemma 1. So Q is Gn, and term(i.j), the last block, is the same as term(in.jn). G_0 is the originator of X. It must be true that $n \geq 1$.

We can prove this lemma by tracing back CSX from the last block to the first block. Now we do induction to show that P will believe for each location H that X appears in CSX, P has #term(H).

Base step (on the n^{th} block, the last block in CSX): X is passed to Gn (which is Q) in the n^{th} block, which is term(in.jn), sent from Gn-1. Since $P \models_h$ #term(i.j.L), by the rule FT1, $P \models_h$ #term(i.j). Since $P \models_h$ G_n @ term(i.j.L), by

LN2, P $\models_h$ Gn @ term(i.j). And obviously, Gn-1$\rightarrow_{in,}$ Gn$\leftarrow_{in} \in$ SharedKnowledge. Then if P $\neq$ Q, A1 can be applied and will produce P $\models_h$ #block(i.j), Gn-1 @ term(i.j). Then by A2 P $\models_h$ Gn-1$^{active}{}_i$.

Further more, suppose a term t^{Gn-1}, represented as term(in.jn.T^{Gn-1}n), is a smallest term visible to Gn-1 in the n^{th} block sent by Gn-1, and t^{Gn-1} contains the passing X which is term(in.jn.T^{Gn-1}n.L^{Gn-1}n). Here in.jn.T^{Gn-1}n.L^{Gn-1}n is the same as i.j.L (in is i, jn is j, and T^{Gn-1}n.L^{Gn-1}n is L). So P $\models_h$ #term(in.jn.T^{Gn-1}n.L^{Gn-1}n). By FT1, P $\models_h$ #term(in.jn.T^{Gn-1}n). Gn-1 must have Gn-1 $\models_{in}$ Gn-1 @ term(in.jn.T^{Gn-1}n). Since P $\models_h$ Gn-1 @ term(in.jn), by LN1, P $\models_h$ Gn-1 @ term(in.jn.T^{Gn-1}n).

Induction step (on the m^{th} block) : Suppose all of the blocks, say the f^{th} block, such that f$\geq$ m $\geq$ 1 , P $\models_h$ #block(if.jf), #term(if.jf.T^{Gf-1}f.L^{Gf-1}f), #term(if.jf.T^{Gf-1}f), Gf-1 @ term(if.jf.T^{Gf-1}f). We want to show that the same result can be applied to the m-1^{th} block. Notice that if the m^{th} block is the first block in CSX, lemma 3 is already proved. We only need to consider the case the Gm-1 receives block(im-1.jm-1) and sends block(im.jm) to pass X in the middle of CSX.

Gm-1 must have Gm-1 $\models_{im}$ Gm-1 @ term(im.jm.T^{Gm-1}m), term(im-1.jm-1.T^{Gm-1}m-1) == term(im.jm.T^{Gm-1}m), Gm-1 @ term(im-1.jm-1.T^{Gm-1}m-1), since these knowledge about Gm-1's strand must have been reflected in Q's knowledge. It is obvious that P $\models_h$ Gm-1$^{active}{}_{im}$. By LN4, P $\models_h$ term(im-1.jm-1.T^{Gm-1}m-1) == term(im.jm.T^{Gm-1}m). Then by FT2, P $\models_h$ #term(im-1.jm-1.T^{Gm}m-1). By ET rules, P $\models_h$ term(im-1.jm-1.T^{Gm}m-1.L^{Gm}m-1) == term(im.jm.T^{Gm}m.L^{Gm}m). By FT2, P $\models_h$ # term(im-1.jm-1.T^{Gm}m-1.L^{Gm}m-1). By A1, P $\models_h$ #block(im-1.jm-1), Gm-2 @ term(im-1.jm-1). Then by A2, P $\models_h$ Gm-2$^{active}{}_{im-1}$. And then by LN1, P $\models_h$ Gm-2 @ term(im-1.jm-1.T^{Gm-2}m-1). Since term(im-1.jm-1.T^{Gm-1}m-1.L^{Gm-1}m-1) is the same as term(im-1.jm-1.T^{Gm-2}m-1.L^{Gm-2}m-1), P $\models_h$ # term(im-1.jm-1.T^{Gm-2}m-1.L^{Gm-2}m-1). By FT1, P $\models_h$ # term(im-1.jm-1.T^{Gm-2}m-1). Induction step is proved.

Following the induction step until m = 1. P will believe that all of the blocks in CSX are fresh.

Lemma 4 EV(P) is a run.

Proof: We can prove lemma 4 by contradiction. Suppose the earliest place where EV(P) fails is at i^{th} message, by Q. Then Q must be supposed to receive msg^i.

Notice that if Q can see a term, then Q can also sense a term. This is also reflected in the rules. We always apply lemma 2 and 3.

Case 1: The previous action of Q is not included in EV(P). This is impossible. By the way of constructing of EV(P), EV(P) will include the prefix of Q's strand up to LatestAction(P, Q), call it *LAQ*.

Case 2: Q sees an atomic SDC X in the i^{th} message, say at i.j.L, but Q sees X with a different value at location m.n.H, $m \leq i$. One of them is a fresh X and the other is an old X. By the way of constructing EV(P), m and i < LAQ. Q $\models_i$ term(i.j.L) == term(m.n.H) $\in$ knowledge(Q). This is true by the soundness of the rules to record the labeled strand. There are two sub-cases.

Case 2.1: Q sees (and sense) a fresh X at i.j.L, an old X at m.n.H. Depends on Q sends or receives the m^{th} message, there are two sub-cases.

Case 2.1.1: Q is the sender of the m^{th} message. Since Q sees an old X at m.n.H, then Q must not be the originator of X (otherwise Q has already failed earlier). So Q must have received and seen an old X at some message msg^j, j < m. Then this case is reduced to case 2.1.2, where Q receives the old X at some place other than i.j.L.

Case 2.1.2: Q is the receiver of the m^{th} message. Since Q receives and sees (also senses) a new X at i.j.L, by lemma 2, P $\models_{endP}$ Q @ term(i.j.L), #term(i.j.L). LAQ is Q is latest active point to P. Since Q $\models_{LAQ}$ term(i.j.L) == term(m.n.H), Q @ term(m.n.H), and i < LAQ, m < LAQ, by applying LN4, P $\models_{endP}$ term(i.j.L) == term(m.n.H), Q @ term(m.n.H). Then by FT2, P has #term(m.n.H). Then by lemma 3, Q must receive and sense a new X at m.n.H. Then Q can not see an old X at m.n.H. Contradictory. This case is impossible.

Case 2.2: Q sees (and sense) an old X at i.j.L, but a new X at m.n.H. Depending on Q sends or receives the m^{th} message, there are two sub-cases.

Case 2.2.1: Q is the sender of the m^{th} message.

Case 2.2.1.1: X originates at m.n.H from Q. Then Q $\models_{LAQ}$ #term(m.n.H). Also Q $\models_{LAQ}$ term(m.n.H) == term(i.j.L). By FT2, Q $\models_{LAQ}$ #term(i.j.L). By Lemma 3, for every block, say block(F), in some carrier sequence of X to i.j.L, Q $\models_{LAQ}$ #block(F). Then by LN6, P will learn all of these facts since P $\models_{endP}$

Q^{active}_{LAQ}. So P will have #block(F). These fresh blocks are all included in EV(P) as fresh blocks. So it is impossible for Q to receive an old X at i.j.L.

Case 2.2.1.2: X is not originated at m.n.H. Then Q must have received and sees a new X at some message earlier, which is reduced to a case that is equivalent to case 2.2.2.

Case 2.2.2: Q is the receiver of the m^{th} message. This case is impossible by the same reasoning of case 2.1.2.

Proof of completeness theorem: By Lemma 4, EV(P) is a run. We only need to show that EV(P) actually will trick P to accept some old atomic SDC X. Say L is the first place where P can see X. and it is reflected in the logic world and $P \models_{endP} P < term(L) \in knowledge(P)$. P is not the originator of X, because otherwise P will have $P \models_{endP}$ #term(L). So it is enough to only consider that case that P receives and sees X at L. The value of X that P sees (also senses) at L must be old, because otherwise by lemma 2, $P \models_{endP}$ #term(L) ∈ knowledge(P), and then $P \models_{endP} P < \#X$ by C1, and it is contradictory.

8. Conclusion and Future Work

This paper defines a common but non-trivial case of authentication attacks. A complete and sound algorithm with polynomial time complexity is presented. It has efficient features for further optimizations and extension. A logic in the style of using beliefs is used as a frame work to construct the algorithm. This logic is based directly on the protocol code and is designed naturally and is easy to be verified. The model we called labeled strand can be applied for checking general kinds of attacks for security protocols. The algorithm is fully automatic. All the protocol examples (including the one that BAN logic can not derive the freshness, see appendix 1) appearing in this paper can be verified easily using the rules of this logic.

In this paper, we take the advantage that in OMA freshness means honesty (if a block is fresh, then the block is sent and received by the honest principal, without modification). In order to deal with more tricky attacks, the freshness and honesty must be separated, and new rules must be designed. We are also interested to see how these efficiency features suggested by the rules can be fully utilized to remove the redundancy of computation to the maximum level. We are continuing this research.

References

1. Martin Abadi, Mark Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*. ACM Press, 1991. pages 201-216.
2. Roberto M. Amadio, and Denis Lugiez. On the reachability problem in cryptographic protocols. In *CONCUR* (2000), vol. 1877 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 180-394.
3. David Basin, Sebastian Mödersheim, Luca Viganò.OFMC: A Symbolic Model-Checker for Security Protocols. ETH Zürich, Computer Science, Technical Report 450, 2004. *http://kisogawa.inf.ethz.ch/WebBIB/publications/papers/2004/0_tr450.pdf*
4. Michael Burrows, Martín Abadi, and Roger Needham: A logic of authentication. *Proceedings of the 12th Symposium on Operating System Principles* (Litchfield Park, Arizona, Dec., 1989). ACM, 1989, pages 1-13.
5. Pierre Bieber. A Logic of Communication in a Hostile Environment. In: *Proceedings of the Computer Security Foundations Workshop III*. Los Alamitos, IEEE Computer Society Press, 1990, pages 14-22.
6. Edmund Clarke, Somesh Jha, and Will Marrero. Verifying security protocols with Brutus. In *ACM Transactions on Software Engineering and Methodology*. Vol 9, issue 4, October 2000, pages 443-487
7. Dorothy .E. Denning and Giovanni Maria Sacco, 1981 Timestamps in Key Distribution Protocols. *CACM* vol. 24, No. 8, pp. 533-536.
8. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2): March 1983, pages 198-208.
9. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols* (1999), FLOC.
10. Klaus Gaarder, Einar Snekkenes. On the Formal Analysis of PKCS Authentication Protocols. Advances in Cryptology – AUSCRYPT'90, vol 453, *Lecture Notes in Computer Science*, Springer-Verlag, pp. 106-121.
11. Li Gong. Variations on the Themes of Message Freshness and Replay. In *Proceedings of the IEEE Computer Security Foundations Workshop VI*, New Hampshire, June 1993, pages 131 - 136, Franconia.
12. Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE symposium on Research in Security and privacy, Oakland, California*, May 1990

13. Nevin Heintze, and J. D. Tygar. **A** model for secure protocols and their compositions. *IEEE Transactions on Software Engineering 22*, 1 (1996), pages 16-30.
14. Nevin Heintze, J. D. Tygar, Jeanette Wing, and H. Chi Wong. Model checking electronic commerce protocols. In *Proceedings of the USENIX 1996 Workshop on Electronic Commerce*, 1996 pages 146-164.
15. Ralf Küsters, Thomas Wilke. Automata-based analysis of recursive cryptographic protocols. *STACS* 2004, pages 382-393.
16. Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3), November 1995, pages 131-136.
17. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of systems*, volume 1055 of *Lecture Notes in Computer Science*, Springer-Verlag, 1996, pages 147-166.
18. Catherine Meadows. Analysis of the internet key exchange protocol using the NRL protocol analyzer. *IEEE Symposium on Security and Privacy* 1999, pages 216-231.
19. Jonathan K. Millen, Vitaly Shmatikov: Constraint solving for bounded-process cryptographic protocol analysis. *ACM Conference on Computer and Communications Security* 2001, pages 166-175.
20. John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using murφ. In *Procedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.
21. Roger Needham and Michael Schroeder. Using encryption for authentification in large networks of computers. *Communications of the ACM*, 21(12), December 1978
22. Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.
23. Dan M. Nessett: A critique of the Burrows, Abadi and Needham logic. *Operating Systems Review* 24 (1990), 35-38
24. Paul C. van Oorschot (1994). An alternate explanation of two BAN-logic "failures". Eurocrypt'93, *LNCS* vol. 765, Springer-Verlag, pages 443-447.
25. Michaël Rusinowitch, Mathieu Turuani: Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science* issues 2003 Issues 1-3, Vol. 299, pages 451-475
26. Lawrence C. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 70-83, 1997
27. Aviel D. Rubin, Peter Honeyman. Nonmonotonic Cryptographic Protocols, *IEEE CSFW* 1994, 100-116.
28. Dawn Xiaodong Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*. Los Alamitos: IEEE Computer Society Press, 1999.
29. Dawn Xiaodong Song, Sergey Berezin, Adrian Perrig: Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. In *Journal of Computer Security* 2001, Vol. 9, number 1/2, pages 47-74.
30. P. Syverson. Formal semantics for logics of cryptographic protocols. In *Proceedings of the Computer Security Foundations Workshop III*. Los Alamitos, IEEE Computer Society Press, 1990, pages 32-41.
31. Paul F. Syverson, Paul C. van Oorschot.. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*. Los Alamitos, IEEE Computer Society Press, 1994, pages 14-28.
32. F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, 1998.
33. F. Javier Thayer, Jonathan C. Herzog, Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 1999, vol. 7, issues 2-3, pages 191-230.
34. F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *11th IEEE Computer Security Foundations Workshop* 1998, pages 66-78.

.

Appendix 1

We include the rule derivations of checking one protocol example in the appendix.

If a fact is an initial knowledge of a principal P, or the fact represents some knowledge about the strand of P, the fact can be achieved without applying reasoning rules. We include this kind of facts and the natural facts and the shared facts directly into the examples, using *italic font*, without mentioning how to derive them, for clarity of the presentation and to emphasize on the derivation using reasoning rules. Only interesting beliefs are included. For each belief derived using the reasoning rule, an id is assigned to it in the form of A-2-3 (A's third belief in message 2) as the superscript. The reasoning rule used is marked as the subscript. The monotonic rule M1 is applied without being mentioned. The checking process with start from the first message to the last message of the protocol. The conclusion rule C1 is highlighted.

Example: Checking Needham-Schroeder with Shared Key Protocol (protocol 4).

1. $A \rightarrow S: A, B, N_A$

2. S → A: $\{N_A, B, K_{AB}, \{K_{AB}, A\}^{\leftrightarrow}K_{BS}\}^{\leftrightarrow}K_{AS}$
3. A → B: $\{K_{AB}, A\}^{\leftrightarrow}K_{BS}$
4. B → A: $\{N_B\}^{\leftrightarrow}K_{AB}$
5. A → B: $\{N_B - 1\}^{\leftrightarrow}K_{AB}$

In this protocol, the last message can also be presented as A → B: $\{f(N_B)\}^{\leftrightarrow}K_{AB}$, where f is a function. To deal with this kind of functions, a labeled strand rule can be added as

$P \models_i P\ @_L\ f(X)$ ➔ $P\ @_{L.\gamma}\ X$	(F1)

A will see three SDCs: N_A, K_{AB}, and N_B. B will see two SDCs: N_B and K_{AB}. S can see N_A and K_{AB}. There is no way for S to verify that N_A is fresh. We will show that A and B will receive and see fresh SDCs.

Message 1:
$A \models_1 \#N_A, A < term(1.3), term(1.3) == N_A$
$\rightarrow_{C1} A \models_1 A < \#N_A^{A\text{-}1\text{-}1}$

Message 2:
$S \models_2 \#term(2.\alpha.3), S < term(2.\alpha.3), term(2.\alpha.3) == K_{AB} \rightarrow_{C1} S \models_2 S < \#K_{AB}^{S\text{-}2\text{-}1}$

$A \models_2 \#term(2.1.\alpha.1) \rightarrow_{FT1} A \models_2 \#term(2.1)^{A\text{-}2\text{-}1}$

$\{A \models_2 \#term(2.1)^{A\text{-}2\text{-}1}, A @ term(2.1)\}$;
$\{S \rightarrow_2, A \leftarrow_2\}, A \neq S$
$\rightarrow_{A1} A \models_2 S @ term(2.1)^{A\text{-}2\text{-}2}$
$A \models_2 S @ term(2.1)^{A\text{-}2\text{-}2} \rightarrow_{A2} A \models_2 S^{active}{}_2{}^{A\text{-}2\text{-}3}$

$A \models_2 S @ term(2.1)^{A\text{-}2\text{-}2}, S^{active}{}_2{}^{A\text{-}2\text{-}3}$
$S \models_2 S @ term(2.1.\alpha.3)$; $A \neq S$
$\rightarrow_{LN1} A \models_2 S @ term(2.1.\alpha.3)^{A\text{-}2\text{-}4}$

$\{A \models_2 S @ term(2.1.\alpha.3)^{A\text{-}2\text{-}4}, S^{active}{}_2{}^{A\text{-}2\text{-}3}\}$;
$S \models_2 \#term(2.1.\alpha.3)$; $S \neq A$
$\rightarrow_{LN3} A \models_2 \#term(2.1.\alpha.3)^{A\text{-}2\text{-}4}$

$\{A \models_2 \#term(2.1.\alpha.3)^{A\text{-}2\text{-}4}, A < term(2.1.\alpha.3), term(2.1.\alpha.3) == K_{AB}\} \rightarrow_{C1} A \models_2 A < \#K_{AB}^{A\text{-}2\text{-}5}$

Message 3:
$B \models_3 B < term(3.1.\alpha.1)$

Message 4:
$B \models_4 B < term(4.1.\alpha), term(4.1.\alpha) == N_B, \#N_B$
$\rightarrow_{C1} B \models_4 B < \#N_B^{B\text{-}4\text{-}1}$

$A \models_4 term(4.1.\beta)==term(2.1.\alpha.3), \#term(2.1.\alpha.3)^{A\text{-}2\text{-}4}$
$\rightarrow_{FT2} A \models_4 \#term(4.1.\beta)^{A\text{-}4\text{-}1}$

$A \models_4 \#term(4.1.\beta)^{A\text{-}4\text{-}1} \rightarrow_{FT1} \#term(4.1)^{A\text{-}4\text{-}2}$

$A \models_4 \#term(5.1)^{A\text{-}4\text{-}2}, A @ term(4.1)\}$;
$\{B \rightarrow_4, A \leftarrow_4\}; A \neq B \rightarrow_{A1} B @ term(4.1)^{A\text{-}4\text{-}3}$

$A \models_4 B @ term(4.1)^{A\text{-}4\text{-}3} \rightarrow_{A2} B^{active}{}_4{}^{A\text{-}4\text{-}4}$

{A |=$_4$ B$^{active}{}_4{}^{A\text{-}4\text{-}4}$, B @term(4.1)$^{A\text{-}4\text{-}3}$ };
{*B @ term(4.1.α)* }; *A ≠ B*
➔$_{LN1}$ A |=$_4$ B @ term(4.1.α)$^{A\text{-}4\text{-}5}$

{A |=$_4$ B @ term(4.1.α)$^{A\text{-}4\text{-}5}$, B$^{active}{}_4{}^{A\text{-}4\text{-}4}$ } ;
{B |=$_4$ *#term(4.1.α)* }; *A ≠ B*
➔$_{LN3}$ A |=$_4$ #term(4.1.α)$^{A\text{-}4\text{-}6}$

A |=$_4$ #term(4.1.α)$^{A\text{-}4\text{-}6}$, *A < term(4.1.α)*,
term(4.1.α) == N_B
➔$_{C1}$ A |=$_4$ A < #$N_B{}^{A\text{-}4\text{-}7}$

Message 5:
A |=$_5$ *term(5.1.β) == term(2.1.α.3)*,
#term(2.1.α.3)$^{A\text{-}2\text{-}4}$
➔$_{FT2}$ A |=$_5$ #term(5.1.β)$^{A\text{-}5\text{-}1}$

B |=$_5$ *#term(5.1.α.γ)* ➔$_{FT1}$ #term(5.1)$^{B\text{-}5\text{-}1}$,

{B |=$_5$ #term(5.1)$^{B\text{-}5\text{-}1}$, *B @ term(5.1)* };
{*A →$_5$, B ←$_5$*}; *A ≠ B* ➔$_{A1}$ A @ term(5.1)$^{B\text{-}5\text{-}2}$

B |=$_5$ A @ term(5.1)$^{B\text{-}5\text{-}2}$ ➔$_{A2}$ B |=$_5$ A$^{active}{}_5{}^{B\text{-}5\text{-}3}$

{ B |=$_5$ A$^{active}{}_5{}^{B\text{-}5\text{-}3}$, A @ term(5.1)$^{B\text{-}5\text{-}2}$ };
{ A |=$_5$ *A @ term(5.1.β)* }; *A ≠ B*
➔$_{LN1}$ B |=$_5$ A @ term(5.1.β)$^{B\text{-}5\text{-}4}$

{ B |=$_5$ A @ term(5.1.β)$^{B\text{-}5\text{-}4}$, A$^{active}{}_5{}^{B\text{-}5\text{-}3}$ };
{ A |=$_5$ #term(5.1.β)$^{A\text{-}5\text{-}1}$ }; *A ≠ B*
➔$_{LN3}$ B |=$_5$ #term(5.1.β)$^{B\text{-}5\text{-}5}$

Comments:
B-5-4 and B-5-5 can also be derived using LN6, which may be simpler

B |=$_5$ *term(5.1.β) == term(3.1.α.1)*,
#term(5.1.β)$^{B\text{-}5\text{-}5}$
➔$_{FT2}$ B |=$_5$ #term(3.1.α.1)$^{B\text{-}5\text{-}6}$

B |=$_5$ #term(3.1.α.1)$^{B\text{-}5\text{-}6}$, *B < term(3.1.α.1)*,
term(3.1.α.1) == K_{AB}
➔$_{C1}$ B |=$_5$ B < #$K_{AB}{}^{B\text{-}5\text{-}7}$

Example 2 : Yahalom (protocol 2 in the introduction section)
Why will A and B accept only fresh NA, NB, and KAB?

1. A -> B: A, N_A
2. B -> S: B, {A, N_A, N_B}$^{\leftrightarrow}K_{BS}$
3. S -> A: {B, K_{AB}, N_A, N_B}$^{\leftrightarrow}K_{AS}$, {A, K_{AB}}$^{\leftrightarrow}K_{BS}$
4. A -> B: {A, K_{AB}}$^{\leftrightarrow}K_{BS}$, {N_B}$^{\leftrightarrow}K_{AB}$

Sketch of the rule applications are provided here

Message 1:
A believe A can see a fresh N_A by conclusion rule

Message 2: omitted.

Message 3:
- A sense the N_A inside the first block, term(3.1),
- By A1, A knows S can sense the firs block
- By A2, A knows S is active at message 3.
- By LN1, A learns that S can sense the K_{AB} inside the first block of message 3
- Since S believes K_{AB} inside message 3 is fresh, and A knows S is active at message 3, A will learn that K_{AB} in message 3 is fresh, which A can see.
- A learn by LN4, A search back and knows that N_A appears at message 2 and message 3. S can sense message 2.
- A learns by LN1 that S can sense the N_B inside message 2.
- By LN5, A learns that the equivalence of the two N_B inside message 2 and message 3.
- Since N_A appears in message 2, A knows that the block in message 2 is fresh, by FT1.
- By A1 and A2, A knows that B is active at message 2. B can sense message 2.
- A learn from B that B can sense the N_B inside message 2, By LN1.
- Since B believes that N_B is fresh in message 2, A will learn from B about it.
- A knows the two N_B in message 2 and message 3 is equivalent, so A will know that N_B in message 3 is also fresh, which A can see.

Message 4:
- A believes that the K_{AB} in message 4 is fresh
- B sense the N_B in message 4, and B believes that it is fresh.
- B believes the second block in message 4 is fresh. And A must be active at message 4.
- B learns from A that A can sense the K_{AB} in the second block in message 4.
- B learns from A that the K_{AB} in the second block in message 4 is fresh, which B can see.
- B search back and knows that N_B appears in both message 2 and message 3. And S is both active at message 2 and message 3.
- B learns from S that the two N_A are equivalent in message 2 and message 3.
- A learns from A that the N_A in message 3 is fresh.
- Since B knows that the two N_A in message 2 and message 3 are equivalent, B knows that the N_A in message 2 is fresh, which B can see.

A knows N_A is fresh since A originates N_A. Similarly B knows N_B is fresh.

Example 3: Yahalom with message 2 changed :
Why can A accept an old NB while B can only accept a fresh NA? Why A and B can still only accept fresh KAB?

1. A -> B: A, N_A
2. B -> S: B, $\{A, N_A\}^{\leftrightarrow}K_{BS}$, $\{B, NB\}^{\leftrightarrow}K_{BS}$
3. S -> A: $\{B, K_{AB}, N_A, N_B\}^{\leftrightarrow}K_{AS}$, $\{A, K_{AB}\}^{\leftrightarrow}K_{BS}$
4. A -> B: $\{A, K_{AB}\}K_{BS}$, $\{N_B\}^{\leftrightarrow}K_{AB}$

With the same reasoning as the original Yahalom,

```
A |= A @ #term(3.1.1.2), A @ #KAB
B |= #term(4.2.0), B @ #KAB, #term(3.1.1.3),
     term(3.1.1.3) == term(2.2.1.2), #term(2.2.1.2)
     B @ #NA
```

But there is no way to generate `A |= A @ #NB`, because it is impossible to have `A |= #term(2.2)`, while in the original Yahalom, `A |=` #term(2.2) by searching back N_A

Example 4: Protocol 1 in the introduction session.

1. A -> B: A
2. B -> A: $\{X, M\}K_A$

3. A -> B: $\{X, N_A\}K_B$
4. B -> A: N_A, $\{M, W\}K_A$

Sketch of the reasoning is provided here.

In message 3,

- B sense the received fresh X, so B knows that A is active at message 3.
- Then B learn from A that A can sense the N_A inside message 3 and it is fresh, which B can see.

In message 4,

- A sense the fresh N_A, A knows that B is active at message 4.
- By searching back using LN4, A knows that N_A also appears in message 3, and B can sense it.
- A knows that B is also active at message 3 and B can sense the X inside message 3.
- A learns from B that the X inside message 3 is fresh, which A can see.
- Then A knows that the X inside message 2 is also fresh.
- A knows B must be active at message 2.
- A learns from B that the M inside message 2 is fresh, which A can see.
- Then A knows that M inside message 4 is also fresh.
- A knows that B must sense the second block in message 4, and the W inside it.
- A learns from B that the W inside message 4 is fresh, which A can see.

Appendix 2

Suggestions to Optimize the Computation of Closure.

Since the same fact can be computed by different rules, we can organize the computation to remove the redundancy. The idea is that if a fact can be learned from other principals, then the fact does not need to be computed again.

The following steps can simulate the behavior of the real world naturally in the style that the principals' knowledge will be accumulated following the steps of the protocol.

Reasoning Stage Steps:

Suppose the protocol has N messages. And for msg^j $1 \le j \le N$, then sender is S_j and the supposed receiver is R_j. For each message msg^j in the protocol, starting from the first message to the last message, do the following. Only beliefs of the form $P \models_j$ facts, $j \le i$, is allowed to be used to apply rules.

1.) S_j update knowledge(S_j) as much as possible.
2.) R_j update knowledge(R_j) as much as possible, LN6 has the higher priority than other rules. So R_j will learn from LN6 first as much as possible before applying other rules to do reasoning.

The above steps will collect the knowledge by the easiest way. What remains to be interesting is how to shut down some rule computations when the result facts of the rule computation are already known. The full solution of this optimization is not shown in this paper. The following are some observation that we can take advantage of to design the optimal closure computation algorithm.

1.) If Q is causally earlier than P and Q's knowledge about fresh blocks is the same as P (not including the last block Q sent to P), then P and Q's knowledge are the same and P does not need to do the reasoning the explore new knowledge.
2.) The goal of the computation is to know whether a term is fresh or not, and its equivalent terms. So the computation can be divided to sub-problems of checking each term to know if it is fresh, and its equivalent terms. When a sub-problems is already computed (it can be learned from other principals), it does not need to be computed again.