Complexity of Checking Freshness of Cryptographic Protocols [*] [†]

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-08-14

September 27, 2008

Keywords: Cryptographic protocol, freshness, replay attack, challenge-response, model checker, undecidability, NP-completeness, Athena.

Abstract

Freshness is a central security issue for cryptographic protocols and is the security goal violated by replay attacks. This paper is the first to formally define freshness and its attacks based on role instances and the attacker's involvement, and is the first work to investigate the complexity of checking freshness. We discuss and prove a series of complexity results of checking freshness in several different scenarios, where the attacker's behavior is restricted differently, with different bounds on the number of role instances in a run.

[*] Research supported in part by NSF grants CCF 0306475 and CNS 0755500.
[†] This paper covers our paper [1] accepted by ICISS08 and provides more details.

Complexity of Checking Freshness of Cryptographic Protocols [*] [†]

Zhiyao Liang Rakesh M. Verma

Abstract

Freshness is a central security issue for cryptographic protocols and is the security goal violated by replay attacks. This paper is the first to formally define freshness and its attacks based on role instances and the attacker's involvement, and is the first work to investigate the complexity of checking freshness. We discuss and prove a series of complexity results of checking freshness in several different scenarios, where the attacker's behavior is restricted differently, with different bounds on the number of role instances in a run.

Index Terms

Cryptographic protocol, freshness, replay attack, challenge-response, model checker, undecidability, NP-completeness, Athena.

I. Introduction

SECURITY of communication protocols is critical in this age when computer communication is ubiquitous. An important research direction in verifying communication protocols is checking attacks while assuming perfect cryptography and a dominant attacker in the network, commonly referred as the Dolev and Yao attacker model [2]. Many researchers follow Dolev and Yao attacker model. Much research on the complexity of checking security goals has focused on checking secrecy [3] [4] [5] [6] [7] [8] [9].

Freshness is a central and fundamental issue of communication protocols [10]. The common ways to maintain freshness of terms of a protocol, without arguing the exact definition of freshness, are by using timestamps or by challenge-response [11]. Intuitively, in a protocol run a term may be considered as "fresh" in two aspects: how the term is created, and how the term is received. When a term is created we may say it is fresh, or it is new or not stale, if it is not created before a certain time, while time can be measured by the timestamps of terms, or time is implicitly referred, such as the creation time of a term. In [6] freshness means uniqueness, which means when a fresh term is created it should not have appeared in the run before, and this approach may also be categorized as emphasizing the freshness of a term on the creation aspect. The freshness of terms on the reception aspect, if implemented by timestamps, could mean that only terms with timestamps after a certain time point can be received in a certain situation. When a regular agent A participates in a protocol run, it is guaranteed that the A will create really fresh terms such as nonces if we assume unbounded creation of fresh terms, but what can cause security failure are the terms that are received by A, which could be not "fresh" when they are supposed to be "fresh"; therefore, we think to define the freshness of terms on the reception aspect is what really matters and deserves more attention. Timestamps have the limitation of relying on a precise global clock. On the contrary, the challenge-response mechanism indirectly restricts how a term can be received and accepted, i.e., a challenge must be passed in order to let a term be accepted. Comparing the challenge-response approach and the timestamp approach to implement and define freshness, we consider the former has wider coverage of cryptographic protocols. Obviously the former is more complicate to analyze. In this paper we address the freshness that may be implemented by challenge-response. Further discussions on the challenge-response mechanism are provided in Appendix A.

The contributions of this paper are as follows.

- We define freshness based on role instances, the terms that are supposed to be fresh, and the attacker's involvement.
- We address different scenarios where the attacker's behavior in a run is restricted differently. We define three kinds of replay attacks that violate freshness, called direct, restricted and general replay attacks.

[*] Research supported in part by NSF grants CCF 0306475 and CNS 0755500.
[†] This paper covers our paper [1] accepted by ICISS08 and provides more details.

- We address three bounds on the number of role instances in a run (NRI), including fixed, individually bounded, and unbounded. This paper is the first to clarify the difference between the three bounds.
- We address and prove a series of the complexity results of checking freshness for DRA, RRA, and GRA, when NRI is fixed, individually bounded, or unbounded. These results are non-trivial to prove. For example, the proof of Theorem 5, which shows the NP-completeness of checking RRA when NRI is fixed, is quite delicate.
- We analyze the performance of the model checker Athena [12] [13]. We improve the presentation, semantics, and efficiency of the algorithm of Athena.

To the best of our knowledge, the closest definition of a freshness goal that is independently defined by other researchers appears in [14], quoted in Appendix B below. Our work is significantly different from [14], and cannot be covered by [14], for the following reasons. First, in [14] the authors demonstrate a freshness goal can be expressed using the constraint solving system, but there is no complexity investigation. It is obvious that a freshness goal, which is defined later in this paper, can be expressed in different systems, since its definition is simple and clear. Second, the attacker's involvement is not discussed in [14] for the definition of a freshness goal. Third, the definition of a freshness goal in [14] is less generally applicable than the one defined in this paper, which is discussed later. Fourth, the discussion on freshness in [14] is sketchy and is not the focus of [14].

An observation is presented in Appendix B, which is relevant to understand the intuition of the definition of freshness.

We do not need to follow the detailed behavior of challenge and response in order to investigate the complexity of checking freshness. Therefore our approach is rather different from, and cannot be covered by other papers that design rules or logic to address the details of challenges and responses, such as [15] and [16], and they do not address complexity issues.

In Section III-A we discuss the reductions between checking freshness and other problems, and especially we explain why we think some complexity results of checking freshness cannot trivially be obtained by reductions between checking secrecy and freshness.

Section II presents the notations of terms of protocols, and the definitions of freshness, different attacks that violate freshness, and different bounds on the number of role instances in a run. Section III presents the complexity results and the proofs. Section IV summarizes this paper.

II. Notations and Modeling

A. Notations

Notations are chosen in a style that is commonly used in the literature. More details of notations and modeling can be found in [17]. A *term* is either an atomic term or a composite term. An *atomic term* is a *variable* (represented by a symbol with at least one upper case letter), or constant (a symbol without any upper case letter). The special constant $\mathcal{I}$ is the name of the attacker. Asymmetric keys are atomic terms. The established public key and private key of an agent A are k_A^1 and k_A^0 respectively. A *composite term* is a list, or an asymmetric encryption, or a symmetric encryption. A *list* has the form of $[X, Y, \cdots]$, where X and Y are terms and the list contains finite number of member terms. A list is a simpler representation of a sequence of nested pairs. For example $[W, X, Y, Z]$ is the same as $[W, [X, [Y, Z]]]$. When a message is a list, the top level enclosing [ ] is omitted. A term constructed by encryption algorithms is called an *encryption*. An *asymmetric encryption* has the form of $\{T\}_{k_A^i}^{\rightarrow}$, $i \in \{0, 1\}$, where T is called the *text*, and k_A^i, for $i \in \{0, 1\}$, is the atomic encryption key, and it can be decrypted using the key k_A^{1-i}. A *symmetric encryption* has the form of $\{T\}_Y^{\leftrightarrow}$, where T is the text, and Y is the encryption key, which can be any term and could be a composite term. When a list, say $[X, Y, Z, \cdots]$ is the text of an encryption, the enclosing square brackets are removed within "{ }". When a message is a list, the enclosing square brackets of the list are also omitted. The word *ground* means variable free.

The set of ***blocks*** of a term T, denoted as $blocks(T)$, is defined as follows:

- If T is an encryption or an atomic term, $blocks(T) = \{T\}$.
- If $T = [X, Y]$, then $blocks(T) = blocks(X) \cup blocks(Y)$.

An ***action step*** can have one of the following three forms. A term sent or received in an action step is called a ***message***.

- If agent P generates a set of (at least one) fresh terms, and then sends a message Msg to agent B, then the action step has the form
$$\#_P(T_1, T_2 \cdots) \ + (P \Rightarrow B) : Msg.$$
Here $\#_P(T_1, T_2 \cdots)$ is the fresh term generation action of P to generate the fresh terms T_1, T_2 etc. These fresh terms should appear as subterms in Msg.
- If agent P sends a message Msg to B without generating any fresh terms, then the action step is denoted as $+(P \Rightarrow B) : Msg$.
- If P receives a message Msg, and by the context of the communication or by analyzing Msg, P considers the supposed sender of Msg should be B, then the action step has the form $\quad -(B \Rightarrow P) : Msg$.

A ***communication step*** can also have three possible forms, which are the same as an action step, except the $+$ and $-$ signs are not used. A communication step implies two corresponding action steps, one is the message sending, maybe with nonce generation, by the sender, and the other is the message receiving by the receiver. A ***communication sequence***, or simply ***CS***, is a sequence of communication steps numbered sequentially starting from 1. A protocol is commonly described as a CS accompanied with other information including the initial knowledge patterns of agents.

B. Model of a Protocol Run

Here we present the essential model of protocol run. More details can be found in [17]. We assume the *free term algebra*, which means that two different symbolic terms must represent two different bit-strings of the real world. This assumption is common for the Dolev-Yao model. We assume *unbounded fresh nonce generation*, which means that when a nonce is generated, it is always different from other nonces and all the terms that have appeared in the run before its generation, and is different from all the terms initially known to some agent.

A ***role*** is a sequence of action steps executed by the same agent A obtained by parsing the CS of a protocol, which is called A's role.

An ***event*** is a tuple $\langle act,\ time\rangle$, where act is a ground action step, described earlier; and, the $time$ field of an event e is referred as $e.time$, which is a positive real number representing when the event occurs after the start of the run.

A ***role instance*** r of role R is a sequence of events, such that the action steps of the sequence of events in r instantiate a prefix of the sequence of action steps of R, not necessarily all of the action steps of R, by a ground substitution. For two events ev and ev' in r, if their message numbers in the corresponding role R are n and n' respectively, and $n < n'$, then $ev.time < ev'.time$.

In a run the ***attacker*** is associated with a set of ground terms that are initially known to $\mathcal{I}$, denoted as $\mathcal{I}.init$. The attacker can analyze and synthesize terms and create nonces by following a set of standard rules, which are presented below, and they are similarly discussed in [18] and [5].

Given a certain set E of events that have occurred in a run, $know_{\mathcal{I}}(E)$ represents the (infinite) set of terms that the attacker can know, and for a term T, $T \in know_{\mathcal{I}}(E)$ is defined as follows. The set of terms $\mathcal{S}_0$ *built on* E is defined as the smallest set of terms that satisfy the following conditions.

- $\mathcal{I}.init \subseteq \mathcal{S}_0$
- $\{msg \mid msg$ is the message sent in a regular event η, $\eta \in E\ \} \subseteq \mathcal{S}_0$

A rule l, of the form $LHS \mapsto RHS$, can be applied to a set of terms $\mathcal{S}$. For a rule l to be applicable to $\mathcal{S}$, except the creation rule Cre, all of the terms included in LHS must be included in $\mathcal{S}$. After applying l to $\mathcal{S}$, a new set $\mathcal{S}'$ of terms is formed by adding terms in RHS to $\mathcal{S}$. The notation $\#_{\mathcal{I}}(X \cdots)$ means the attacker generates one or more fresh nonces .

- Creation
  - Cre: $\#_{\mathcal{I}}(X \cdots) \quad \mapsto X \cdots$
- Synthesis
  - Syn_1: $X, Y, Z, \ldots \quad \mapsto [X, Y, Z, \ldots]$
  - Syn_2: X, k^i_G for $i \in \{0, 1\} \quad \mapsto \quad \{X\}^{\rightarrow}_{k^i_G}$
  - Syn_3: $X, Y \quad \mapsto \quad \{X\}^{\leftrightarrow}_{Y}$.
- Analysis
  - Ana_1: $[X, \cdots Y] \quad \mapsto \quad X, \cdots Y$

- Ana_2: $\{X\}^{\rightarrow}_{k_G^m}, k_G^{1-m}$ for $m \in \{0,1\} \quad \mapsto \quad X$
- Ana_3: $\{X\}^{\leftrightarrow}_{Y}, Y \quad \mapsto \quad X$

A ***derivation*** with length j, $0 \leq j$, has the form $\mathcal{S}_0 \rightarrow^{l_1} \mathcal{S}_1 \rightarrow^{l_2} \cdots \mathcal{S}_{j-1} \rightarrow^{l_j} \mathcal{S}_j$, where $\mathcal{S}_h$, $1 \leq h \leq j$ is a set of terms obtained by applying rule l_j to $\mathcal{S}_{j-1}$. $\mathcal{S}_0$ is a derivation from $\mathcal{S}_0$ with length 0. A derivation starting from the $\mathcal{S}_0$ built on E is called a *derivation on* E. $T \in know_{\mathcal{I}}(E)$ if and only if there is a derivation on E with length j such that $T \in \mathcal{S}_j$, for some $j \geq 0$.

A ***run*** is a tuple: $\langle Pro,\ D,\ R,\ AN,\ E\rangle$, where Pro is the protocol; D is the initial knowledge pattern of the attacker who is involved in the run; R is a set of role instances that are executed honestly by regular agents; AN is the set of ground names of the agents who participate in the assumed perfect initial knowledge establishing stage of the run, including all of the regular agents and sometimes the attacker, and the agents in AN are insiders; E is a set of events that occur in the run, including, nothing more and nothing less, all of the events of the role instances in R. The E field of the run is referred as $run.E$. Other fields of the run are similarly referred. The set of ***time points*** of the run is defined as $\{t \mid t = \eta.time,\ \eta \in run.E\} \cup \{0\}$. Given a time point t, we define $E_{<t}$ as the set of events $\{\eta \mid \eta.time < t, \eta \in run.E\}$. The following conditions should be satisfied: For any event η in E, if η receives a message msg then $msg \in know_{\mathcal{I}}(E_{<\eta.time})$. The set of all possible runs of a protocol Pro and with some specific initial knowledge pattern D of the attacker, is denoted as $Runs^{D:Pro}$. Note that we allow two events to occur at the same time in a run, which is different from the trace based models like [18] and [7], but agrees with the strand space model [19]. Although the strand space model is equivalent to the trace based models in terms of checking security failures, we think strand space model seems to be more close to reality and using it to check protocols does not require extra effort.

C. Definition of Freshness and Its Attacks

Definition 1: Given a certain pattern D of the attacker's initial knowledge, and a protocol Pro, where a role of A receives a term X which should be freshly generated by B's role such as a nonce variable, the ***freshness*** of X to A's role is that it is impossible to have a run, $run \in Runs^{D:Pro}$, where there are two different role instances r and r' of A's role, such that the following two conditions are satisfied:

- In r and r', B is not instantiated by $\mathcal{I}$, which is the attacker's name.
- The same ground term, say c, instantiates X in both r and r'.

The security goal that the freshness of a term to a role in a protocol is called a ***freshness goal***.

Note that in the above definition r and r' do not need to be executed by the same agent, i.e., A may be instantiated by different agents in r and r', which is more generally applicable than the freshness defined in [14]. We require that B is not instantiated by $\mathcal{I}$, since if the nonce X is supposed to be generated by the attacker, then the attacker can obviously send the same X to both r_1 and r_2.

Freshness, as defined above, is a necessary condition of authentication. Lowe provided some well-known definitions of authentication goals in [20], and the strongest definition is the follows. The protocol, which is a communication sequence, implies a set of variables that appear in both A's role and B's role, which is called the set of shared data. The authentication goal of B's role to A's role, for any two agent variables A and B, is that in every run of the protocol, when a role instance r of B finishes execution, there is a one-to-one correspondence between r and another role instance r' of A's role such that in r and r' the shared variables are instantiated by the same values. In order to implement the one-to-one correspondence, commonly a nonce N_A is created by A and received by B, and N_A is shared by A's role and B's role. If the authentication goal of B's role to A's role is satisfied then the freshness goal of N_A to B's role is obviously satisfied. However the freshness goal is not sufficient for the authentication goal. Even when all of the freshness goals to B's role of the nonce variables shared between A and B are satisfied, the authentication goal of B's role to A's role may still not be satisfied, since it is possible that the values of these shared variables in a role instance r of B's role does not appear together in a single role instance r' of A's role. The idea of authentication goal is to describe a condition that should not be violated in the normal situation, which is a one-to-one correspondence between two role instances. The freshness goal defined above extends this idea to describe a one-to-one correspondence between a role instance and a nonce that normally should be satisfied.

A unique ***location*** is assigned to each occurrence of a term in the messages of a communication sequence of a protocol as follows.

- The message with message number i, $1 \leq i$, has location i.
- If an encryption $\{X\}_Y^{\rightarrow}$ or $\{X\}_Y^{\leftrightarrow}$ has the location L, then X is located at $L.\alpha$, and Y is located at $L.\beta$.
- If a term $[X, Y, \cdots]$ has location L, then the members X, Y $\cdots$ have the locations, respectively, $L.1$, $L.2$, $\cdots$.

Since roles are parsed from a CS, the location of an occurrence of a term T in a role is the location of the corresponding term occurrence in the CS.

***Definition** 2:* We consider the following restrictions on the attacker's behavior in a protocol run.

1) In a run when a regular agent receives a block T in a message, T must have appeared as a block in some message that has already been sent earlier by some regular agent.
2) In any message Msg received in a regular role instance, if T is the block in Msg with location L, then T must be a block with the same location L in some message sent earlier by a regular role instance.

For an attack (a run of the protocol) that violates a freshness goal, if the attacker's behavior is restricted by

- both restrictions 1 and 2, the attack is called a ***direct replay attack*** (DRA).
- only restriction 1, the attack is called a ***restricted replay attack*** (RRA).
- no restrictions, the attack is called a ***general replay attack*** (GRA).

It is not obvious how to formally describe the three replay attacks defined above using the taxonomy discussed in [21].

D. Bounding the Number of Role Instances in a Run

We consider the ***number of role instances in a run***, call it ***NRI*** for short, for different problem settings of checking freshness. Note that every run has a finite number of role instances. NRI has been used to analyze the complexity of checking secrecy in the literature [3] [6] [4] [7] [8]. We clarify different notions of bounding NRI, depending on different settings of the inputs to the algorithm, as follows.

- We say NRI is ***bounded by an individual number***, or simply, is ***individually bounded*** if the problem to be decided is a tuple $\langle Pro, D, R, V, N\rangle$ where Pro is the protocol, whose size is measured by the size of its CS, and the freshness goal of variable V to the role R needs to be checked, and N is a natural number representing the bound on NRI. Only the runs with the number of role instances no more than N are considered. Note that for different problem instances N could be different.
- We say NRI is ***bounded by a fixed number***, or simply, is ***fixed***, if the problem to be decided is $\langle Pro, D, R, V, n\rangle$, for some fixed number n and all of the instances of the problem share the same n.
- We say NRI is ***unbounded***, if the problem to be decided is just $\langle Pro, D, R, V\rangle$, where there is neither fixed nor individual bound considered on NRI, and there could be any finite number of role instances in a run.

The advantage of clarifying these different settings of bounds is to avoid possible confusion in understanding the terms for bounds including bounded, fixed, unbounded, finite and infinite that appear in the literature. Although in practice the restriction 1 or 2, which is described in Definition 1, is valid only with some specific D and Pro, when we check DRA or RRA we have already assumed the restriction 1 or 2 on the attacker's behavior, without considering how the restriction is implemented; therefore, the attacker's initial knowledge pattern D is irrelevant to check DRA or RRA.

III. Complexity Results on Checking Freshness

A. Reductions Between Freshness and Other Problems

Based on our works in [8] and [9], which improves the work of [4], of proving undecidability results by direct reductions from the well-known reachability problem of 2-counter machines, we think this direct approach is convenient to prove the undecidability results of checking freshness. Comparing to the approach of reductions from other security goals such as secrecy, we think this direct approach of proving undecidability may be easier for readers to verify, especially when the ideal proof of checking secrecy for the corresponding setting is not obvious or not easily available; therefore, the proof of Theorem 1, which is on checking RRA, is by this direct reduction.

One approach of studying the complexity of checking freshness is to reduce the problem of checking secrecy, which is another problem of checking cryptographic protocols, to the problem of checking freshness, or the other way, since there are published results of the complexity of checking secrecy (but not much for authentication).

This approach requires that the convincing proofs of the corresponding complexity results of checking secrecy are available.

Reduction from checking secrecy to checking freshness can be established, and the idea is demonstrated in the proof of Theorem 2, which is on checking GRA. Such a reduction may be useful to show that checking freshness is undecidable when the number of role instances (NRI) is unbounded, and NP-hard when NRI is fixed, since we believe checking secrecy has the same complexity results correspondingly. We present two different proofs for Theorem 2, one by a direct reduction from 2-counter machine, and the other by a reduction from secrecy. The reduction scheme from secrecy to freshness works for proving the undecidability of checking GRA when NRI is unbounded, but not for RRA, since the attacker have to construct some blocks in the attack, which is another reason why Theorem 1 is proved by a reduction directly from 2-counter machines. The NP-hardness proof of checking secrecy we have noticed is provided by [5], which is by a reduction from 3-SAT. However, that proof assumes protocols as non-matching roles instead of communication sequences, and the secret term is declared in a non-realistic way, which are unconvincing aspects as discussed in [22] and [9]; hence, this paper we prove the NP-hardness of checking freshness in Theorem 5 by a direct reduction from 3-SAT while considering realistic protocols, not by a reduction from secrecy.

Reduction from freshness to secrecy can also be established, as demonstrated below in Section III, although it may not as obvious as the reduction in the other direction. Such a reduction may show that checking freshness is NP when NRI is fixed since we believe checking secrecy is NP with fixed NRI. The complexity result that checking secrecy is in NP has been addressed in [5], where the setting is that the number of sessions are bounded, which is similar to, but different from bounding the number of role instances. However, besides the significant contributions made by [5], we think there is an error in the proof of [5] to show checking secrecy is NP. More specifically it seems the error is due to an assumption in the proof of Theorem 1 in [5] that the DAG size of an instance of a term is no less than the DAG size of the term, which is incorrect. Study notes of the modeling, proofs, and the error of [5], and our fix of the error, are presented in details in [23]. It seems that the problem similarly exists in another related paper [24] written by the authors of [5], which proves a more general NP complexity of checking secrecy when XOR operators are considered. We consider that the NP result of checking secrecy while bounding the number of sessions does not imply, or is implied by, the NP result of checking secrecy while bounding the number of sessions. In [3] there is an NP-complete result of checking secrecy when the number of role instances is bounded. But that proof in [3] assumes a bound on message size, and encryption keys must be atomic. Bounding the message size makes the NP proof easier, and less general. We do not want to bound the message size. Our reduction from freshness to secrecy, presented below in Section III-B, needs composite keys. By the above reasons, in this chapter, we prove the NP result of checking freshness by directly analyzing a model checker, not by a reduction from secrecy. By the above reasons, in this paper, we prove the NP result of checking freshness by directly analyzing a model checker, not by reduction from secrecy.

B. Reduction from Freshness to Secrecy

We demonstrate a reduction scheme for freshness to secrecy. Although we do not use this reduction to obtain complexity results, such as the results of checking GRA, in this paper, it can be used in subsequent researches to do so after we are convinced with the corresponding complexity results of checking secrecy.

***Lemma** 1:* Given a problem of checking freshness a term X to a role R for a protocol Pro, with a certain attacker's initial knowledge pattern D, with a bound N on NRI, denoted as $\langle Pro, D, X, R, N\rangle$, the freshness problem can be reduced in polynomial time to a secrecy problem, which is denoted as $\langle Pro', D', S', N'\rangle$, where S' is the specification of secret terms (other fields have been similarly explained for the freshness problem), such that the freshness goal of X to R for Pro is violated by an attack with no more than N role instances, if and only if, the a secret term X' can be known to $\mathcal{I}$ by an attack to Pro' with no more than N' role instances.

Proof: We construct the secrecy problem according to the freshness problem as follows. A protocol Pro defines the initial knowledge patterns of regular agents. In the constructed protocol Pro', the initial knowledge of every regular agent is the same as in Pro, except that for every two agents, say P and Q, they share a symmetric key $k'_{P:Q}$, which is same as $k'_{Q:P}$ and is only known to P and Q. We require that $k'_{P:Q}$ does not appear in Pro. Even if Pro also requires that two agents P and Q use some shared symmetric key(s) between P and Q, then Pro' will allow P and Q shall one more key, which is $k'_{P:Q}$.

D' is the same as D, and especially D' does not allow $\mathcal{I}$ to know $k'_{P:Q}$ for $P \neq \mathcal{I}$ and $Q \neq \mathcal{I}$.

We explain some reduction idea first. One reduction approach is that when the freshness goal is violated, a special term, say T can be generated and T is used to trigger the secrecy failure. T should describe or imply that the same X is received in two different role instances of R. One difficulty is how to express two different role instances. If we require a role to explicitly check the disequality between two nonces, which are used to identify two different role instances, then the reduction may be less convincing since disequality tests of two nonces are rare among cryptographic protocols. The idea is to express the disequality of two nonces in a role that generates them, since it is guaranteed that two nonces generated in a role instance is guaranteed to be mutually different.

The communication sequence of Pro' includes the one of Pro but with more communication steps as follows.

First, choose two agent names that do not appear in Pro, without generality say A and B, and append the following two communication steps between A and B to the communication sequence of Pro. Actually as long as the relative order between the two communication steps are kept, where they appear in the communication sequence of Pro' does not matter. By doing so we add two roles, A's role and B's role, into Pro.

$$\begin{aligned} \#_A(Secret)(A \Rightarrow B): &\quad \{1, A, B, Secret, T\}^{\leftrightarrow}_{k'_{A:B}} \\ (B \Rightarrow A): &\quad \{2, B, A, Secret\}^{\leftrightarrow}_{T} \end{aligned}$$

The secrecy declaration S' states that a secrecy goal is to protect any instance of the term $Secret$ which is generated by A for B, when B and A are instantiated by some regular agents. Two constants are chosen such that they do not appear in Pro, and without loss of generality we say they are 1 and 2. The special term T represents a term which is critical for this reduction and we will reveal what T represents soon. The reduction is designed in a way such that $\mathcal{I}$ knows T if and only if the freshness goal of X to R in Pro is violated, and the attacker $\mathcal{I}$ can know a term $Secret$ if and only if the attacker can know T.

Second, choose two different agents, say F and G, that do not appear in Pro, and add the following communication sequence between F and G to the one of Pro, where 3, 4, and 5 are three constants that do not appear in Pro.

$$\begin{aligned} \#_F(N^1_F)(F \Rightarrow G): &\quad F, G, \{3, N^1_F\}^{\leftrightarrow}_{k'_{F:G}} \\ \#_G(N^1_G, Y)(G \Rightarrow F): &\quad G, F, \{4, N^1_F, Y, N^1_G\}^{\leftrightarrow}_{k'_{F:G}} \\ \#_F(N^2_F)(F \Rightarrow G): &\quad F, G, \{3, N^2_F\}^{\leftrightarrow}_{k'_{F:G}} \\ \#_G(N^2_G)(G \Rightarrow F): &\quad G, F, \{4, N^2_F, Y, N^2_G\}^{\leftrightarrow}_{k'_{F:G}} \\ (F \Rightarrow G): &\quad F, G, \{5, N^1_G, N^2_G\}^{\leftrightarrow}_{k'_{F:G}} \end{aligned}$$

Note that in F's role that is parsed from the above communication sequence, i.e., from F's point of view of the above 5 messages, it is guaranteed that N^1_F and N^2_F are mutually different since they are two nonces generated by F in F's role, and it is implicit that the two occurrences of Y are instantiated by the same value, but there is no guarantee that $N^1_G \neq N^2_G$, since we do not introduce the disequality check, which is rare in cryptographic protocols, between two received nonces into the protocol.

Third, for the freshness goal in Pro, which is the freshness of term X to role R, suppose R is executed by Q, i.e., R is Q's role. Choose an agent name that does not appear in Pro, say P, and then append the following two communication steps to the communication sequence of Pro.

$$\begin{aligned} \#_P(N_P)(P \Rightarrow Q): &\quad P, Q, \{3, N_P\}^{\leftrightarrow}_{k'_{P:Q}} \\ (Q \Rightarrow P): &\quad Q, P, \{4, N_P, X, e\}^{\leftrightarrow}_{k'_{P:Q}} \end{aligned}$$

Now the Q's role in Pro' has been extended with two more action steps comparing with Pro. Note that in the second message a special constant e is included in the encryption. We choose a constant that does not appear in Pro, call it e, and in Pro' e is initially known to all regular agents. Note that e is different from any nonce that is generated in a run of Pro'.

Now we replace T with $\{5, e, e\}^{\leftrightarrow}_{k'_{A:B}}$.

Suppose the freshness goal is violated for Pro by an attack, call it $attack$, where are two role instances r_1 and r_2 of the role R, such that X is instantiated by the same term x in both r_1 and r_2. Now we construct an attack, call it $attack'$, to Pro' such that a secret term is leaked, as follows.

$attack'$ is the same as $attack$, except that in $attack'$, the two role instances r_1 and r_2 has two more action steps to execute, and there are three more role instances in $attack'$: r_3 of A's role, r_4 of B's role, and r_5 of F's role.

After executing the events that are in $attack$, $attack'$ continues executing the extra events as follows, and the events are represented by their action steps. Let r_1 be executed by a. If the agent who executes r_2 is different from

a, then agent b is chose as the executor of r_2, otherwise if r_2 and r_1 are both executed by a, then we chose b as any regular agent other than a.

r_5 is executed by a, and the action steps of r_5 are the follows.

$$\begin{aligned} \#_a(n_a^1) + (a \Rightarrow b) :&\quad a, b, \{3, n_a^1\}^{\leftrightarrow}_{k'_{a:b}} \\ -(b \Rightarrow a) :&\quad b, a, \{4, n_a^1, x, e\}^{\leftrightarrow}_{k'_{F:G}} \\ \#_a(n_a^2) + (a \Rightarrow b) :&\quad a, b, \{3, n_a^2\}^{\leftrightarrow}_{k'_{a:b}} \\ -(b \Rightarrow a) :&\quad b, a, \{4, n_a^2, x, e\}^{\leftrightarrow}_{k'_{a:b}} \\ +(a \Rightarrow b) :&\quad a, b, \{5, e, e\}^{\leftrightarrow}_{k'_{a:b}} \end{aligned}$$

The last two action steps of r_1, which is executed by a, are the follows.

$$\begin{aligned} -(b \Rightarrow a) :&\quad b, a, \{3, n_a^1\}^{\leftrightarrow}_{k'_{a:b}} \\ +(a \Rightarrow b) :&\quad a, b, \{4, n_a^1, x, e\}^{\rightarrow}_{k'_{a:b}} \end{aligned}$$

The last two action steps of r_2, if it is executed by a, are the follows.

$$\begin{aligned} -(b \Rightarrow a) :&\quad b, a, \{3, n_a^2\}^{\leftrightarrow}_{k'_{a:b}} \\ +(a \Rightarrow b) :&\quad a, b, \{4, n_a^1, x, e\}^{\rightarrow}_{k'_{a:b}} \end{aligned}$$

The last two action steps of r_2, if it is executed by b, are the follows.

$$\begin{aligned} -(a \Rightarrow b) :&\quad a, b, \{3, n_a^2\}^{\leftrightarrow}_{k'_{a:b}} \\ +(b \Rightarrow a) :&\quad b, a, \{4, n_a^1, x, e\}^{\rightarrow}_{k'_{P:Q}} \end{aligned}$$

Remind that X is instantiated by x in both r_1 and r_2, and $k'_{a:b}$ is the same as $k'_{b:a}$. We use msg_5^1 to represent the first message of the role instance r_5, where the super-script is the message number and the lower-script is the ID of the role instance. Especially, msg_1^i or msg_2^i means the i^{th} extra message of r_1 or r_2, for $i \in \{1, 2\}$. After r_5 sends msg_5^1, $\mathcal{I}$ constructs msg_1 and lets r_1 receives it. After r_1 sends msg_1^2, $\mathcal{I}$ constructs msg_5^2 and let r_5 receives it. After r_5 sends msg_5^3, $\mathcal{I}$ constructs msg_2^1 and lets r_2 receives it. After r_2 sends msg_2^2, $\mathcal{I}$ constructs msg_5^4 and lets r_5 receives it. Then r_5 sends msg_5^5 and $\mathcal{I}$ obtains $T = \{5, e, e\}^{\leftrightarrow}_{k'_{a:b}}$.

Then r_3 and r_4 start to execute, and they are executed by a and b respectively. The action steps of r_3 and r_4 are interleaved according to the following communication sequence between them.

$$\begin{aligned} \#_a(secret)(a \Rightarrow b) :&\quad \{1, a, b, secret, \{5, e, e\}^{\leftrightarrow}_{k'_{a:b}}\}^{\leftrightarrow}_{k'_{a:b}} \\ (b \Rightarrow a) :&\quad \{2, b, a, secret\}^{\leftrightarrow}_{\{5,e,e\}^{\leftrightarrow}_{k'_{a:b}}} \end{aligned}$$

It is obvious that before a message is received by a role instance, the attacker can obtain the message. Since the attacker knows $\{5, e, e\}^{\leftrightarrow}_{k'_{a:b}}$, it is obvious that the attacker knows the term $secret$ after msg_4^2 is sent.

We can let $N' = N + 3$, since for the secrecy attack of Pro' besides the role instances that are related to let two role instances of R receive the same X, there will be 3 extra role instances, one for A's role, one for B's role and one for F's role. Alternatively, if we choose to include the added communication steps into the existing roles of Pro, and no extra agents are introduced in Pro', then $N' = N$.

The attacker can know an instance of the term $Secret$ only if T is leaked. The only way that the attacker can know T is that T is generated by the last message of a role instance, call it r_F, of F's role. Then in the two messages received by r_F there must be two encryptions T_1 and T_2, where T_1 has the form $\{4, N_F^1, Y, e\}^{\leftrightarrow}_{k'_{F:G}}$ and T_2 has the form $\{4, N_F^2, Y, e\}^{\leftrightarrow}_{k'_{F:G}}$. T_1 and T_2 must be sent by the second message of two role instances, call them r_Q^1 and r_Q^2, of Q's role. Note that T_1 and T_2 cannot be sent in a role instance of G's role, since e cannot be a nonce freshly generated. r_Q^1 and r_Q^2 are two different role instances since $N_F^1 \neq N_F^2$. In r_1 and r_2 the same Y is received which instantiates X. So the freshness goal is violated, and the same attack can be applied to Pro. A fully formal reasoning of this direction can be established with more technical details. For this paper we think the proof of this reduction is sufficient enough, especially because we do not use this reduction to obtain complexity results.

■

Note that the above reduction from freshness to secrecy can consider the attacker as an insider and the protocols as communication sequences. Reductions consider only an outsider attacker and protocols as non-matching roles, which are unrealistic, have been considered less convincing, as disscussed in [22] and [9].

C. Undecidability results

We present the definition of 2-counter machine before the undecidability proofs.

Definition 3: A ***deterministic 2-counter machine*** [25] with empty input is a pair $\langle Q, \delta \rangle$, where Q is a set of states including the starting state q_0 and the accepting state q_{final} and δ is a set of transition rules. A configuration of a 2-counter machine is a tuple (q, V_1, V_2), where q is the current state and V_1 and V_2 are two non-negative integers representing the two counters. The 2-counter machine can detect whether a counter is 0 or not. A transition rule, (call the rule $T \in \delta$) is of the form $[q, i_1, i_2] \rightarrow [q', j_1, j_2]$, where $q, q' \in Q$; $i_1, i_2 \in \{0, 1\}$; $j_1, j_2 \in \{-1, 0, +1\}$. An application of T can be described as $(q, V_1, V_2) \longrightarrow^T (q', V_1', V_2')$, where LHS and RHS are the configuration before and after the transition respectively. For $h \in 1, 2$, when $i_h = 0$, it means that $V_h = 0$. When $i_h = 1$, it means that $V_h > 0$. When $j_h = +1$ ($j_h = 0$, $j_h = -1$), it means that after the transition, $V_h' = V_h + 1$ ($V_h' = V_h$, $V_h' = V_h - 1$). Especially, when $j_h = -1$, i_h must be 1, since decrementing 0 is not allowed. The reachability problem of such a 2-counter machine is to decide that, starting from the initial configuration $(q_0, 0, 0)$, after applying some applicable transition rules, whether some final configuration $(q_{final}, \_, \_)$ can be reached, where $\_$ represents an arbitrary possible value. We assume (for convenience) that $q_0 \neq q_{final}$ and, for nontriviality, that δ is not empty.

It is obvious that a 2-counter machine allowing $q_0 = q_{final}$ can be equivalently simulated by a 2-counter machine defined above, and the reachability problem of 2-counter machines defined above is undecidable.

Theorem 1: Checking RRA for a freshness goal is undecidable when the number of role instances in a run (NRI) is unbounded.

Proof: We reduce the reachability problem of a deterministic 2-counter machine to a problem of checking RRA for a freshness goal of a protocol. We have used this approach similarly in [9]. In this reduction the attacker cannot construct any blocks that can be received by a regular agent since these blocks must be encrypted by a symmetric key K which is unknown to the attacker. Reachable configurations of a 2-counter machine M are encoded by special terms called configuration terms in a protocol run. The reduction ensures that a certain freshness goal is violated in a run if and only if a configuration term T is known to $\mathcal{I}$, such that T encodes a final configuration reachable to M.

Given a 2-counter machine $M = \langle Q, \delta \rangle$, let $Q = \{q_0,\ q_{final},\ q_1,\ q_2,\ \cdots,\ q_m\}$ and $\delta = \{T_1,\ T_2,\ \cdots,\ T_n\}$.

Encoding is a correspondence between a term and its designed meaning in a reduction. A ***configuration term*** has the form $\{5, X,\ q, C_1, C_2\}_K^{\leftrightarrow}$. C_1 and C_2 and X could be any terms. q is a constant that could be any state in Q. The number 5 is used for configuration terms to distinguish them from connection terms (introduced later) where the number 7 is used. K is a symmetric key that is not known to the attacker but known to all of the regular agents. Using K makes restriction of the attacker's behavior defined by RRA obviously valid in the reduction, i.e., if the attacker can violate a freshness goal of the constructed protocol then the attack must be a RRA. Note that $\mathcal{I}$ cannot construct a configuration term since $\mathcal{I}$ does not know K. A ***connection term*** has the form of $\{7, C_1, C_2\}_K^{\leftrightarrow}$. Connection terms are used to build the encoding of a counter value (described later).

A protocol Pro is constructed according to M. Pro requires the initial knowledge pattern of agents as follows.

$$\forall P, P \in AN : P.init = AN \cup Q \cup \{z, 1, 2, 3, 5, 7, K\}$$

The set of roles of Pro is $\{S_0, R_0, S_1, R_1, \cdots, S_n, R_n, S_{final}, R_{final}\}$. Note that there are n transition rules in δ.

We describe a matching pair of roles S_i and R_i, $i \in \{0, 1, \cdots n, final\}$ by the CS between them. The action steps of two matching roles can be straightforwardly parsed from the CS between them. We choose different variable names in different matching pairs of roles so there is no confusion when the CSes of these pairs of roles are concatenated to form the CS of the whole protocol.

Two roles S_0 and R_0, executed by A and B respectively, are designed to generate the initial configuration term. The CS between them is the follows.

1. $\#_A(N_A)\ (A \Rightarrow B):\quad A, B, \{5, N_A, q_0, z, z\}_K^{\leftrightarrow}$
2. $\#_B(N_B)\ (B \Rightarrow A):\quad B, A, N_B, \{q_{final}, N_A, N_B\}_K^{\leftrightarrow}$

The **target problem** of the reduction is to check the freshness of N_B for S_0.

The two roles S_{final} and R_{final} are executed by A_{final} and B_{final} respectively. The CS between A_{final} and B_{final} is the follows.

1. $\#_{A_{final}}(N_X, N_Y, C1_{final}, C2_{final})\ (A_{final} \Rightarrow B_{final})$:
   $A_{final}, B_{final}, N_Y, \{5, N_X, q_{final}, C1_{final}, C2_{final}\}_K^{\leftrightarrow}$
2. $(B_{final} \Rightarrow A_{final}) : B_{final},\ A_{final},\ \{q_{final}, N_X, N_Y\}_K^{\leftrightarrow}$

For each $T_f \in \delta$, for some f, $1 \leq f \leq n$, suppose $T_f = [q, i_1, i_2] \rightarrow [q', j_1, j_2]$. T_f corresponds to two roles S_f and R_f (starter and responder), executed by agents A_f and B_f respectively. The general template of the CS between S_f and R_f is the following. The exact CS between A_f and B_f will be formed after a set of rewrite rules (described later) are applied to this general template.

1. $\#_{A_f}(C1_f, C2_f, C1_f^{-1}, C2_f^{-1}, N_f) \quad (A_f \Rightarrow B_f):$
   $A_f, B_f, \{5, N_f, q, C1_f, C2_f\}_K^{\leftrightarrow},\ \{7, C1_f^{-1}, C1_f\}_K^{\leftrightarrow}, \{7, C2_f^{-1}, C2_f\}_K^{\leftrightarrow}$
2. $\#_{B_f}(C1_f^{+1}, C2_f^{+1}) \quad (B_f \Rightarrow A_f):$
   $B_f, A_f, \{5, N_f, q', C1'_f, C2'_f\}_K^{\leftrightarrow},\ \{7, C1_f, C1_f^{+1}\}_K^{\leftrightarrow},\ \{7, C2_f, C2_f^{+1}\}_K^{\leftrightarrow}$

In the first message A_f will choose B_f as the interlocutor, $B_f \in AN$ and $B_f \neq A_f$. Note that B_f will carry the nonce N_f received in the first message to the second message. The variables Ch'_f, $h \in \{1, 2\}$, will not appear in the actual CS between S_f and R_f since they will be replaced by Ch_f or Ch_f^{-1} or Ch_f^{+1}, after applying the rewrite rules.

For $h \in \{1, 2\}$, the following rewrite rules, each is described as "condition $\mapsto$ effects", will be applied as much as possible to the general template between S_f and R_f, according to the conditions satisfied by the transition rule T_f of M, $1 \leq f \leq n$. $W \rightarrowtail V$ means to replace W with V in the above template. $W \rightarrowtail \varepsilon$ means to remove W. An *implicit rule* is that any term that is not removed or changed will still appear in the CS between S_f and R_f. Especially, if every term in $terms$ of an action of $\#_{agentName}(terms)$ is removed, then this whole fresh term generation action is removed. Note that when a rule is applied, the term removing tasks in RHS are arranged following the order from left to right, so that the smaller terms are removed later and the bigger terms containing the smaller terms are removed earlier, to avoid possible confusion.

1. $i_h = 0 \quad \mapsto \quad Ch_f \rightarrowtail z;\ \{7, Ch_f^{-1}, Ch_f\}_K^{\leftrightarrow} \rightarrowtail \varepsilon$
2. $i_h = 1 \quad \mapsto \quad \{7, Ch_f^{-1}, Ch_f\}_K^{\leftrightarrow} \in Msg_1$
3. $j_h = +1 \quad \mapsto \quad Ch'_f \rightarrowtail Ch_f^{+1}$
4. $j_h = 0 \quad \mapsto \quad Ch'_f \rightarrowtail Ch_f;\ \ \{7, Ch_f, Ch_f^{+1}\}_K^{\leftrightarrow} \rightarrowtail \varepsilon;\ Ch_f^{+1} \rightarrowtail \varepsilon$
5. $j_h = -1 \quad \mapsto \quad Ch'_f \rightarrowtail Ch_f^{-1};\ \ \{7, Ch_f, Ch_f^{+1}\}_K^{\leftrightarrow} \rightarrowtail \varepsilon;\ Ch_f^{+1} \rightarrowtail \varepsilon$

The counter value 0 must be represented by z. When a counter h should be positive, the term $\{7, Ch_f^{-1}, Ch_f\}_K^{\leftrightarrow}$ is needed in Msg_1, which shows that Ch_f^{-1} encodes a number one less than the number encoded by Ch_f.

If M can reach a final configuration $(q_{final}, \_, \_)$ starting from $(q_0, 0, 0)$ by some finite computation $Comp$, then $Comp$ can be represented as a finite sequence of configurations connected by applicable rules in δ, as follows.

$$(q_0, 0, 0) \longrightarrow^{t_1} (Q^1, V_1^1, V_2^1) \cdots\ (Q^{u-1}, V_1^{u-1}, V_2^{u-1}) \longrightarrow^{t_u} (Q^u, V_1^u, V_2^u)$$

Here $u > 0$, $t_1, \cdots, t_u \in \delta$.

For a certain time t, let E be the set of events that has been executed in the run before t. we say a term X is the ***encoding*** of a positive integer N at t if and only if there is a sequence of terms :

$$\{7, z, X_1\}_K^{\leftrightarrow}, \{7, X_1, X_2\}_K^{\leftrightarrow}, \cdots, \{7, X_{N-2}, X_{N-1}\}_K^{\leftrightarrow}, \{7, X_{N-1}, X\}_K^{\leftrightarrow}$$

such that the attacker knows each element T of this sequence ($T \in know_{\mathcal{I}}(E)$). Here X and X_j, for some integer j, $1 \leq j \leq N-1$, are different variables that can represent any terms (could be composite terms). We call N the ***i_value*** of X (i stands for integer), or X is the ***encoding*** of N, or X ***encodes*** N, denoted as $N = \underline{X}$. The above term sequence is called the ***encoding sequence*** of X. The encoding sequence of z is z.

Here is the idea of the proof. Suppose the freshness goal of N_B to S_0, where S_0 is A's role, is violated, then there are two role instances r and r' of S_0 where a term, say n_b, is received, which instantiates N_B in both r and r'. In r and r', N_A must be instantiated by some freshly generated nonces. Let N_A be instantiated by n_a in r, and by n'_a in r', where n_a and n'_a are freshly generated in r and r' respectively. Then in the second message of r, a block $\{q_{final}, n_a, n_b\}_K^{\leftrightarrow}$ must be received. In order to let the role instance r' of S_0 receive the same n_b, $\mathcal{I}$ has to provide a term $\{q_{final}, n'_a, n_b\}_K^{\leftrightarrow}$, where n'_a is the nonce freshly generated by r'. There is only one way the attacker $\mathcal{I}$ can provide this term. $\mathcal{I}$ can provide n_b as the term N_Y to a role instance r_{final} of R_{final} and then in the second message of r_{final} a term of the form $\{q_{final}, N_X, n_b\}_K^{\leftrightarrow}$ will be produced. However N_X must be n'_a. The only way to let N_X be n'_a is to obtain a final configuration term of the form $\{5, n'_a, q_{final}, C_1, C_2\}_K^{\leftrightarrow}$, which is a block received in the first message of r_{final}. In order to do that, $\mathcal{I}$ has to obtain a sequence of configuration terms that contain n'_a, starting from $\{5, n'_a, q_0, z, z\}_K^{\leftrightarrow}$, which is obviously provided by the first message of r'. Note that n'_a is carried from configuration term to configuration term. The protocol run corresponds to a computation of M. $\mathcal{I}$ can obtain $\{5, n'_a, q_{final}, C_1, C_2\}_K^{\leftrightarrow}$ and then let r' receive n_b, and thus violates the freshness goal of N_B

for S_0, if and only if the 2-counter machine M can reach some final configuration in some computation. The two directional proof (for if and only if) can be done similarly to the proofs in [9] and [8].

■

***Theorem** 2:* Checking GRA for a freshness goal is undecidable when the number of role instances in a run (NRI) is unbounded.

Proof: There are two ways to prove this theorem. First, we can do reduction from the reachability problem of a deterministic 2-counter machine. The protocol used in the reduction is the same as the one used in [9], which has been specially designed so that the attacker $\mathcal{I}$ is an insider and $\mathcal{I}$ has to construct blocks in order to commit an attack. The reduction use the same protocol, which is translated from a 2-counter machine M, as in [17] [9]. In the protocol, the two roles S_{final} and R_{final} carry out an adjusted version of the public key Needham-Schroeder protocol, they are executed by agents A and B respectively. The communication sequence between A and B is follows.

1. $\#_A(N_A, C1_{final}, C2_{final})\ (A \Rightarrow B):$
   $\{\ 1, N_A, A, \{5, B, e, q_{final}, C1_{final}, C2_{final}\}_{\overrightarrow{k_A^0}}\ \}_{\overrightarrow{k_B^1}}$
2. $\#_B(N_B)\quad (B \Rightarrow A) : \{2, N_A, N_B\}_{\overrightarrow{k_A^1}}$
3. $\qquad\qquad (A \Rightarrow B) : \{3, N_B\}_{\overrightarrow{k_B^1}}$

The freshness goal chosen for the reduction is the one of the term $C1_{final}$ or $C2_{final}$ to the role R_{final} executed by agent B. Following the proof in [17], the attacker can know a final configuration term, which has the form $\{5$, B, e, q_{final}, $C1_{final}$, $C2_{final}\ \}_{\overrightarrow{k_A^0}}$, if and only if M can reach a final configuration.

It is obvious that if the attacker can obtain a final configuration term, the attacker can use the same final configuration term to construct the first messages that is received in two role instances r_1 and r_2 of R_{final}, and the freshness goal is violated. Let N_A be instantiated by n_a in r_1, and by n'_a in r_2. The attacker knows n_a and n'_a, which are sent from some agent A to $\mathcal{I}$ when A executes the role S_{final} and $\mathcal{I}$ is an insider initially known to A. $\mathcal{I}$ initially knows 1, A, and k_B^1. Therefore, once $\mathcal{I}$ knows a final configuration term, $\mathcal{I}$ can construct the the first messages that are received in two role instances r_1 and r_2, and the same $C1_{final}$ or $C2_{final}$ is received in both r_1 and r_2. The attack is very similar to the attack to the one described in [17], which simulates the attack to the PKNS protocol [26].

In the other direction, if the freshness goal is violated in a run, then two different role instances r_1 and r_2 of the role R_{final} will receive a subterm, which is a final configuration term, in their first messages, with the same term $C1_{final}$ or $C2_{final}$. One of these two first messages of r_1 and r_2 must be constructed by the attacker, since otherwise the values of $C1_{final}$ or $C2_{final}$ will be freshly generated by regular agents and will be different in r_1 and r_2; therefore, $\mathcal{I}$ must know a final configuration term, which implies that M can reach a final configuration.

The second way to prove this theorem is by reduction from secrecy to freshness, as sketched below. Given a protocol Pro of the secrecy problem, we construct a protocol Pro' for the freshness problem by adding the following lines in some proper way into the CS of Pro.

$$\begin{array}{lll} \#_A(N_1) & (A \Rightarrow B): & \{m, A, B, N_1, SECRET\}_{\overrightarrow{k_B^1}} \\ \#_B(N_2) & (B \Rightarrow A): & \{m+1, B, A, N_1, N_2\}_{\overleftrightarrow{SECRET}} \\ & (A \Rightarrow B): & \{m+2, A, B, N_2\}_{\overleftrightarrow{SECRET}} \end{array}$$

Here N_1 and N_2 are fresh nonces created by A and B respectively. $SECRET$ is a term that is supposed to be shared between A and B. The three numbers (constants) m, $m+1$ and $m+2$, for some integer m, are used to distinguish the three messages from other encryptions appearing in the protocol to avoid confusion. Then the freshness goal to be checked is the one of N_1 to B's role, assuming the attacker initially knows the names and public keys of all agents and all the constants in the protocol. ■

D. Athena

We obtain several decidability results based on analyzing the performance of the model checker Athena [12] [13]. We introduce the notions and algorithm of Athena first, which are adapted for the modeling of this paper. We arrange the presentation and the proof of Athena differently for simplicity and clarity.

A ***strand*** is a sequence of action steps formed by instantiating a role by some substitution. There are two differences between a strand and a role instance. First in a strand variables can appear, while in a role instance all terms are ground. Second a role instance includes events, where action steps are associated with $time$ fields, but strand includes only action steps. During the reasoning of the model checker a strand represents a role instance of a run.

Every strand is associated with a unique identifier in the reasoning. A ***node*** of a strand is a pair $\langle r, L\rangle$ where r is the unique identifier of the strand, and L is the location of a block in a message of the strand, which has been introduced in Section II. Each node can be used as a unique identifier of a block occurrence. The ***term of node*** nd, for $nd = \langle r, L\rangle$, denoted as $term(nd)$, means the term (the block) appearing at location of L of the strand r. A node in a message received in a strand is called a ***negative node***, otherwise a node in a message sent in a strand is called a ***positive node***. A ***state*** is a tuple $\langle strands, binding, counter\rangle$, where $strands$ is a set of strands, $binding$ is a binary relationship mapping from one negative node in $strands$ to a positive node in $strands$, and $counter$ is a natural number corresponding to the number of strands in a state. The notation $\langle r, L\rangle \twoheadrightarrow \langle r', L'\rangle$ means that the negative node $\langle r, L\rangle$, which is called the ***goal***, binds to the positive node $\langle r', L'\rangle$, which is called the ***binder***, where $r, r' \in strands$. $counter$ is used to check if the bounds on NRI is satisfied in a state or not. $counter$ can also be used to name a new strand that is introduced into a state and to name the variables of the new strand.

The ***causally precedence*** $\lesssim$, also called ***causally earlier*** relationship between two nodes $nd = \langle r, L\rangle$ and $nd' = \langle r', L'\rangle$ is defined as follows.

- if $r == r'$, i.e., they refer to the same strand, if nd appears in a message with message number m, and nd' appears in a message with number m', and $m' < m$, then $nd' \lesssim nd$.
- if $nd \twoheadrightarrow nd'$ then $nd' \lesssim nd$. Note that $\lesssim$ is opposite to $\twoheadrightarrow$.
- if $nd' \lesssim nd''$ and $nd'' \lesssim nd$, for some node nd'', then $nd' \lesssim nd$. This condition means that $\lesssim$ is transitive.

In [13] $\lesssim$ is defined to be reflexive, but not in this paper since it is not necessary. In a state, node nd' is not causally earlier than node nd is denoted as $nd' \not\lesssim nd$.

$\lesssim$ is obviously extended for action steps of the strands in a state. For two action steps stp and stp' appearing in a state S, $stp' \lesssim stp$ if one of the following conditions satisfies.

- If stp' and stp appear in the same strand with message number m' and m respectively and $m' < m$.
- If there is a node nd of stp and a node nd' of stp' such that $nd' \lesssim nd$.
- If $stp' \lesssim stp''$ and $stp'' \lesssim stp$, for some step stp'' in S.

The strand space model [19] describes a run as a bundle of strands, where each negative node is bound to a positive node. In [12] the author extended the strand space model of [19] for the model checker Athena by introducing variables and the unification mechanism and a set of new notions including semi-bundle and goal binding. A semi-bundle, which may have goals unbound, is expanded and updated during the computation of Athena and finally forms a bundle. We only use a subset of the notions used in [12] [13] that are enough for this paper, and in some aspects these notions are presented in a different perspective since we want to clarify the relationship between the notions of strand space used by Athena and our model of protocol run. The advantage of our presentation of Athena is that the meaning and the correctness of the algorithm can be understood more clearly. Based on the improved understanding, the algorithm is also simplified. For example we directly introduce the goal binding relation $\twoheadrightarrow$ for the model checker without using the $\rightarrow$ relationship, which is defined and used in Athena [12] [13], since we consider $\rightarrow$ is unnecessary or its meaning is unclear.

The semantics of goal binding can be explained more intuitively in our model. When checking RRA, note that the attacker's behavior of constructing blocks does not need to be considered due to the restriction, node nd_1 binds to node nd_2 means the follows: Let ev_1 and ev_2 be the two events that nd_1 and nd_2 belong to respectively. Then $term(nd_1)$ is sent by ev_2 as a block and $term(nd_1)$ cannot be sent at any event with its time point earlier than $ev_2.time$ in the run. This semantics of goal binding can be extended for GRA, when the attacker's internal computations need to be described using term derivations.

The model checker has to ensure three properties, call them ***correctness properties***, of a reachable state S during the reasoning.

1) The $\lesssim$ relationship of the action steps and nodes in S is acyclic.
2) All of the negative nodes with the same term in S must bind to the same positive node in S.
3) If a node nd' is the binder of nd, i.e., $nd \twoheadrightarrow nd' \in S.binding$, then there is *no* node nd'', which is different from nd' in S such that the $term(nd'') == term(nd') == term(nd)$ and $nd'' \lesssim nd'$.

The second correctness property is not introduced in [12] and [13], but we consider it can reduce redundant state exploration significantly in some situation. Even though there could be several nodes that possibly send the same term T at the earliest time, only one of these possible binders for T needs to be considered in a child state of S, and then let other children states of S to consider the other nodes as binders. Property 3 means that a goal should only bind to the (causally) earliest binder as described in [12] and [13].

In a state the variables are global across different strands, which means that if a variable X will be instantiated by Y, then all X appearing in all strands in the state will be replaced by Y. The function $unifiable(T, T')$ returns true, if T and T' are unifiable, otherwise false. Type information can be introduced for unification. For example if according to the protocol in a certain role a variable A is required to be some known agent name, then in a run A can only be unified with an agent name, and A cannot be unified with a nonce generated in the run. For two terms T and T', $mgu(T, T')$ represents the most general unifier (MGU) of T and T'. For a substitution γ, $\gamma(X)$ means to apply γ to X, where X could be a term, an action step, a strand, or a state, in the obvious way.

RRA_Checker(Pro, R, V, N), to check the freshness goal of a variable V to role R in a protocol Pro, no more than N role instances in a run, $N \geq 2$

```
1:  Let $r^1$ and $r^2$ be two different strands of $R$ formed as follows. $r^1$ and $r^2$ are the same as $R$ except for two
    aspects: First, for every variable $X$ in $R$ that is not freshly generated in $R$, except $V$ whose freshness needs to
    be checked, $X$ is renamed as $X^1$ in $r^1$, and as $X^2$ in $r^2$. The variable $V$ remains unchanged and appears in
    both $r^1$ and $r^2$. Second, for each variable $Y$ if $Y$ is freshly generated in $R$, $Y$ is renamed by a unique constant
    in $r^1$, and by another unique constant in $r^2$.
2:  Let $state^0 := \langle \{r^1, r^2\}, \emptyset, 2 \rangle$; Let $STATES := \{state^0\}$.
3:  while $STATES \neq \emptyset$ do
4:    let $S$ be an arbitrary state in $STATES$; $STATES := STATES - S$.
5:    if for every negative node $nd$ in $S$, there is some positive node $nd'$ in $S$ such that $nd \twoheadrightarrow nd' \in S.binding$
      then
6:      print BAD: an attack found, the freshness goal of $V$ for $R$ is violated.
7:      Quit the algorithm.
8:    end if
9:    Let $nd$ be an arbitrarily chosen negative node in $S$ such that $nd \twoheadrightarrow nd' \notin S.binding$ for any positive node
      $nd'$. It means $nd$ has not been bound (to any positive node) yet. Let $T := term(nd)$.
10:   for all positive node $nd'$ in $S.strands$ such that $nd \lnsim nd'$ do
11:     if $unifiable(\ T,\ term(nd')\ )$ then
12:       Let $\gamma := mgu(\ T,\ term(nd')\ )$.
13:       Let state $S'$ be a new state formed as
                 $\langle \gamma(S.strands),\ S.binding \cup \{nd \twoheadrightarrow nd'\},\ S.counter \rangle$.
14:       if $S'$ satisfies the three correctness properties as described earlier in this Section then
15:         Let $STATES := STATES + S'$. { /* Insert $S'$ into $STATES$. */}
16:       end if{ /* $S$ spawns $S'$ */ }
17:     end if
18:   end for{/* Finish trying to bind $nd$ to nodes of existing strands*/}
19:   if $S.counter < N$ then
20:     for all blocks $T'$ of the protocol, such that $T'$ appears at location $L'$ in a role $R'$ in a sent message
        numbered with $m$, and $unifiable(T, T')$ do
21:       Let $n = S.counter + 1$; Let $r^n$ be a new strand formed as follows. $r^n$ is the same as the prefix of the
          action steps of $R'$ up to the message numbered with $m$, denote this prefix as $R'^{\uparrow m}$, except that for each
          variable $X$ of $R'^{\uparrow m}$ if $X$ is not freshly generated in $R'$, $X$ is renamed with $X^n$ in $r^n$. Otherwise if $X$
          is freshly generated in $R'^{\uparrow m}$, then $X$ is replaced in $r^n$ by a unique constant that has not appeared in $S$
          yet.
22:       Let $T''$ be the block located at $L'$ in $r^n$; Let $\gamma := mgu(T, T'')$; Let $nd'$ be the node $\langle r^n, L' \rangle$
23:       let $S'$ be a new state formed as (note $S$ does not change)
                 $\langle \gamma(S.strands \cup \{r^n\}),\ S.binding \cup \{nd \twoheadrightarrow nd'\},\ n \rangle$
24:       if $S'$ satisfies the second correctness property, described earlier then
25:         insert $S'$ into $STATES$. {/* $S$ spawns $S'$ */}
26:       end if{/* No need to consider the correctness properties 1 and 3 */}
```

```
27:     end for{/* Finish trying to bind $nd$ to nodes of new strands. */}
28:   end if{/* Finish handling the state $S$ */}
29: end while
30: print Good: the freshness goal of $V$ for $R$ is satisfied.
```

More discussion of the algorithm is presented in Appendix C.

Lemma 2: When NRI is individually bounded, $RRA\_Checker$ terminates in 2-EXPTIME, and when NRI is fixed, $RRA\_Checker$ terminates in EXPTIME.

Proof: Terms appear as bit-strings to the actual algorithm. Note that a bit-string here does not mean the data in physical network. The length of a bit-string Str is the number of bits and is denoted as $|Str|_{bit}$. The input of $RRA\_Checker$, which is $\langle Pro, R, V, N\rangle$, can be considered as a bit-string and its size is measured as $\zeta = |\langle Pro, R, V, N\rangle|_{bit}$. We want to prove that when NRI is individually bounded by N, or NRI is fixed to n (N is replaced by n), $RRA\_Checker$ terminates with time cost $O(2^{(2^{\mathcal{P}(\zeta)})})$, or $O(2^{\mathcal{P}(\zeta)})$, respectively, where $\mathcal{P}(\zeta)$ is a polynomial function of ζ. Time is measured by the number of instructions executed.

First we consider the case that NRI is individually bounded by N. The states that can be reached in a computation of $RRA\_Checker$ can be viewed as a tree. The top state is s^0. If a state S' is created from a state S (by the line 15 or 25) then S' is a child state of S. Let $\psi = |Pro|_{bit} \times N$. It is obvious that $N < 2^{|N+1|_{bit}} = 2^{O(|N|_{bit})}$. The number of occurrences of subterms of a term T is no more than $|T|_{bit}$. Since each state can have at most N strands, and each strand can have no more than $|Pro|_{bit}$ nodes, the total number of nodes in a state is at most $|Pro|_{bit} \times N = \psi$. Each state has at most $O(\psi)$ children states, since for the single arbitrarily chosen negative node nd of S (see line 9), there are at most ψ positive nodes in the existing strands or new strands that can be the possible binders for nd. The depth from the top state s^0 to the bottom of the tree is at most ψ, since each child state has one more negative node bound (to some positive node) than its parent state, and there are at most ψ negative nodes in a state. So the tree of states is at most $O(\psi)$ branching and at most $O(\psi)$ deep. So the number reachable states is at most $O(\psi^{\psi})$.

For efficiency purpose of unification, all terms are represented as DAGs [5] in the reasoning of RRA_Checker. A DAG of a term T is a tree where each subterm of T appears as a node of the tree exactly once. The subterms of T is defined in the common way as in [5]. Note that encryption key is considered as a subterm of an encryption. The DAG size of a term T is the number of subterms of T, denoted as $|T|_{DAG}$. Obviously $|T|_{DAG} \leq |T|_{bit}$. All messages sent or received in the protocol Pro is translated into DAG representation, which can be done in $O(|Pro|_{bit})$ time. $|Pro|_{DAG}$ is defined accordingly. Let $\mathcal{M}$ be the number of all distinct subterms appearing in all messages sent or received in all strands in a state. Since for RRA a regular agent can only receive and accept a block that is sent in a message by a regular agent, obviously only the messages sent in the strands contributes to $\mathcal{M}$. It is obvious to prove that for a strand r of role R, r can contribute at most $|R|_{DAG}$ to $\mathcal{M}$, and $|R|_{DAG} \leq |Pro|_{DAG}$. Since there are at most N strands, for any reachable state $\mathcal{M} \leq |Pro|_{DAG} \times N < \psi$. So for any message Msg in a strand of a state, $|Msg|_{DAG} < \psi$.

The time cost to generate a new state is $O(\psi^3)$ for the following reasons: First, for two terms T_1 and T_2, $mgu(T_1, T_2)$ and $unifiable(T_1, T_2)$ (Line 12 and 21) have time cost $O(|T_1|_{DAG} + |T_2|_{DAG})$, and since $|T_1|_{DAG} < \psi$ and $|T_2|_{DAG} < \psi$, the time cost is $O(\psi)$. Second, the cost of applying a substitution to the strands in a state is $O(\psi^3)$ (line 23), since there are at most ψ action steps in a state, and there are at most ψ variables in an action step, and for the instantiation T of each variable $|T|_{DAG} = O(\psi)$. Note that it is possible to reduce the cost of applying a substitution to the strands in a state to $O(\psi)$ if we arrange the computation using pointers and links. But $O(\psi^3)$ is enough to prove our complexity results. Third, with proper organization of the data structure for the $\lesssim$ relationship, the cost to check the correctness properties for a state is also $O(\psi)$, which is the maximum number of nodes in a state.

$RRA\_Checker$ will terminate after

$$O(\psi^{\psi} \times \psi^3) = O(\{2^{log_2\psi}\}^{\psi} \times \psi^3) = O(2^{log_2\psi \times \psi + log_2\psi^3}) = O(2^{\psi^2})$$

instructions. Since $|Pro|_{bit}$ is at most ζ and N is at most 2^{ζ},

$$\psi = |Pro|_{bit} \times N < \zeta \times 2^{\zeta} = 2^{log_2\zeta + \zeta} < 2^{2\zeta}.$$

The algorithm terminates in no more than

$$O(2^{\psi^2}) = O(2^{(2^{2\zeta})^2}) = O(2^{2^{4\zeta}})$$

instructions, which is in 2-EXPTIME, since 4ζ is a polynomial function of ζ.

Now we analyze the other case. When NRI is a fixed number n, the input size of $RRA\_Checker$ is ζ, and $|Pro|_{bit} = O(\zeta)$. Then $\psi = |Pro|_{bit} \times n < n\zeta$. The time complexity is no more than $O(2^{\psi^2}) < O(2^{(n\zeta)^2}) = O(2^{n^2\zeta^2})$, which is in EXPTIME, since $n^2\zeta^2$ is a polynomial of ζ where n is a constant. ■

The following lemma shows the completeness of $RRA\_Checker$. There is a proof of the complete-inclusive property of Athena in [13]. We believe our approach provides more details from a different perspective.

***Lemma** 3:* For every possible attack atk such that the freshness goal of V to R is violated, the following is true. During the computation of $RRA\_Checker$ for the freshness goal, before every iteration of the while loop, i.e, whenever line 3 is reached, atk is associated to a state s in the set $STATES$ by a map θ and a substitution η, and θ and η must satisfy the following conditions.

1) There is a non-empty subset of the role instances of atk, call it ris, and a one-to-one map θ between the strands of s ($s.strands$) and the role instances in ris. θ can be obviously extended to associate the nodes of s to the blocks of ris, and the steps of s to the events of ris.
2) There is a substitution η from the atomic terms appearing in $s.strands$ and the ground terms appearing in ris, such that if $\theta(st) = ev.step$, where st is a step (of a strand) of s, and $ev.step$ is the step of the corresponding event ev in ris, then $\eta(st) = ev$.
3) If $nd \twoheadrightarrow nd' \in s.binding$, let stp and stp' be the two steps in s where the two nodes nd and nd' belong respectively, and let T be the term of two nodes, then $\theta(stp')$ is the earliest event in atk such that the block $\eta(T)$ is sent, i.e., for every event ev in atk such that a block $\eta(T)$ is sent by ev, then $ev.time \geq \theta(stp').time$.
4) For two steps stp and stp' of s, if $stp' \lesssim stp$, then $\theta(stp').time < \theta(stp).time$.

Proof: The proof is by induction on the iterations of the while loop. The base case is obvious for $state^0$. Since the algorithm consider all possible ways that a block can be generated the first time in a run, if an attack is associated with a state s by some map θ and substitution η, then the attack must be associated with a state s', which is a child of s, by some θ' and η', which can be easily obtained by adjusting and extending θ and η of s. ■

The soundness and completeness of Athena to check secrecy and authentication when NRI is bounded have been addressed in [13], and the soundness and completeness of RRA_Checker can be proved similarly. The soundness of the RRA_Checker is obvious: when RRA_Checker reports an attack, then there is an attack that violates the freshness goal. The completeness of RRA_Checker can be proved by induction on the iterations of the RRA_Checker to show that for any run of the protocol that violates the freshness goal, a subset of the role instances in the run can always be mapped to the strands of a reachable state during the computation of RRA_Checker. In [27] further details of the correctness of RRA_Checker are provided; therefore, we can prove the following theorem.

E. Decidability Results

***Theorem** 3:* Checking RRA for a freshness goal is 2-EXPTIME when NRI is individually bounded, and is EXPTIME when NRI is fixed.

When NRI is individually bounded or fixed, the complexity of checking DRA is no more than checking RRA, and we have the following corollary.

***Corollary** 1:* Checking DRA for a freshness goal is 2-EXPTIME when NRI is individually bounded, and is EXPTIME when NRI is fixed.

Since DRA can be considered a special case of RRA, RRA_Checker can be adapted to check DRA. Then the goal binding mechanism is further restricted, so that a node nd received at location L can only bind to a node nd' sent at location L, according to the protocol. Suppose r' is the new strand introduced to a state by the binding from nd to nd', where nd' appears in r', and nd appears in a strand r already included in the state. Let m and m' be the largest message number of a receiving action step in r and r' respectively. Then it is true that $m' < m$. By this observation, it is not difficult to design a measure which is strictly decreasing in any branch of the state tree of RRA_Checker. Since RRA_Checker is finitely branching, its termination of checking DRA is obvious, even when NRI is unbounded. The soundness and completeness of the algorithm still hold for checking DRA. So we have the following theorem.

***Theorem** 4:* Checking DRA for a freshness goal is decidable when the number of role instances in a run (NRI) is unbounded.

***Theorem** 5:* Checking RRA for a freshness goal is NP-complete when NRI is fixed by a number n.

Proof: A non-deterministic algorithm can just guess a branch of the tree of states of $RRA\_Checker$, which has at most $\psi = |Pro|_{bit} \times n$ states. Since the cost to generate a state is $O(\psi^3)$, and $\psi < n\zeta$, the total cost of a branch of RRA_Checker is $O(\psi) \times O(\psi^3) = O(\psi^4) = O(n^4\zeta^4) = O(\zeta^4)$. Since ζ^4 is a polynomial function of ζ, checking RRA for a freshness goal is NP when NRI is fixed.

Second, we show that the problem is NP-hard, by a reduction from a well-known NP-hard problem, 3-SAT. Given an instance of 3-SAT problem, we translate it into a problem $\langle Pro, R, V, n\rangle$. Here n is the fixed number, $n \geq 2$, such that no run with the number of role instances (NRI) more than n is considered. We prove the NP-hardness for $n = 2$, since the same proof can be applied to show the NP-hardness of deciding other languages of $\langle Pro, R, V, n\rangle$ for $n > 2$.

A 3-SAT problem is a tuple $\langle Var, formula\rangle$, where Var is a set of variables, $formula$ is a conjunction of a sequence of clauses, with the form $C_1 \bigwedge C_2 \cdots \bigwedge C_u$, where C_i is a clause, for $1 \leq i \leq u$ and $1 \leq j$. C_i is a disjunction of three literals of form $L_i^1 \vee L_i^2 \vee L_i^3$. Each literal is a variable X or a negation of a variable $\neg X$, $X \in Var$. The problem is to decide if there is an assignment of $\top$ (truth) or $\bot$ (false) to the variables in Var such that $formula$ can be evaluated to $\top$.

A 3-SAT problem $\langle Var, formula\rangle$ is translated to a protocol Pro. Suppose $Var = \{X_1, X_2, \cdots X_g\}$, $1 \leq g$, these g variables correspond to g different constants $x_1, x_2, \cdots x_g$. The other constants that appear in the protocol are $\{1, 2, \cdots u, true, final\}$. Note that there u clauses in $formula$.

The communication sequence of the protocol starts with the following 4 message exchanges between A and B.

1. $\#_A(N_A^0)(A \Rightarrow B):$ $A,\ B,\ \bot,\ \top, \{x_1, \bot, N_A^0\}_k^{\leftrightarrow},\ \{x_1, \top, N_A^0\}_k^{\leftrightarrow},$
$\cdots \{x_g, \bot, N_A^0\}_k^{\leftrightarrow},\ \{x_g, \top, N_A^0\}_k^{\leftrightarrow},$
$\{0, true, N_A^0\}_k^{\leftrightarrow}, \cdots \{u, true, N_A^0\}_k^{\leftrightarrow}$
2. $\#_B(X_1, X_2, \cdots X_g)(B \Rightarrow A):$ $B, A, X_1, X_2, \cdots X_g$
3. $\#_A(N_A, N_A')(A \Rightarrow B):$ $A,\ B,\ \{x_1, X_1, N_A\}_k^{\leftrightarrow}, \cdots \{x_g, X_g, N_A\}_k^{\leftrightarrow},$
$\{final, N_A, N_A'\}_k^{\leftrightarrow},\ \{0, true, N_A\}_k^{\leftrightarrow},\ N_A'$
4. $\#_B(N_B)(B \Rightarrow A):$ $B,\ A,\ \{final, N_A, N_B\}_k^{\leftrightarrow}$

Here we explain some of the above message exchanges. In the first message, call it Msg_1, the terms of the form $\{x_i, \top/\bot, N_A^0\}_k^{\leftrightarrow}$, which means $\{x_i, \top, N_A^0\}_k^{\leftrightarrow}$ or $\{x_i, \bot, N_A^0\}_k^{\leftrightarrow}$, for $1 \leq i \leq g$, are designed as "dummy terms" to simply fill some needed parts in a message, and they will be used to finish some role instances since in the attack two role instances need to finish execution. For example we will see that in the proof in a role instance of of A, A will need to receive some term of the form $\{x_i, \top/\bot, N_i^j\}_k^{\leftrightarrow}$, then the attacker can use a term $\{x_i, \top/\bot, N_A^0\}_k^{\leftrightarrow}$ generated in Msg_1 instead. The dummy terms will not actually be used to simulate an evaluation of a formula. The terms of the form $\{f, true, N_A^0\}_k^{\leftrightarrow}$, $0 \leq f \leq u$, are also dummy terms, which will be used to fill the places of $\{i, true, N_i^j\}_k^{\leftrightarrow}$ (introduced later in the role of A). In Msg_2, A is supposed receive g nonces, and the attacker need to provide either $\top$ or $\bot$ for each of the g nonces in order to possibly make an attack. In Msg_3, A send out the terms of the form $\{x_i, \top/\bot, N_A\}_k^{\leftrightarrow}$. These terms are used to indicate the value of the variable X_i in an assignment. The nonce N_A can ensure that X_i is associated with a unique value, since it is impossible that both $\{x_i, \top, N_A\}_k^{\leftrightarrow}$ and $\{x_i, \bot, N_A\}_k^{\leftrightarrow}$ can appear in a run, for a unique N_A; therefore, X_i can only be assigned with one value. N_A' is generated as a block in Msg_3 so that N_A' can be used in the attack to replace N_Y which will appear in the last two messages (introduced later) of the protocol.

For each clause C_i, $1 \leq i \leq u$, starting from C_1, do the following. For each literal of C_i, say the j^{th} literal, $1 \leq j \leq 3$, if a positive literal of a variable X_h appears in C_1, $X_h \in Var$ which means $1 \leq h \leq g$, then the following message exchanges are appended to Pro, which is considered as a communication sequence. Note that we do not indicate the message numbers of the message exchanges to be appended, since the proper message numbers should be obviously assigned sequentially starting from 5. Each message includes the sender's name and the receiver's name to make the meaning more obvious.

. $\#_B(N_i^j)(B \Rightarrow A):$ $B, A, \{i-1, true, N_i^j\}_k^{\leftrightarrow},\ \{x_h, \top, N_i^j\}_k^{\leftrightarrow}$
. $(A \Rightarrow B):$ $A, B, \{i, true, N_i^j\}_k^{\leftrightarrow}$

Otherwise if a negative literal of X_h, which is $\neg X_h$, appears in the j^{th} literal of C_i, then the following message exchanges are appended Pro.

. $\#_B(N_i^j)(B \Rightarrow A):$ $B,\ A,\ \{i-1, true, N_i^j\}_k^{\leftrightarrow},\ \{x_h, \bot, N_i^j\}_k^{\leftrightarrow}$
. $(A \Rightarrow B):$ $A,\ B,\ \{i, true, N_i^j\}_k^{\leftrightarrow}$

Finally the following two messages are appended to $Pro.CS$.

. $\#_B(N_X, N_Y)(B \Rightarrow A):\ \ B,\ A,\ \{u, true, N_X\}_k^{\leftrightarrow},\ N_Y$

. $(A \Rightarrow B):\ \ A,\ B,\ \{final, N_X, N_Y\}_k^{\leftrightarrow}$

The protocol implies two roles executed by A and B respectively, call the two roles R_A and R_B. The freshness goal to be checked is N_B to R_A. The target problem is $\langle Pro, D, R_A, N_B, 2\rangle$. The translation time is in linear to the size of the source problem $\langle Var, formula\rangle$.

Now we show the correctness of the reduction in two directions. When we describe a role instance we only show the steps of its events, not the $time$ fields. The $time$ fields are not shown since they can be arranged easily by assign some sequential number following the order that the events are introduced in the attack.

Direction 1: we show that if there is an assignment to the variables of the 3-SAT problem to evaluate the formula to $\top$, there is an attack with 2 role instances of R_A such that the freshness goal of N_B to R_A can be violated. The attack includes two different role instance r_1 and r_2 of R_A, both are executed by an agent a while a considers the interlocutor is b, while the attacker impersonates b and construct all messages that are supposed to be sent by b. The first 4 messages sent or received by a in r_1 are as follows.

1. $\#_(n_a^0) + (a \Rightarrow b):\ \ a,\ b,\ \bot,\ \top,\ \{x_1, \bot, n_a^0\}_k^{\leftrightarrow},\ \{x_1, \top, n_a^0\}_k^{\leftrightarrow},$
$\cdots\ \{x_g, \bot, n_a^0\}_k^{\leftrightarrow},\ \{x_g, \top, n_a^0\}_k^{\leftrightarrow},$
$\{0, true, n_a^0\}_k^{\leftrightarrow},\ \cdots\ \{u, true, n_a^0\}_k^{\leftrightarrow}$
2. $-(b \Rightarrow a):\ \ b,\ a,\ \top/\bot,\ \top/\bot,\ \cdots\ \top/\bot$
3. $\#_A(n_a, n'_a) + (a \Rightarrow b):\ \ a,\ b,\ \{x_1, \top/\bot, n_a\}_k^{\leftrightarrow}, \cdots\ \{x_g, \top/\bot, n_a\}_k^{\leftrightarrow},$
$\{final, n_a, n'_a\}_k^{\leftrightarrow},\ \{0, true, n_a\}_k^{\leftrightarrow},\ n'_a$
4. $-(b \Rightarrow a):\ \ b,\ a,\ \{final, n_a, n'_a\}_k^{\leftrightarrow}$

All of the messages that are received by a and are supposed to be sent by b are constructed by the attacker using the blocks already sent by a. In message 2, the attacker choose the assignment, by which the formula of the 3-SAT problem is evaluated to $\top$, to replace X_1 to X_j. $\top/\bot$ means the term is either $\top$ or $\bot$.

Now r_1 has executed four messages. Before r_1 finishes the remaining messages, another role instance of $A's$ role starts, call it r_2. We choose a to execute r_2, but r_2 can be executed by any agent and the proof still works. The first 3 messages sent or received by r_2 are as follows.

1. $\#_(m_a^0) + (a \Rightarrow b):\ \ a,\ b,\ \bot,\ \top, \{x_1, \bot, m_a^0\}_k^{\leftrightarrow},\ \{x_1, \top, m_a^0\}_k^{\leftrightarrow},$
$\cdots \{x_g, \bot, m_a^0\}_k^{\leftrightarrow},\ \{x_g, \top, m_a^0\}_k^{\leftrightarrow},$
$\{0, true, m_a^0\}_k^{\leftrightarrow}, \cdots \{u, true, m_a^0\}_k^{\leftrightarrow}$
2. $-(b \Rightarrow a):\ \ b, a, \top/\bot, \top/\bot, \cdots \top/\bot$
3. $\#_A(m_a, m'_a) + (a \Rightarrow b):\ \ a, b, \{x_1, \top/\bot, m_a\}_k^{\leftrightarrow}, \cdots \{x_g, \top/\bot, m_a\}_k^{\leftrightarrow},$
$\{final, m_a, m'_a\}_k^{\leftrightarrow},\ \{0, true, m_a\}_k^{\leftrightarrow},\ m'_a$

Note that in Msg_2 of r_2 the attacker choose the same assignment of variables by which the formula evaluates to $\top$. We want to let a receive n'_a again in r_2 which instantiates N_B the same as in r_1, which means that Msg_4 of r_2 should be

$$4.\ -(b \Rightarrow a) : \{final, m_a, n'_a\}_k^{\leftrightarrow}.$$

However the term $\{final, m_a, n'_a\}_k^{\leftrightarrow}$ has not been generated yet.

Before r_2 finishes its Msg_4, r_1 continues to execute the rest of its messages. According to the design of the protocol, for each literal in the formula there are two messages in A's role, listed sequentially. Since the assignment makes the formula true, for each clause there is at least one literal in it that is evaluated to be $\top$. For the the j^{th} literal $1 \leq j \leq 3$, of the i^{th} clause C_i, $1 \leq i \leq u$, call it L_i^j. If L_i^j is evaluated to $\top$ by the assignment, and variable X_h appears in L_i^j, $1 \leq h \leq g$, then the corresponding twos messages to handle L_i^j in r_1 are as follows.

. $-(b \Rightarrow a):\ \ b,\ a,\ \{i-1, true, m_a\}_k^{\leftrightarrow},\ \{x_h, \top/\bot, m_a\}_k^{\leftrightarrow}$

. $+(a \Rightarrow b):\ \ a,\ b,\ \{i, true, m_a\}_k^{\leftrightarrow}$

where $\top/\bot$ means $\top$ (or $\bot$) if according to the assignment the value of X_h should be $\top$ (or $\bot$ respectively), so that L_i^j is evaluated $\top$. It can be proved by induction that the term $\{i-1, true, m_a\}_k^{\leftrightarrow}$ can always be generated in an earlier message sent by r_1. Note that the first one $\{0, true, m_a\}_k^{\leftrightarrow}$ has been generated in r_2 in Msg_3.

Otherwise if L_i^j is evaluated to $\bot$ by the assignment, by which the whole formula is evaluated to $\top$, then the corresponding twos messages to handle L_i^j in r_1 are as follows.

. $-(b \Rightarrow a):\ \ b,\ a,\ \{i-1, true, n_a^0\}_k^{\leftrightarrow},\ \{x_h, \top/\bot, n_a^0\}_k^{\leftrightarrow}$

. $+(a \Rightarrow b):\ \ a,\ b,\ \{i, true, n_a^0\}_k^{\leftrightarrow}$

Again the choice of $\top$ or $\bot$ depends on the design of the Pro described earlier. The two blocks $\{i-1, true, n_a^0\}_k^{\leftrightarrow}$ and $\{x_h, \top/\bot, n_a^0\}_k^{\leftrightarrow}$ have been generated in Msg_1 of r_1. Note that we can also use the dummy terms of $\{i -$

$1, true, m_a^0\}_k^{\leftrightarrow}$ and $\{x_h, \top/\bot, m_a^0\}_k^{\leftrightarrow}$ generated by r_2, since the purpose here is just to let r_2 continue its execution.

It is guaranteed that for each clause C_i, $1 \leq i \leq u$, a block of $\{i, true, m_a\}_k^{\leftrightarrow}$ is generated; therefore, a term $\{u, true, m_a\}_k^{\leftrightarrow}$ should be generated following the steps described above. Then the last two messages of r_1 are the follows.

. $-(b \Rightarrow a):\ \ b,\ a,\ \{u, true, m_a\}_k^{\leftrightarrow},\ n'_a$

. $+(a \Rightarrow b):\ \ a,\ b,\ \{final, m_a, n'_a\}_k^{\leftrightarrow}$

Note that the attacker can provide n'_a since it was sent by Msg_3 of r_1.

Now the r_2 resumes to execute its Msg_4, where $\{final, m_a, n'_a\}_k^{\leftrightarrow}$ is received, and n'_a instantiates N_B, the same as in r_1. The attacker can provide the messages to be received by the rest of r_2 by using the dummy terms generated by the first 3 messages of either r_1 or r_2. So there is an attack with only two role instances that violates the freshness goal of N_B to A's role. Note that by the definition of GRA the two role instances need to be fully executed.

Direction 2: we want to prove that if there is a run with 2 role instances, r_1 and r_2, such that the freshness goal of N_B to A's role is violated, then the corresponding $3-SAT$ formula can be evaluated to $\top$ with an value assignment to its variables. r_1 and r_2 must both belong to $A's$ role in order to make the attack possible (by the definition of a freshness goal, and we are considering only two role instances). In this run in Msg_4 of r_1 the term $T_1 = \{final, n_a, v\}_k^{\leftrightarrow}$ is received, for some term v, while in Msg_4 of r_2 a term $T_2 = \{final, m_a, v\}_k^{\leftrightarrow}$ is received, with the same v that instantiates N_B, and n_a and m_a are freshly generated in r_1 and r_2 respectively to instantiate N_a. Note that $T_1 \neq T_2$ since $n_a \neq m_a$. There are four terms of the form $\{final, X, Y\}_k^{\leftrightarrow}$ that can be generated (sent, not received) by r_1 and r_2: $\{final, n_a, n'_a\}_k^{\leftrightarrow}$ generated by Msg_3 of r_1, or $\{final, m_a, m'_a\}_k^{\leftrightarrow}$ generated by Msg_3 of r_1, or the last message of r_1 $\{final, G, H\}_k^{\leftrightarrow}$, or the last message of r_2 $\{final, U, V\}_k^{\leftrightarrow}$.

There are several impossible cases to consider where T_1 and T_2 are generated.

- T_1 and T_2 are generated in the same place in the same role instance. It is impossible since $T_1 \neq T_2$.
- T_1 and T_2 are both generated by the last message of A's role, one in r_1 and the other in r_2. It is impossible since it means the attacker can provide one of T_1 and T_2 before it is generated. We can see the conflict by reasoning about which message is generated earlier.
- T_1 and T_2 are both generated by the Msg_3 of A's role, one in r_1 and the other in r_2. It is impossible since the two terms generated in r_1 and r_2 at Msg_3 are $\{final, n_a, n'_a\}_k^{\leftrightarrow}$ and $\{final, m_a, m'_a\}_k^{\leftrightarrow}$ where $n'_a \neq m'_a$.

The only possible case is that T_1 is generated in Msg_3 of A's role and T_2 is generated in the last message of A's role, either in a single role instance or in two different role instances of A's role. Without loss of generality, we consider T_1 is generated in Msg_3. Then T_1 must be generated in Msg_3 of r_1 since the term N_A is uniquely generated in A's role. So $T_1 = \{final, n_a, n'_a\}_k^{\leftrightarrow}$. Then it must be true that $T_2 = \{final, m_a, n'_a\}_k^{\leftrightarrow}$. Since T_2 is received by Msg_4 of r_2, T_2 cannot be generated by the last message of r_2, which occurs later. So T_2 must be generated by the last message of r_1. Then in the second to the last message of r_1 a term $\{u, true, m_a\}_k^{\leftrightarrow}$ must be received by r_1. The attacker can always provide the other terms for this message, including the agent names A and B and n'_a, which is a block generated by Msg_3 of r_1. The question is that how $\{u, true, m_a\}_k^{\leftrightarrow}$ is generated. Note that $\{0, true, m_a\}_k^{\leftrightarrow}$ is available since it is generated by the Msg_3 of r_2. Then it can be proved by induction that for each term of the form $\{i, true, m_a\}_k^{\leftrightarrow}$ can be generated in the run, for $1 \leq i \leq u$, there is a value assignment to the variables in Var such that all of the clause C_f, $1 \leq f \leq i$, are evaluated to $\top$; therefore, since $\{u, true, m_a\}_k^{\leftrightarrow}$ is generated, all of the clauses are evaluated to $\top$ by some assignment.

The above proof works for $n = 2$. And the same proof can apply to other languages of $\langle Pro, D, R, V, n\rangle$, where $n > 3$, to show the NP-completeness. ■

We have attempted to check DRA by a set of efficient reasoning rules that take advantage of the restrictions for the attacker. It seems that checking DRA is P (decidable in polynomial time) even NRI is unbounded. Adapting Athena to check DRA cannot get a polynomial time result since we have found some protocols that require exponential time for Athena to check DRA.

To show the NP-completeness of checking GRA following the approach of this paper will need to incorporate the attacker's internal computation (the attacker strands of Athena) into the model checker. Furthermore we have to prove that in a non-deterministic branch of the computation of the model checker, the DAG size of the substitution is polynomial to the DAG size of the protocol, as discussed in [5]. Since we have some doubts on the existing proofs of NP-completeness for checking secrecy [5] as mentioned earlier, and we discuss the NP proof in details in [23], we are trying to have a better approach to show the NP-completeness of secrecy and authentication while a bound on role instances is considered, which we believe should cover the NP-completeness of checking GRA when NRI is fixed.

We summarize the complexity results of checking freshness in Table III-E. The results surrounded by two question marks are by postulation and have not been proved by this paper and are under investigation.

attack \ NRI	unbounded	individually bounded	fixed
DRA	Decidable, ?P?	2-EXPTIME, ?P?	EXPTIME, ?P?
RRA	Undecidable	2-EXPTIME	NP-complete
GRA	Undecidable	?2-EXPTIME?	?NP-complete?

TABLE I
COMPLEXITY OF CHECKING FRESHNESS. THE RESULTS MARKED WITH DOUBLE QUESTION MARKS ARE BY POSTULATION AND HAVE NOT BEEN PROVED BY THIS PAPER. OTHER RESULTS ARE PROVED.

The following remark shows that the difference between the considerations of fixed or individually bounded NRI may not affect the complexity results if we do not measure NRI as length of bits.

***Remark** 1:* The decidability complexity results depends on the measure of the of the input size of the checker. If we measure the number of role instances separately by a natural number, not by length of bits, which is similar to the way that the number of elements of a sequence is measured as the input size of a sorting algorithms, and the input size of checking GRA or RRA is measured as $|\langle Pro, D, R, V|_{bit} + N$ or $|\langle Pro, D, R, V|_{bit} \times N$, then for checking RRA, the complexity is the NP-complete for both cases of fixed or individually bounded NRI. And we believe that with the adjusted input size measure, the complexity results of checking GRA with fixed or individually bounded NRI are the same.

IV. SUMMARY

In this paper we define freshness goal and its attacks and investigate the complexity of checking freshness, which is the first research on this topic. The techniques of modeling, reduction, and model checking have novel features and can be applied generally in this area. Currently we are investigating the polynomial time algorithm to check DRA, and we use an approach that achieves efficiency by tracing the mechanism of challenge-response, which is rather different from the approach of using a model checker in this paper. We are also studying an improved approach to prove NP-completeness of checking secrecy, which can also be applied to authentication and checking GRA. We expect to extend the NP-complete proof of this paper and to handle several demanding issues discussed at the end of Section III. These substantial works are proper to be addressed in subsequent researches beyond the scope of this paper.

REFERENCES

[1] Z. Liang and R. M. Verma, "Complexity of Checking Freshness of Cryptographic Protocols," in *Fourth International Conference on Information Systems Security (ICISS 2008)*, JNTU, Hyderabad, India, December 2008.

[2] D. Dolev and A. C.-C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.

[3] N. A. Durgin, P. Lincoln, and J. C. Mitchell, "Multiset rewriting and the complexity of bounded security protocols." *Journal of Computer Security*, vol. 12, no. 2, pp. 247–311, 2004.

[4] R. Ramanujam and S. P. Suresh, "Undecidability of secrecy for security protocols," Manuscript, July 2003, http://www.imsc.res.in/˜jam/TR-undec.ps.gz.

[5] M. Rusinowitch and M. Turuani, "Protocol insecurity with a finite number of sessions, composed keys is NP-complete." *Theor. Comput. Sci.*, vol. 1-3, no. 299, pp. 451–475, 2003.

[6] Ferucio L. Ţiplea and C. Enea and C. V. Bîrjoveanu, "Decidability and complexity results for security protocols," "Al.I.Cuza" University of Iaşi, Faculty of Computer Science, Tech. Rep. TR 05-02, 2005. [Online]. Available: http://thor.info.uaic.ro/˜tr/tr05-02.pdf

[7] J. K. Millen and V. Shmatikov, "Constraint solving for bounded-process cryptographic protocol analysis." in *ACM Conference on Computer and Communications Security*, 2001, pp. 166–175.

[8] Z. Liang and R. M. Verma, "Secrecy Checking of Protocols: Solution of an Open Problem," in *Automated Reasoning for Security Protocol Analysis (ARSPA 07)*, July 2007, pp. 95–112.

[9] ——, "Improving Techniques for Proving Undecidability of Checking Cryptograhpic Protocols," in *The Third International Conference on Availability, Security and Reliability*. Barcelona, Spain: IEEE Computer Society, March 2008, pp. 1067–1074, workshop on Privacy and Security by means of Artificial Intelligence (PSAI).

[10] L. Gong, "Variations on the themes of message freshness and replay—or the difficulty of devising formal methods to analyze cryptographic protocols," in *Proceedings of the Computer Security Foundations Workshop VI.* IEEE Computer Society Press, 1993, pp. 131–136. [Online]. Available: citeseer.ist.psu.edu/gong93variations.html

[11] K.-Y. Lam and D. Gollmann, "Freshness Assurance of Authentication Protocols," in *ESORICS '92: Proceedings of the Second European Symposium on Research in Computer Security*. London, UK: Springer-Verlag, 1992, pp. 261–272.

[12] D. X. Song, "Athena: A new efficient automatic checker for security protocol analysis." in *CSFW*, 1999, pp. 192–202.

[13] D. X. Song, S. Berezin, and A. Perrig, "Athena: A novel approach to efficient automatic security protocol analysis," *Journal of Computer Security*, vol. 9, no. 1/2, pp. 47–74, 2001. [Online]. Available: citeseer.ist.psu.edu/song01athena.html

[14] R. Corin, S. Etalle, and A. Saptawijaya, "A logic for constraint-based security protocol analysis," in *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 155–168.

[15] M. Backes, A. Cortesi, R. Focardi, and M. Maffei, "A Calculus of Challenges and Responses," in *Proceedings of 5th ACM Workshop on Formal Methods in Security Engineering (FMSE)*, November 2007.

[16] J. D. Guttman and F. J. Thayer, "Authentication tests." in *IEEE Symposium on Security and Privacy*, 2000, pp. 96–109.

[17] Z. Liang and R. M. Verma, "Improving Techniques for Proving Undecidability of Checking Cryptographic Protocols," Computer Science Department, University of Houston, Texas, USA, http://www.cs.uh.edu/preprint or http://www.cs.uh.edu/˜zliang, Tech. Rep., January 2008.

[18] L. C. Paulson, "The inductive approach to verifying cryptographic protocols." *Journal of Computer Security*, vol. 6, no. 1-2, pp. 85–128, 1998.

[19] F. J. Thayer, J. C. Herzog, and J. D. Guttman, "Strand spaces: Proving security protocols correct." *Journal of Computer Security*, vol. 7, no. 1, 1999.

[20] G. Lowe, "A hierarchy of authentication specifications," in *CSFW '97: Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*. Washington, DC, USA: IEEE Computer Society, 1997, p. 31.

[21] P. F. Syverson, "A taxonomy of replay attacks." in *CSFW*, 1994, pp. 187–191.

[22] S. Froschle, "The insecurity problem: Tackling unbounded data," in *IEEE Computer Security Foundations Symposium 2007*. IEEE Computer Society, 2007, pp. 370–384.

[23] Z. Liang and R. M. Verma, "Study Notes of a NP-completeness Proof," Computer Science Department, University of Houston, www.cs.uh.edu/preprint, Tech. Rep. UH-CS-08-15, October 2008.

[24] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani, "An np decision procedure for protocol insecurity with xor." *Theor. Comput. Sci.*, vol. 338, no. 1-3, pp. 247–274, 2005.

[25] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computability*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[26] G. Lowe, "An Attack on the Needham-Schroeder Public-Key Authentication Protocol." *Inf. Process. Lett.*, vol. 56, no. 3, pp. 131–133, 1995.

[27] Z. Liang and R. M. Verma, "Complexity of Checking Freshness of Cryptographic Protocols," Computer Science Department, University of Houston, Texas, USA, http://www.cs.uh.edu/preprint, Tech. Rep., September 2008, uH-CS-08-14.

[28] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers." *Commun. ACM*, vol. 21, no. 12, pp. 993–999, 1978.

Appendix

A. The PKNS Protocol and Challenge-Response

$$\begin{array}{lllll}
1. & \#_A(N_A) & (A \Rightarrow B): & \{N_A, A\}_{\overrightarrow{k^1_B}} \\
2. & \#_B(N_B) & (B \Rightarrow A): & \{N_A, N_B\}_{\overrightarrow{k^1_A}} \\
3. & & (A \Rightarrow B): & \{N_B\}_{\overrightarrow{k^1_B}}
\end{array}$$

$$\begin{array}{lll@{\qquad}lll}
Role_A & & & Role_B & & \\
1. & \#_A(N_A) & +(A \Rightarrow B): \{N_A, A\}_{\overrightarrow{k^1_B}} & 1. & & -(A \Rightarrow B): \{N_A, A\}_{\overrightarrow{k^1_B}} \\
2. & & -(B \Rightarrow A): \{N_A, N_B\}_{\overrightarrow{k^1_A}} & 2. & \#_B(N_B) & +(B \Rightarrow A): \{N_A, N_B\}_{\overrightarrow{k^1_A}} \\
3. & & +(A \Rightarrow B): \{N_B\}_{\overrightarrow{k^1_B}} & 3. & & -(A \Rightarrow B): \{N_B\}_{\overrightarrow{k^1_B}}
\end{array}$$

Fig. 1. The communication sequence of the PKNS protocol and its two roles.

Challenge-response is a mechanism in a protocol that an agent A freshly creates a term N_A and send it out (in A's role) and then later N_A appears in some message received by A (in A's role). The challenge-response of the simplest form can be found in the public key Needham-Schroeder protocol with three messages (the core part), we call it the ***PKNS*** protocol [28], showed in Figure 1. In the PKNS protocol, A and B mutually challenge each other. A performs a freshness challenge to B by creating and sending a nonce N_A to B in Msg_1 and receiving N_A back in Msg_2. Similarly B challenges A by creating and sending a nonce N_B to A in Msg_2 and receiving it back in Msg_3.

Freshness-challenge can guarantee some security properties. For example, for the PKNS protocol, when A receives Msg_2, A can make sure that the if the sender (or creator), call it P, of Msg_2 is honest, it is guaranteed that N_B is generated by P after P receives N_A. By this way A can make sure P has actively responded A's message. This observation is true despite there is a famous attack to the PKNS protocol found by Lowe [26]. For the PKNS protocol, the problem is that A cannot ensure that the creator of Msg_2 is B, whom A intended to talk with.

B. On Defining Freshness

The definition of freshness of [14] is quoted as follows.

> Data D is fresh whenever a principal A never completes a run with another principal agreeing on D, if once in the past A has already completed a protocol run with another principal agreeing on the same data D.

The following observation describes a protocol with the simplest challenge-response, and assumes an ideal situation where the attacker's behavior is restricted enough so that obviously the attacker does not interfere with the challenge-response mechanism, although the attacker can replay messages. This observation is relevant to understand the freshness goal and the attacker's involvement, which will be discussed soon.

Observation 1: Consider a protocol with the communication sequence between A and B as follows.

1. $\#_A(N_A)$ $\quad (A \Rightarrow B) : \{A, N_A\}^{\leftrightarrow}_{K}$
2. $\#_B(N_B)$ $\quad (B \Rightarrow A) : \{B, N_A, N_B\}^{\leftrightarrow}_{K}$

Assume the message received by each role instance of A's role (or B's role) must be sent by some role instance of B's role (or A's role respectively). It is impossible to have a run of the protocol with two role instances of A's role where in the variable N_B is instantiated by the same ground term in the two role instances.

Proof: Suppose the contrary of the observation, in a run of the protocol there are two role instances of A's role, r_1 and r_2, where N_B is instantiated by the same term n_b. Let r_1 and r_2 be executed by A_1 and A_2 respectively, where A_1 and A_2 could be different agents. There must be two different nonces n_a^1 and n_a^2 that are freshly generated in r_1 and r_2 respectively to instantiate N_A. The second message of r_1 must be $\{B_1, n_a^1, n_b\}^{\leftrightarrow}_{K}$, call it msg_1, and the second message of r_2 must be $\{B_2, n_a^2, n_b\}^{\leftrightarrow}_{K}$, call it msg^2. Note that B_1 and B_2 could be different. By the condition assumed by the observation, msg_1 and msg_2 must be sent by some role instances of B's role. Since in B's role only one message is sent, and $msg_1 \neq msg_2$, msg_1 and msg_2 must be sent by two different role instances of B's role, which implies that n_b is freshly generated in two different role instances to instantiate N_B, impossible due to our assumption of unbounded nonce generation. ■

It can be complex to check the property described by the above observation, that no two role instances of A's role can receive the same N_B, when the challenge-response mechanism is more complex or the attacker has more complex involvement in a protocol run.

C. Discussion of $RRA\_Checker$

For $RRA\_Checker$, the set of strands $strands$ in a state can be implemented in two other ways.

1) $strands$ is a queue. Then the state chosen to be checked is popped (line 4) and new states are appended to the queue (Line 14 and 26). In this way, the algorithm is doing breadth-first exploration of the possible states.
2) $strands$ is a stack. Then the state chosen to be checked is popped and the new states are pushed to the stack. In this way the algorithm is doing depth-first exploration of the possible states.

Generally the options of implementing $strands$ be a stack, queue or set are equivalent in terms of decidability and performance, when the number of role instances (NRI) in a run is fixed or individually bounded. But when NRI is unbounded, choosing $strands$ to be a queue and doing breadth-first exploration is better since it will guarantee that if there is an attack the algorithm will find it, but if there is no attack the algorithm is not guaranteed to terminate. In contrast choosing $strands$ to be a stack and doing depth-first exploration when NRI is unbounded will guarantee neither finding an attack nor proving the security goal is satisfied.

We have considered several implementation features to improve the performance of the model checker, and to avoid the redundant computations in some special cases, but they will not change the general complexity results of this paper.