Complexity of Checking Freshness of Cryptographic Protocols *

Zhiyao Liang Rakesh M Verma

Computer Science Department, University of Houston,
Houston TX 77204-3010, USA
Email: zliang@cs.uh.edu, rmverma@cs.uh.edu

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 goal 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 goals in several different scenarios, where the attacker's behavior is restricted differently, with different bounds on the number of role instances in a run.

Key words: Cryptographic protocols, freshness, replay attack, challenge response, model checker, undecidability, NP-completeness, Athena.

1 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 [1]. Many researchers follow Dolev and Yao attacker model. Much research on the complexity of checking security goals has focused on checking secrecy [2] [3] [4] [5] [6] [7] [8].

Freshness is a central and fundamental issue of communication protocols [9]. 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 [10]. 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. In [5] 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 points can be

* Research supported in part by NSF grants CCF 0306475 and CNS 0755500.

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 to timestamps, 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 goal that may be implemented by challenge-response. Further discussions on the challenge-response mechanism and its relationship with freshness are provided in [11].

The contributions of this paper are as follows.

- We define freshness goal 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 goals, called direct, restricted and general replay attacks.
- 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]. However our work is significantly different from [14], and cannot be covered by [14] for the following reasons. First, in [14] the authors demonstrate that the freshness goal can be expressed using the constraint solving system, but there is no complexity investigation. It is obvious that the 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 the 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].

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.

One approach of studying the complexity of checking freshness is to reduce the problem of checking freshness 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). But this approach requires that convincing proofs of the corresponding complexity results of checking secrecy are available.

Based on our works in [7] and [8], which improves the work of [3], 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 and 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 is by this direct reduction.

Reduction from checking secrecy to checking freshness can be done, and the idea is demonstrated in the proof of Theorem 2. 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. Details of these proofs are provided in [11]. The NP-hardness proof of checking secrecy we have noticed is provided by [4], which is by 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 [17] and [8]. Therefore in this paper we prove the NP-hardness of checking freshness in Theorem 5 by a direct reduction from 3-SAT.

Reduction from freshness to secrecy can also be done, as demonstrated in [11], although it may not be obvious. Such a reduction may show that checking freshness is NP when NRI is fixed since we believe checking secrecy is NP with fixed NRI, and this complexity result have been addressed in [4]. However, besides the significant contributions made by [4], we think there is an error in the proof of [4] to show checking secrecy is NP. More specifically the error is due to an assumption in the proof of Theorem 1 in [4] that the DAG size of a substitution of a term is no less than the DAG size of the term, which is incorrect. More details of the error is described in [11]. Currently we are studying another related paper [18]. Therefore we prove the NP result of checking freshness by directly analyzing a model checker, not by reduction from secrecy.

For space limit most of the detailed proofs are in the technical report [11], while in the paper we present the essential definitions and proving methods.

2 Notations and Modeling

[Notations] Notations are chosen in a style that is commonly used in the literature. More details of notations and modeling can be found in [11]. 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 I is the name of the attacker. Asymmetric keys are atomic terms. The established public key and private key of an agent A is 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 $\{T\}^{\rightarrow}_{k_A^i}$, $i \in \{0, 1\}$, 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 $\{T\}^{\leftrightarrow}_{Y}$, where T is the text and Y is any term working as the encryption key. When a list, say $[X, Y, Z, \cdots]$ is encrypted symmetrically or asymmetrically, the enclosing square brackets are removed within "{ }". 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 $-(B \Rightarrow P) : Msg$.

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.

[Model of protocol run] Here we present the essential model of protocol run. A verbose model can be found in [11]. 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 commonly 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 appeared in the run before its generation or 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 $\langle act,\ time\rangle$. act is a ground action step, described earlier. 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, such that the action steps of the sequence of events in r instantiate a prefix of the sequence of action steps, 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 I, denoted as $I.init$. The attacker can analyze and synthesize terms and create nonces by following a set of standard rules, as discussed by [19] and [4]. Given a certain set E of events that have occurred in a run, based on $I.init$ and the messages sent by E, $know_I(E)$ represents the (infinite) set of terms that can be derived following these standard rules.

A ***run*** is a tuple: $\langle Pro,\ D,\ R,\ AN,\ E\rangle$ 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. 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 in the role instances in R. The set of ***time points*** of 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 $msg \in know_I(E_{<\eta.time})$. 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 [19] and [6], but agrees with the strand space model [20].

[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, $run \in Runs^{Pro:D}$, 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 I, which is the attacker's name.
- The same ground term, say c, instantiates X in both r and r'. □

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 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 [21], 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 ([11] explains more). 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 described s 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 $\{X\}^{\rightarrow}_{Y}$ or $\{X\}^{\leftrightarrows}_{Y}$ has the location L, then X is located at $L.\alpha$, and Y is located at $L.\beta$.
- If 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 [22].

[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 [4] [2] [5] [3] [6] [7]. 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 shares 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, 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. Note that to check DRA or RRA, the intruder's initial knowledge pattern D is irrelevant.

3 Complexity Results on Checking Freshness

[Undecidability results]

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. We have used this approach similarly in [8]. In the 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 configuration 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 is known to I which encodes a final configuration reachable to M. Detailed proof is in [11]. □

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 [8], which has been specially designed so that the attacker I is an insider and I has to construct blocks in order to commit an attack. 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.

Second, we can do 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.

$\#_A(N_1)\ (A \Rightarrow B) : \{m, A, B, N_1, SECRET\}^{\rightarrow}_{k_B^1}$
$\#_B(N_2)\ (B \Rightarrow A) : \{m+1, B, A, N_1, N_2\}^{\leftrightarrow}_{SECRET}$
$(A \Rightarrow B) : \{m+2, A, B, N_2\}^{\leftrightarrow}_{SECRET}$

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. □

[Decidability results] 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 2. 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 [20] 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 [20] 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 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 computation 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, a step, 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 $nd' \lesssim nd$ **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.

Lemma 1: 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 [4] 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 [4]. 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 $|T|_{DAG} = O(\psi)$. 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.

Therefore $RRA\_Checker$ 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},

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

Therefore 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 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 [11] further details of the correctness of RRA_Checker are provided. Therefore we can prove the following theorem.

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. To prove NP-hardness we reduce the satisfaction problem of 3-SAT to a problem of checking freshness with NRI bounded to 2 ($n = 2$). The same proof can be applied for other cases where $n > 2$. Detailed proof is included in [11], which is delicate. □

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 [4]. Since we have some doubts on the existing proofs of NP-completeness for checking secrecy [4] as mentioned earlier, we are trying to have a better approach to show the NP-completeness of secrecy and authentication, which we believe should cover the NP-completeness of checking GRA when NRI is fixed.

We summarize the complexity results of checking freshness in the following table. 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?

4 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 3. These substantial works are proper to be addressed in subsequent researches beyond the scope of this paper.

References

1. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–207
2. Durgin, N.A., Lincoln, P., Mitchell, J.C.: Multiset rewriting and the complexity of bounded security protocols. Journal of Computer Security **12**(2) (2004) 247–311
3. Ramanujam, R., Suresh, S.P.: Undecidability of secrecy for security protocols. Manuscript (July 2003)
4. Rusinowitch, M., Turuani, M.: Protocol insecurity with a finite number of sessions, composed keys is NP-complete. Theor. Comput. Sci. **1-3**(299) (2003) 451–475

5. Ferucio L. Ţiplea and C. Enea and C. V. Bîrjoveanu: Decidability and complexity results for security protocols. Technical Report TR 05-02, "Al.I.Cuza" University of Iaşi, Faculty of Computer Science (2005)
6. Millen, J.K., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: ACM Conference on Computer and Communications Security. (2001) 166–175
7. Liang, Z., Verma, R.M.: Secrecy Checking of Protocols: Solution of an Open Problem. In: Automated Reasoning for Security Protocol Analysis (ARSPA 07). (July 2007) 95–112
8. Liang, Z., Verma, R.M.: 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) 1067–1074 Workshop on Privacy and Security by means of Artificial Intelligence (PSAI).
9. Gong, L.: 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) 131–136
10. Lam, K.Y., Gollmann, D.: Freshness Assurance of Authentication Protocols. In: ESORICS '92: Proceedings of the Second European Symposium on Research in Computer Security, London, UK, Springer-Verlag (1992) 261–272
11. Liang, Z., Verma, R.M.: Complexity of Checking Freshness of Cryptographic Protocols. Technical report, Computer Science Department, University of Houston, Texas, USA, `http://www.cs.uh.edu/preprint` (September 2008) UH-CS-08-14.
12. Song, D.X.: Athena: A new efficient automatic checker for security protocol analysis. In: CSFW. (1999) 192–202
13. Song, D.X., Berezin, S., Perrig, A.: Athena: A novel approach to efficient automatic security protocol analysis. Journal of Computer Security **9**(1/2) (2001) 47–74
14. Corin, R., Etalle, S., Saptawijaya, A.: 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) 155–168
15. Backes, M., Cortesi, A., Focardi, R., Maffei, M.: A Calculus of Challenges and Responses. In: Proceedings of 5th ACM Workshop on Formal Methods in Security Engineering (FMSE). (November 2007)
16. Guttman, J.D., Thayer, F.J.: Authentication tests. In: IEEE Symposium on Security and Privacy. (2000) 96–109
17. Froschle, S.: The insecurity problem: Tackling unbounded data. In: IEEE Computer Security Foundations Symposium 2007, IEEE Computer Society (2007) 370–384
18. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An np decision procedure for protocol insecurity with xor. Theor. Comput. Sci. **338**(1-3) (2005) 247–274
19. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6**(1-2) (1998) 85–128
20. Thayer, F.J., Herzog, J.C., Guttman, J.D.: Strand spaces: Proving security protocols correct. Journal of Computer Security **7**(1) (1999)
21. Lowe, G.: 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) 31
22. Syverson, P.F.: A taxonomy of replay attacks. In: CSFW. (1994) 187–191