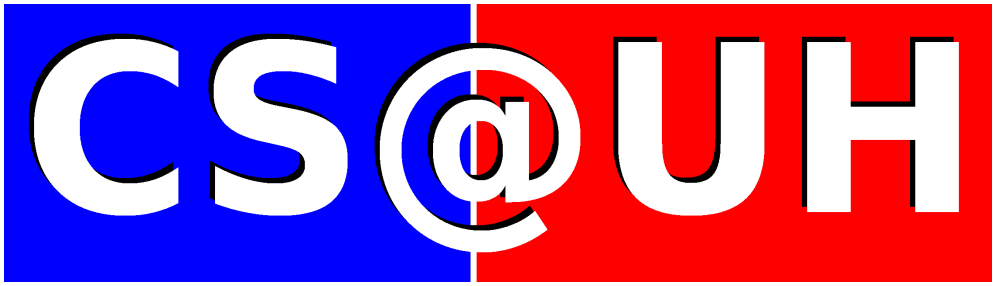

A Note on An NP-completeness Proof for Cryptographic Protocol Insecurity *

Zhiyao Liang and Rakesh M. Verma
zliang@cs.uh.edu, rmverma@cs.uh.edu

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

Technical Report Number UH-CS-08-15

October 05, 2008

Keywords: Cryptographic protocols, secrecy, NP-complete, formal method.

Abstract

This article discusses the paper "Protocol insecurity with a finite number of sessions and composed keys is NP-complete" [1]. Some understanding of the paper is recorded in this article. Especially a non-trivial error of the NP proof of [1] is presented, and we provide a solution to fix this error. We suggest that the NP-completeness proof can be improved in several aspects.

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

A Note on An NP-completeness Proof for Cryptographic Protocol Insecurity *

Zhiyao Liang and Rakesh M. Verma
zliang@cs.uh.edu, rmverma@cs.uh.edu

Abstract

This article discusses the paper "Protocol insecurity with a finite number of sessions and composed keys is NP-complete" [1]. Some understanding of the paper is recorded in this article. Especially a non-trivial error of the NP proof of [1] is presented, and we provide a solution to fix this error. We suggest that the NP-completeness proof can be improved in several aspects.

Index Terms

Cryptographic protocols, secrecy, NP-complete, formal method.

I. Overview

The paper "Protocol insecurity with a finite number of sessions and composed keys is NP-complete" [1], which can also be found at [2], proves the NP-completeness of checking secrecy in a bounded situation. We believe this is among the most important complexity results of checking cryptographic protocols. The approach of [1] is extended in a more recent paper [3], which is dedicated to prove that checking secrecy is NP when XOR operations are allowed. We are motivated to have a thorough understanding of the proofs of [1], not only for the interest in the theory of complexity of checking cryptographic protocols, but also because we may clarify and improve our understanding of the features a deterministic checker with possibly optimal performance. I.e., the proof of [1] discuss the notion of a normal attack, which is a simplest attack to a protocol, and for a deterministic checker, its performance of checking a protocol may not be any better than finding a simplest attack to the protocol.

In this section we demonstrate an important challenge faced by a NP proof of checking secrecy, which explains what the error of [1] is about, which is presented in details later, and we discuss the main idea of the NP proof of [1].

In order to prove that checking secrecy is NP-complete in a certain setting, the proof needs to show that the size of the solution guessed by the non-deterministic algorithm is bounded by a polynomial w.r.t. the protocol size, where size can be measured in some proper way, such as the number of bits of the protocol code. A guessed solution describes a possible run of the protocol, and the solution must describe a substitution of the variables in the protocol code, since in a run the variables of the protocol are instantiated by terms according to the substitution.

The following example shows that if a term is not represented using a special data structure called DAG, which is defined later in this paper, in an attack a variable may have an instance with of exponential size w.r.t. the protocol. Here are the notations for the example, which are similarly used by [1] except the notation for fresh nonce generation. $\#_A(X, Y)$ means that the agent A generates two fresh nonces X and Y. K_A means the public key of agent A. K^{-1} means the inverse key of an asymmetric key K. For example the inverse key of a public key is the corresponding private key. $\{T\}^s_M$ means a symmetric encryption where the encryption key is M. $\{T\}^p_{K_A}$ means an asymmetric encryption where K_A is the encryption key.

Suppose a protocol code is the communication sequence as follows.

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

$\#_{A_0}(X_0)$ $(A_0 \Rightarrow B_0) : A_0, B_0, \{1, e, X_0\}^s_k$.
$\#_{A_1}(X_1, Y_1)$ $(A_1 \Rightarrow B_1) : A_1, B_1, \{1, X_1, Y_1\}^s_k$.
$(B_1 \Rightarrow A_1) : B_1, A_1, \{2, X_1, [Y_1, Y_1]\}^s_k$.
...
$\#_{A_i}(X_i, Y_i)$ $(A_i \Rightarrow B_i) : A_i, B_i, \{i, X_i, Y_i\}^s_k$.
$(B_i \Rightarrow A_i) : B_i, A_i, \{i+1, X_i, [Y_i, Y_i]\}^s_k$.
...
$\#_{A_m}(X_m, Y_m)$ $(A_m \Rightarrow B_m) : A_m, B_m, \{m, X_m, Y_m\}^s_k$.
$(B_m \Rightarrow A_m) : B_m, A_m, \{m+1, X_m, [Y_m, Y_m]\}^s_k$.
$\#_A(X, Y)$ $(A \Rightarrow B) : \{\ A, B, X, \{m+1, e, Y\}^s_k\ \}^p_{K_B}$.
$\#_B(Secret)$ $(B \Rightarrow A) : \{B, A, Secret\}^s_X$.

We assume the attacker does not know the special key k, while the attacker knows the names and public keys of all agents. A special constant e appears in the messages between A_0 B_0, and between A B. It is obvious to prove that the only way for the attacker to know the term $Secret$ is by constructing the message $\{\ A, B, X, \{m+1, e, Y, A, B\}^s_k\ \}^p_{K_B}$, which requires that the attacker knows a term $\{m+1, e, Y\}^s_k$, call it T, for some integer m, $m > 0$. Integers are constants in the protocol. Note that e is a special constant which can not be generated as a nonce. The only way for the attacker to obtain T is by the following actions. First, obtain the term $\{1, e, X_0, A_0, B_0\}^s_k$ sent from A_0 to B_0, call it T_0. Then construct $Msg1 = A_1, B_1, T_0$. Send Msg_1 to B_1, and obtain the term $\{1, e, [X_0, X_0]\}^s_k$. Repeat this process m times, each time send T_i to B_i, for some i, $1 \leq i \leq m$. Finally T is obtained and the attacker can construct the term $\{\ A, B, X, \{m+1, e, Y\}^s_k\ \}^p_{K_B}$ where X is chosen by the attacker. Then the $Secret$ term is revealed to the attacker. It is obvious that in the attack finally the variable X is instantiated by e, and the term Y is instantiated by a huge term where X_0 appears 2^m times, which is obviously exponential w.r.t. the size of the protocol. However if terms are described as DAGs (introduced later) only $m+1$ nodes need to be used in the DAG that describe the instance of Y.

The error of [1] is in the proof to show the polynomial size of a substitution for a variable in a non-deterministic attack.

In [4] a NP-procedure is claimed. We do not focus on the work of [4] since we consider its NP proof is not comparable to the one of [1] for the following reasons.

- In [4] encryption keys must be atomic, while in [1] encryption keys are more generally allowed to be composed. Extending atomic keys to composed keys is a non-trivial task, as mentioned in [4], and we think the non-triviality can be explained as follows. By assuming atomic keys the attacker's computation to derive a message from a set of terms E can be organized in two stages, to analyze terms in E as much as possible and construct terms. But assuming composed keys this two-stage computation, which some proofs may rely on, is not sufficient. A composed key could be instantiated by any term, which makes the analysis seems to be much more complex.
- The proof in [4] of the NP result is rather sketchy, which is less than a page besides the procedure. It seems that the work of [4] for the NP result is on the level of arguing that using DAGs the NP-time result can be obtained (as mentioned by the abstract of [4]), and is not comparable to the detailed proving in [1]. In [4] several other problems beside the NP result are addressed, while the main focus of [1] is the NP proof.

In [5] there is a NP-complete result of checking secrecy with a bounded number of role instances. The proof of [5] assumes atomic key, and the size of each message in a run is assumed to be less than a certain number K. Bounding the message size makes the NP proof considerably easier, and the DAGs are not needed to represent terms.

The contributions of the paper can be highlighted as follows:

- We discuss the modeling of [1], and we address several important issues in order to clarify and improve the modeling, including role instances vs. sessions, specifying variables and constants of a protocol, the attacker's initial knowledge, and the definition of normal attack.
- We present an improved and clarified proof of Lemma 4, which is the most important lemma of [1] for proving the NP result. The improved proof of Lemma 4 uses several generally applicable observations and lemmas, which is powerful enough to prove other results.
- We prove that the attacker does not need to generate nonces.
- We point out an non-trivial error in the proof of Theorem 1 in [1], which is in the proof of the theorem for

the NP result, and show a counter-example to clarify the error.
- We have checked the related paper [3], which is more recent than [1], and understand that the error has not been fixed in [3].
- We provide a solution to fix the non-trivial error and thus prove the NP result of checking secrecy.
- We justify the linear bound on the derivation length for deriving a received message based on the messages sent earlier and the attacker's initial knowledge.

Note that for a observation, lemma, corollaries or theorem, if it is marked with a '*', then it is provided by this article, not by [1] or [3], otherwise it is stated [1] and is presented here with the exactly identical statement and with the same number.

II. The modeling and notations of [1]

In this section the notations and definitions of [1] are presented. Although these notations and definitions can be found in [1], they are included in this article for three reasons: 1) for the convenience of readers; 2) to discuss some notions of [1] and to explain them by common ideas of modeling a Dolev-Yao attacker; 3) more importantly these notations and definitions are used as the solid foundation to present the error, which is described in later sections.

An ***Atom*** is an atomic ground term (variable free), i.e., a constant. A *message* is a ground term (variable free), while a ***term*** can have variable. The asymmetric key must be an atomic term (belonging to the set $Keys$), while the symmetric key is allowed to be a composed term. Message and term are defined by the following grammars:

$$msg \ ::== \ Atoms| < msg, msg > |msg^p_{Keys}|msg^s_{msg}$$
$$term ::= Var|Atoms| < term, term > |term^p_{Keys}|term^s_{term}$$

Note that variables can appear in the descriptions of $Keys$, such as K_A. The $Keys$ appearing in the above grammar for msg can be more proper to be considered as the ground instantiations of keys such as K_a, which is an instance of K_A.

Variables are represented as x, x_1, x_2 and so on. In an execution (described later), variables are global i.e., all occurrences of x are substituted together by the same term. Constants are the names starting with uppercase letter. such as N_A. Note that in [6] uppercase letters are used for variables instead.

A ***protocol*** is specified by a set of rules:

$$\{(l, R_l \Rightarrow S_l)|l \in \varphi\}$$

.

Each rule is associated with an index l, and l has the form (A, i). φ is the set of rule indexes. Each principal, say A is associated with a set of numbers W_A, and two numbers i and j in W_A can be compared with the relationship $<_{W_A}$. Actually $<_{W_A}$ can be simply understood as the common $<$ relationship comparing natural numbers.

Note that the protocol can be considered as a set of roles. Each role is executed by a principal (same as agent). For a rule $(l, R_l \Rightarrow S_l)$ where $l = (A, 2)$, it means that in the second step of A's role, where A receives message R_l and then sends message S_l. The protocol specified in [1] is *role oriented*, not a *communication sequence*, and a protocol can be a set of *non-matching roles*, i.e., for a message received in a step of a role, the corresponding role where the message is sent could be missing. This issue of non-matching roles are explained in more details in [6].

There are cases that in a role an agent may send a message without needing to receive a messages earlier, which can be expressed using the modeling of [1] as rule where the received message R_i, for some $i > 0$, is an empty message.

A ***correct execution order*** π is a one-to-one mapping $\pi : \varphi \rightarrow \{1, \ldots, |\varphi|_{card}\}$, where φ is the set of rule indexes of the protocol, and $|\varphi|_{card}$ is the number of elements (cardinality) of φ, which is the number of rules of the protocol specification. Note that in [1] $|\varphi|$ is used to represent the cardinality of φ. But later the same notation $|.|$ is used for a different purpose to define the size of a term; therefore, in this article we use $|S|_{card}$ to indicate the cardinality of a set S. For all $A \in Names$ and $i <_{W_A} j$, it must be true that $\pi(A, i) < \pi(A, j)$. A ***protocol execution*** is given by a ground substitution σ, a correct execution order π and a sequence of environments (an environment is a set of terms) $E_0, \ldots, E_{|\varphi|}$, that satisfy the following conditions.

- The message received in the h^{th} step in the execution is included in the previous environments, denoted as $R_{\pi^{-1}(h)}\sigma \in E_{h-1}$.
- The message sent in h^{th} step in the execution is included in the later h^{th} environment, denoted as $S_{\pi^{-1}(h)}\sigma \in E_h$.

Note that $\sigma(T)$ and $T\sigma$ are equivalent and they mean to apply the substitution σ to a term T. It obvious that E_{h-1} denotes the set of terms that attacker can obtain before the message is received in the h^{th} step of the execution. It

is the same as the common idea that the attacker should be able to obtain a message before it is received, and the attacker records all messages sent in the network.

The discussion on the notion of session is important and is included in the next subsection.

The following rewrites can be applied to a set of terms, by replacing the terms on LHS with the set of terms on the RHS. These rules describe the attacker's internal computation to analyze and construct terms. Note that the attacker can also generate fresh nonces. We believe that in [1] the authors assume that the attacker have generated all of the nonces needed before the start of a protocol run, and these nonces are included in the attacker's initial knowledge S_0.

Decomposition rules:

- $L_d(\langle a, b\rangle): \ \langle a, b\rangle \rightarrow a, b, \langle a, b\rangle$.
- $L_d(\{a\}_K^p): \ \{a\}_K^P, K^{-1} \rightarrow \{a\}_K^P, K^{-1}, a$.
- $L_d(\{a\{_b^s) : \{a\}_b^s, b \rightarrow \{a\}_b^s, b, a$

Composition rules:

- $L_c(\langle a, b\rangle): \ a, b \rightarrow a, b, \langle a, b\rangle$
- $L_c(\{a\}_K^P): \ a, K \rightarrow a, K, \{a\}_K^P$
- $L_c(\{a\}_b^s) : a, b \rightarrow a, b, \{a\}_b^s$

For a rewrite rule $L_d(T)$ or $L_c(T)$, T is called the ***principle term*** of the rule. The rewrite relation between two set of terms M and M', denoted as $M \rightarrow M'$ means there exists one of the rule $l \rightarrow r$ described above such that l is a subset of M and M' is obtained by replacing l by r in M. $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$.

Definition 1 (forge). Let E be a set of terms and let t be a term such that there is E' with $E \rightarrow^* E'$ and $t \in E'$. Then we say that t is forged from E and it is denoted as $t \in forge(E)$.

A derivation is implicitly defined in [1]. It is better to explicitly define it. A **derivation** of a set of terms E_0 is a sequence $E_0 \rightarrow_{r_1} E_1 \rightarrow_{r_2} \cdots \rightarrow_{r_n} E_n$, where E_i for $1 \leq i \leq n$ is a set of terms, and r_i is an application of a rewrite rule described above. If r_i, for $1 \leq i \leq n$, has the form $LHS_i \rightarrow RHS_i$, for two sets of terms LHS_i and RHS_i, then it must be true that $LHS_i \in E_{i-1}$ and $E_i = E_{i-1} \cup RHS_i$.

Throughout this paper, when there is no confusion comma is used as as the union of two sets. For example for a set of terms E and a term t, $E \cup \{t\}$ can be represented as E, t. For a set of terms E', $E \cup E'$ can be represented as E, E'. For two terms t and t', $\{t, t'\}$ can be represented as t, t'.

Definition 2 (attack). Given a protocol $P = \{R'_l \Rightarrow S'_l | l \in \varphi\}$, a secret message $Secret$ and assuming the intruder has initial knowledge S_0, an ***attack*** is described by a ground substitution σ and a correct execution order $\pi : \varphi \rightarrow 1, \ldots, k$, where $k = |\varphi|_{card}$, such that for all $i = 1, \ldots, k$, we have $R_i\sigma \in forge(S_0, S_1\sigma, \ldots, S_{i-1}\sigma)$ and $Secret \in forge(S_0, S_1\sigma, \ldots, S_k\sigma)$ where $R_i = R'_{\pi^{-1}(i)}$ and $S_i = S'_{\pi^{-1}(i)}$.

Here is some explanation. At the i^{th} step of the execution a rule is executed. Suppose the rule has index l, i.e., the rule is $R'_l \Rightarrow S'_l$, then R_i is R'_l and S_i is S'_l.. The relationship between i and l is represented by $i = \pi(l)$ and $l = \pi^{-1}(i)$.

Note that since a correct execution order is defined by a substitution and a sequence of environments, we should clarify the above definition of attack by specify the sequence of environments, from E_0 to E_k. Let $\mathcal{E}_0 = S_0$, where S_0 is a set of atoms describing the attacker's initial knowledge, which will be discussed later in Section III-B. Let $\mathcal{E}_i = \{S_0, S_1\sigma, \cdots S_i\sigma\}$ for $0 \leq i \leq k$. Then for the environment E_i, for $0 \leq i \leq k$, $E_i = forge(\mathcal{E}_i)$.

We consider it is better to directly introduce the definition of a protocol execution after the attacker's behavior $forge()$ is introduced, and the discussion of the environments are avoided. Then an attack is a protocol execution such that $Secret \in forge(\mathcal{E}_k)$.

The ***size*** of a message term t is denoted $|t|$ and defined as:

- $|t| = 1$ for any $t \in Atoms$, except for $Charlie$, which is the reserved name of the attacker, where $|Charlie| = 0$.
- recursively, $|\langle x, y\rangle| = |\{x\}_y| = |x| + |y| + 1$.

Given a multiset M of terms, the multiset value $M(t)$ for a term t is 0 $t \notin M$, otherwise $M(t)$ is the number of copies of t in M.

The ordering $M \gg N$ between two multisets M and N can be defined in the following two equivalent ways.

- $M \cup \{s\} \gg N \cup \{t_1, \ldots, t_n\}$ if $M = N$ and $s > t_i$ for all $i \in 1, \ldots, n$.
- $M \gg N$ if (i) $M \neq N$ and (ii) whenever $N(x) > M(x)$ then $M(y) > N(y)$ for some $y > x$.

Definition 3 (normal attack). Given a protocol $P = \{R'_l \Rightarrow S'_l | l \in \varphi\}$, an attack (σ, π) is *normal* if the multiset of non-negative integers $\{|R_1\sigma|, \ldots, |R_k\sigma|\}$ is minimal, with $R_i = R'_{\pi^{-1}(i)}$ and $S_i = S'_{\pi^{-1}(i)}$.

Note that the minimal size is only defined on the received messages.

The set of ***subterms*** of a term is defined in the usual way. Note that for an encryption its encryption key is its subterm. Given a set of terms E, the subterms of all of the terms in E is indicated as $Sub(E)$. For a term t, $Sub(t) = Sub(\{t\})$.

The ***DAG*** representation of a set E of message terms is the graph $(\mathcal{V}, \mathcal{E})$ with labeled edges, where:

- the set of vertexes $\mathcal{V} = Sub(E)$, the set of subterms of E,
- the set of edges $\mathcal{E} = \{v_s \rightarrow_{left} v_e | \exists b, v_s = \{v_e\}_b^{s/p}$ or $v_s = \langle v_e, b\rangle\} \cup \{v_s \rightarrow_{right} v_e | \exists b, v_s = \{b\}_{v_e}^{s/p}$ or $v_s = \langle b, v_e\rangle\}$.

Remark 1 The DAG-representation is unique.

Suppose there are n subterms of a set of messages E, then there are at most n nodes and $2.n$ edges in the DAG representation. The ***DAG-size*** of E is denoted by $|E|_{DAG}$, to be the number of distinct subterms of E. For a term t, we simply write $|t|_{DAG}$ for $|\{t\}|_{DAG}$.

The set of variables that are the subterms of a term t is denoted as $Var(t)$, similarly $Var(\mathcal{P})$ means the set of variables appearing as subterms in the protocol $\mathcal{P}$.

However the above definition of normal attack is unnecessarily complex. In [3] a simpler definition of normal attack is provided, which uses only DAG size of terms, not the complex multi-sets. It is seems the definition of normal attack is only used in the proof of Lemma 4 (showed later), an the simplified definition of normal attack can used to clarify the proof of Lemma 4.

Definition 1:* (new normal attack by [3]). Given a protocol $P = \{R'_l \Rightarrow S'_l | l \in \varphi\}$, an attack (σ, π) is *normal* if $\Sigma_{x \in Var(\mathcal{P})} |\sigma(x)|_{DAG}$ is minimal.

III. Modeling Discussion

Modeling is the foundation of proofs. When the modeling is improved, correspondingly the result of the proof is improved.

A. Sessions Versus Role Instances

We think modeling a protocol as a set of roles, where each role is a sequence of steps of sending or receiving messages, is more direct and convenient than modeling a protocol as a sequence of rules, where the indexes of the rules are designed to show that the rules that are executed by the same agent must follow a certain relative order in an execution. We can observe that a rule always combines a message receiving step and a message sending step, but there are protocols where an agent may receive several messages before sending one.

The notion of ***session*** deserves special attention. A session is an execution of the protocol. In [1] π is defined as a one-to-one map, which means that in an execution the number of steps must be the same as the number of rules of the protocol, and each rule of the protocol must appear in an execution exactly once. But it is counter-intuitive, and it may have some trouble to define an attack. There are attacks that do not need to execute all rules of the protocol. In the more recent paper [3] the definition of an execution (a session) is improved, so that only a subset of the rules of the protocol may be executed in an execution, and a rule of the protocol can be executed at most once in an execution. Note that if a rule with index (A, j) is executed in an execution, all rules with the index (A, i), where $i \leq j$, are also executed in the execution.

Since a role can be considered as the sequence of steps executed by the same agent, say A, as described by a protocol, we can equivalently define a session as an interleaving of the steps of a set of role instances, while the relative orders of the steps within a role instance are kept in the session, with a special condition that every role of the protocol can have at most one role instance in a session. This proper interpretation of session used in [1] is also mentioned in [7].

Considering a bound on the number of role instances (NRI) instead of a bound of the number of sessions is a better choice to discuss the complexity of checking protocols, for the following two reasons.

- Sessions do not have practical meaning, compared to the notion of role instances. The notion of a session is global across different role instances, as if there is some logic control over, or some logic connection between, the role instances so that these role instances are grouped into a session, while every role must have (or can at

most have) one role instance in this group. However in practice this logical connection between role instances to form a session is usually not possible. Unless there is some term working as the ID of a session, there is no clear way to group role instances into a session.

- On the other hand, to organize the actions of sending and receiving messages into a role instance is a built-in notion of the setting of the problem. Role instance is based on the intention of a regular agent. For a regular agent A, when A executes some actions of receiving or sending messages, A has a clear intention of whether the different actions should belong to the same or different role instances. However A's intension can only organize the actions of her own, and cannot effect B's intension. The connection between A and B's intention to form a session usually is not clear. That is why in the majority of the researches of checking protocols, including [5] [8] [9], the notion of sessions is not used. When we try to find an attack of a protocol, usually we describe (and think about) the attack as a set of role instances, instead of sessions.

We think the NP result of checking protocols assuming a bound on sessions does not directly imply, or is implied by, the corresponding complexity result of assuming a bound on role instances.

Suppose we want to reduce a problem, call it problem one, of checking secrecy of a protocol Pro with K role instances, to a problem, call it problem two, of checking secrecy of Pro with K' sessions, for some K', we think the reduction is not obvious. Problem one implies that there are no more than K sessions in a run, since each session introduces at least one role instance. But if we choose $K' = K$, the reduction will not work, since K sessions could allow $K \times R$ role instances, where R is the number of roles in pro, and then the if-and-only-if property of the reduction will not be satisfied, i.e., maybe there is any attack with $K \times R$ role instances, but there is no attack with K role instances.

Suppose we want to reduce problem two to problem one, the reduction is not obvious neither. problem two implies that there are no more than $K' \times R$ role instances, where R is the number of roles in Pro, since each session will have no more than R role instances. However if we choose $K = K' \times R$, the if-and-only-if property of the reduction will not be satisfied, since $K' \times R$ role instances can possibly allow $K' \times R$ sessions, not K' sessions, where each session introduce just one role instance.

Since considering role instances is more direct and convenient than considering sessions in order to describe an attack, we believe the NP result of checking protocols with bounded role instances is more useful.

B. The Attacker's Initial Knowledge

During an execution there is no nonce generated by the attacker or by any agent. All of the nonces generated by regular agents in a session is assumed to be initially known to the regular agents, which is no problem since the total number of nonces generated regular agents in a session is no more than the DAG size of the protocol. However if we similarly consider that all nonces that are generated by the attacker by a normal attack are included in the set S_0, which is considered as the set of terms initially known to the attacker, then the proof may not obviously work unless there is a bound on the total number of nonces generated by the attack and the bound has to be polynomial in the DAG size of the protocol. S_0 is not specified in the proofs. How to specify and bound S_0?

This question seems to be answered at the end of [1], in the conclusion section. The authors of [1] have a *statement* that in a normal attack the attacker does not need to generate any nonce at all, since for any attack, if the nonces generated by the attacker are replaced with the attacker's name $Charlie$, it is also an attack. We believe this statement is true since normally a regular agent do not explicitly check the disequality among nonces.

Note that if the attacker does not generate nonces, and the attacker simply use different levels of encryptions of its name, say I or $Charlie$, to simulate different nonces, then if these terms are represented using DAG, it does not introduce any more cost than actually generating different nonces, since the total number of subterms in a run by the two approach DAG size is the same.

Although we believe the statement is intuitively true, but to prove it formally may still need some arguments. We proved this statement in Section VII. We think if a proof can directly allow the attacker to generate nonces it will have better presentation and be more convincing, and may be more generally applicable for other problems.

Based on this statement, we can consider that the initial knowledge of the attacker assumed in [1] includes the agent names and public keys, which can be accepted as a part of input of the problem.

For the cases that disequality tests are allowed to compare two nonces in a role, as addressed in [5], the statement is not true and the analysis of checking secrecy may have to consider the nonces generated by the attacker. The

complexity result is not obtained for the *problem* of checking secrecy in the setting with a bounded number role instances, unbounded message size, and with disequality check operations allowed. In [5] it is showed that checking secrecy is NP-complete with bounded role instances, bounded message size, with disequality check allowed, and without composed keys, which is a setting different from the *problem* described above.

C. Specification of Variables and Constants of a Protocol

According to the example of protocol specification provided in [1], the variables and constants are specified by the following way. In the rules of an agent A, i.e. in A's role, the atomic terms that first appear in some received messages are others-chosen, and they are represented by variables except the atomic term that represents A's name or those specifically indicated as constants. Other atomic terms are self-chosen by A, and they are represented by atoms (constants). This self-chosen and others-chosen idea is further clarified in [3] where it is clearly stated that variables first appear in some received messages of A's rules. Note that the nonces created by A are always represented by unique constants. Different roles have different variables so that the instantiations of the variables in one role are independent to the instantiations of the variables of other roles. The process to assign variables and constants to the roles (a role is the rules executed by the same agent) is called skolemization.

There is one question about the choice of agent constants. In the protocol for the proof of the setting of one session, in A's rule, A is replaced with a constant a, and in B's rules, B is replaced with a constant b. Maybe it makes sense to assign different agent names to different roles since usually a session, which should be similarly understood as a communication sequence, agents communicate with each other and these agents are mutually different. However, the proof of n sessions, $n \geq 2$, is by consider n sessions as one single larger session, and the A's rules may have at most n copies in the larger session, but each copy is considered independently, i.e. as a separate role. The question is should the n copies of A associated with n different agent constants? It is possible that choosing, or not choosing, different agent constants for the n copies of A's rules in the larger session is irrelevant to the correctness of checking or not, but it is not clear to prove.

If we describe the question by the modeling using role instances, it could be like: if there are n role instances of A's role, should the checker consider all these n role instances be executed by different agents, or don't have to? Although the question does not get any easier by stating using role instances, but it seems the statement is more clarified and with less technical clutter. The researches of [10] and [11] are relevant to the answer of this question.

D. Possible Typos of Notations and Modeling

At the bottom of page 457, the for $\pi(1) = (A, 1)$, it is a wrong usage of π, it should be $\pi^{-1}(1) = (A, 1)$.

At the top of page 463: "We define $R_i = R'_{\pi^{-1}}$ and $S_i = S'_{\pi^{-1}}$ " should be $R_i = R'_{\pi^{-1}(i)}$ and $S_i = S'_{\pi^{-1}(i)}$

A possible mistake is that for a protocol with $|\varphi|_{card}$ rules, an execution (of one session) of the protocol should not be defined with a rigid number of steps of $|\varphi|_{card}$. Suppose the protocol has 5 rules, there may be an attack needing only two steps, but maybe it is impossible to include the other three steps in the attack, in order to make 5 steps in the execution, since there may be no way for the attacker to construct the received messages due to the fact the protocol can be non-matching roles. In [3] this consideration is improved by allowing the rules be executed in a session be $\leq |\varphi|_{card}$.

The overline notation $\overline{U}$ is defined and used in the proof of Theorem 1 in page 464. But when it is defined, the line over U is rather small, looks different from the over line notation later used.

IV. Tree of Term and Substitution

In order to clarify the proofs of later sections, in this section we define the tree of a term and its relationship with substitution. The discussions of this section are not provided in [1].

Every term corresponds to a ***tree***, which is a unique representation of the term, where each node of the tree is associated with a ***mark*** and each non-leaf node has links to children nodes, as follows.

- The tree of an atomic term t is a single node marked with t, which is also the top node of the tree.
- The tree of a pair $\langle t_1, t_2 \rangle$, has the top node nd with the mark $\langle\rangle$, and there are 2 links leaving from nd arranged from left to right, and the j^{th} link, $j \in \{1, 2\}$ points to the top node of the tree of t_j.
- The tree of an encryption of the form $\{t_1\}^s_{t_2}$ has the top node nd labeled with $\{\}^s$ where the left link of nd points to the top node of the tree of t_1, and the right link of n points to the top node of the tree of t_2.

- The tree of an encryption of the form $\{t_1\}^p_{t_2}$ has the top node nd labeled with $\{\}^p$ where the left link of nd points to the top node of the tree of t_1, and the right link of nd points to the top node of the tree of t_2.

If there is a link pointing from node nd to node nd', then nd is the ***parent node*** of nd', and nd' is a ***child node*** of nd. If there is a path pointing from node nd to node nd' in a tree, then nd is an ***ancestor node*** of nd' and nd' is a ***descendant node*** of nd. Every node in a term tree is associated with a ***depth***, which is a non-negative integer, as follows: the top node has the depth 0, and for a node with depth j, all of its children nodes have the depth $j+1$.

We can obviously determine the unique term represented by a tree based on the structure and the marks of the nodes of the tree; therefore, we can also assign labels to the nodes of a tree, and the ***label*** of a node nd is the term represented by the tree whose top node is nd.

The following observation on the labels of the nodes of a tree is obviously tree.

- The tree of a pair $\langle X_1, X_2\rangle$, has the top node nd labeled with $\langle X_{,1}, X_2\rangle$, where there are 2 links leaving from n arranged from left to right, and the j^{th} link, $j \in \{1,2\}$, points to a node labeled with X_j.
- The tree of $\{X\}^p_Y$ (same for $\{X\}^s_Y$) has the top node nd labeled with $\{X\}^p_Y$ (with $\{X\}^s_Y$), where the left link of nd points to a node labeled with X and the right link of nd points to a node labeled with Y.

We can assign locations to the nodes in the tree of a term T in order to identify them, as follows.

- The top node of the tree of term T has the location 0.
- If the top node nd of a tree of term $\langle t_1, t_2\rangle$ has location L, then the j^{th} child node of nd, which is the top node of a tree of t_j, $1 \le j \le 2$, has the location $L.j$.
- If the top node nd of a tree of term $\{t_1\}^{\leftrightarrow}_{t_2}$ or $\{t_1\}^{\rightarrow}_{t_2}$ has location L, then the left child node of nd, which is the top node of a tree of t_1, has the location $L.\alpha$, and the right child node of nd, which is the top node of a tree of t_2, has the location $L.\beta$.

A node nd is an ancestor of a node nd' if the location of nd is a prefix of the location of nd'.

A ***substitution*** σ of the form $[t \leftarrow t']$ for two terms t and t' is function such that given a term T, $\sigma(T)$ represents a term calculated by the following rules.

- If T is t, then $\sigma(T) = t'$.
- If $T = \langle t_1, t_2\rangle$, then $\sigma(T) = \langle \sigma(t_1), \sigma(t_2)\rangle$.
- If $T = \{X\}^p_Y$, then $\sigma(T) = \{\sigma(X)\}^p_{\sigma(Y)}$.
- If $T = \{X\}^s_Y$, then $\sigma(T) = \{\sigma(X)\}^s_{\sigma(Y)}$.
- If $T = k_X$, where $i \in \{0,1\}$, then $\sigma(T) = k_{\sigma(X)}$.

Then the definition of a substitution can be extended to $[t_1 \leftarrow t'_1; t_2 \leftarrow t'_2; \cdots]$. Note that a substitution defined here allows t and t' to be any term, ground or not ground, while the substitution associated with a role instance in a protocol run only replaces variables with ground terms.

For a term g, and a substitution $\sigma = [t \leftarrow t']$, for two (general) terms t and t', $\sigma(g)$ represent a term obtained by replacing every occurrence of t in g with t'. More exactly, we can understand that $\sigma(g)$ represents a term whose tree is formed by replacing every subtree, which labeled with t in the tree of g, with a tree of t'. For a set of terms E, $\sigma(E)$ means $\{\sigma(g) | g \in E\}$. The meaning of applying a substitution to a protocol or an action step (defined below) is obvious.

The following observations on trees of terms and substitutions can obviously be justified.

1) A node in a tree is labeled or marked by an atomic term if and only if the node is a leaf in the tree.
2) Every node in the tree of a term T is labeled with a subterm of T.
3) If σ is a substitution replacing only atomic terms, then for a term T, if T is a symmetric encryption or an asymmetric encryption or a pair, then $\sigma(T)$ must be a symmetric encryption or an asymmetric encryption or a pair respectively.
4) For a substitution σ replacing only atomic terms, and a term T, and a node nd at location L in the tree of T labeled with t, for some $t \in Sub(T)$, there is a node nd' in the tree of $\sigma(T)$ at same location L labeled with $\sigma(t)$.
5) For a substitution σ replacing only atomic terms, and a term T, for a non-leave node nd at location L in the tree of T, there is a node nd' in the tree of $\sigma(T)$ at same location L, such that the mark of nd and the mark of nd' are the same.

If a term X occur in T in several positions, then each occurrence of X in T is associated with a different node that is labeled with X. In contrast, saying that X is a subterm of T does address the different occurrences of X in T.

V. Results Before Theorem 1

We present the lemmas, propositions, and corollaries before Theorem 1. They will be used to prove Theorem 1, and to validate the error that is showed later, i.e., we should verify that the error does not violate any of these results, and we believe these results are correct. All of the lemmas and corollaries are exactly quoted from [1], while we only describe the proof ideas.

Lemma 1. For all set of terms E, for all variables x and for all messages t, we have: $|E[x \leftarrow t]|_{DAG} \leqslant |E, t|_{DAG}$.

Note that $[x \leftarrow t]$ means a substitution to replace the variable x in E with a ground term t, while all other variables in E remains the same.

Lemma 1 is very intuitive. $|E[x \leftarrow t]|_{DAG}$ can be understood as the sum or two parts. Part 1, The DAG size of removing one atomic term x from $sub(E)$. Part 2, the DAG size of (the number of subterms of) t. Surely the sum of this two parts is $\leqslant |E, t|_{DAG}$.

Corollary 1 For all set of terms E, for all ground substitutions γ, we have $|E\gamma|_{DAG} \leqslant |E, \gamma(x_1), \ldots \gamma(x_k)|_{DAG}$ where $\{x_1, \ldots, x_k\}$ is the set of variables in Var. (recall that Var is the finite set of variables in the protocol).

Corollary 1 can be proved by using Lemma 1 to each variable, one by one.

Remark 2. We only need polynomial time to check that a rule $l \rightarrow l, r'$ can be applied to E and to compute the resulting DAG of E', when we have already a DAG-representation of E, l and r'. This is due to the fact that we only need to first check that all terms in l are also in E and then to compute the DAG-representation E' of E, r'.

In the above remark $E' = E, r'$, where E, r' means $E \cup r'$ for a set of terms r'. Sometimes for a set E and an element e, $E \cup \{e\}$ is represented as E, e.

Definition 4. Given a derivation $D = E_0 \rightarrow_{R_1} E_1 \rightarrow_{R_2} \cdots \rightarrow_{R_n} E_n$, a term t is a goal of D if $t \in E_n$ and $t \notin E_{n-1}$.

Definition 5. We denote $Deriv_t E$ a derivation (normal derivation) of minimal length among the derivations from E with goal t (chosen arbitrarily among the possible ones).

Note that in Definition 5, a normal derivation has minimal number of rewrite rules applied.

Lemma 2 If there exists t' such that $L_d(t') \in Deriv_t(E)$ then t' is a subterm of E.

Lemma 3 If there exists t' such that $L_c(t') \in Deriv_t(E)$ then t' is a subterm of $\{t\} \cup E$.

Lemma 2 and Lemma 3 are based on the intuitions that the attacker never needs to construct a term and then later decompose it, or the other way. Another idea is that every term constructed by the attacker must be used in some way. Note that no nonce is generated in a run. All nonces generated by regular agents or by the attacker are considered to be initially generated. Allowing the nonces to be generated in the middle of a run seems to be more proper.

Proposition 1. For any set of terms E and for any term t, if $Deriv_t(E) = E \rightarrow_{L_1} E_1 \cdots \rightarrow_{L_n} E_n$ then $n \leqslant |t, E|_{DAG}$ and for all $1 \leqslant i \leqslant n$, $|E_i|_{DAG} \leqslant |t, E|_{DAG}$.

Proposition 2. Let $t \in forge(E)$ and $\gamma \in forge(E)$ be given with $Deriv_\gamma(E)$ ending with an application of a rule in L_c. Then there is a derivation D with goal t starting from E and verifying $L_d(\gamma) \notin D$.

Note that Proposition 2 does not require that the derivation D with goal t is normal.

Now we clarify the notations of P, $\mathcal{P}$ and $\mathcal{SP}$.

- P represents a protocol and $P = \{R'_l \Rightarrow S'_l | l \in \varphi\}$. Remind that $R'_l \Rightarrow S'_l$ is a rule of the protocol and l is the index of the rule.
- $\mathcal{P}$ represents a set of terms including the attacker's initial knowledge plus the messages sent and received by the rules of P executed in a session, and the rules are numbered by their order appearing in the session.
$$\mathcal{P} = \{R_j | j = 1, \cdots k\} \cup \{S_j | j = 0, \cdots k\},$$
where k is the number of rules applied in the session, and $k \leq |\varphi|_{card}$. Here S_0 is the attacker's initial knowledge. R_i and S_i means that the i^{th} rule applied in the session has the form $R'_{\pi^{-1}(i)} \Rightarrow S'_{\pi^{-1}(i)}$, for $1 \leq i \leq k$. Note that the terms in $\mathcal{P}$ have variables.
- $\mathcal{SP}$ means the set of subterms of the terms in $\mathcal{P}$. So $\mathcal{SP} = Sub(\mathcal{P})$. In our proofs we will use the notation $Sub(\mathcal{P})$ more often since it is more intuitive.
- For $0 \leq i \leq k$, $\mathcal{P}_{\leq i} = \{R_j | j = 1, \ldots, i\} \cup \{S_j | j = 0, \ldots, i\}$.
- $\sigma(\mathcal{P}) = \{\sigma(R_j) | j = 1, \cdots k\} \cup \{\sigma(S_j) | j = 0, \cdots k\}$. Note that $\sigma(S_0) = S_0$. Similarly for $0 \leq i \leq k$, $\sigma(\mathcal{P}_{\leq i}) = \{\sigma(R_j) | j = 1, \cdots k\} \cup \{\sigma(S_j) | j = 0, \cdots i\}$. $\sigma(\mathcal{P}_{<i})$ is defined similarly.

We avoid using the notation $\mathcal{SP}_{\leq i}$, which in [1] means $Sub(\sigma(\mathcal{P}_{\leq i}))$. Instead we directly use $Sub(\sigma(\mathcal{P}_{\leq i}))$ to avoid confusion.

Definition 6. Let t and t' be two terms and θ a ground substitution. Then t is a θ-match of t' if t is not a variable and $t\theta = t'$. This will be denoted by $t \sqsubseteq_\theta t'$.

Lemma 4. Given a normal attack σ for all variable x, there exists $t \sqsubseteq_\sigma \sigma(x)$ such that $t \in \mathcal{SP}$.

We provide an improved proof of Lemma 4 in next section.

VI. An Improved Proof of Lemma 4

Lemma 4 is the most important result of [1] and [3] in order to prove their NP results. We think the proof of Lemma 4 in [1] can be simplified and clarified in some aspects.

In [1] and [3], the proof of the result of Lemma 4 is by contradiction. Suppose Lemma 4 is not true, then there is a variable x that violates Lemma 4. Then in a normal attack if we replace $\sigma(x)$ with a special term u, such that the size (depending on the definition of term size) of u is smaller than the size of x, then after the substitution the normal attack is still an attack but with strictly smaller size, which is contradictory to the fact that a normal attack is defined to be the attack with smallest size.

In the proof of Lemma 4 in [1] the special term u is chosen as the attacker's name $Charlie$, and the term size is defined based on multi-set and is different from DAG size, and the term size of $Charlie$ is specially defined as 0 which is less than the size of any term. We consider the issue of replacing x by u can be improved in two aspects. First the size of a term can be directly defined as its DAG size. Second, we will show that for the x that we assume violates Lemma 4, $\sigma(x)$ is a composed term. Therefore we can choose any atomic term that is initially known to the attacker as u, and then it is guaranteed that $|u|_{DAG} = 1 < |\sigma(x)|_{DAG}$, and there is no need to specify the special term size of $Charlie$, which seems to be awkward. More details are showed later.

The proof process of Lemma 4 in [1] is to define a substitution δ replacing a ground term $\sigma(x)$ by $Charlie$, and then prove that $\delta(\sigma(R_i)) \in forge(S_0, \delta(\sigma(S_1)), \cdots, \delta(\sigma(S_{i-1}))$. We think although the argument is intuitive, it may need further justification. Only showing $\delta(\sigma(R_i)) \in forge(S_0, \delta(\sigma(S_1)), \cdots, \delta(\sigma(S_{i-1}))$ may not be sufficient to show that it is still an attack. The following questions need to be answered.

- A message in an execution is formed by apply a substitution σ, which replaces variables with ground terms, to a message template Msg of the protocol, which is R_i or S_i for $1 \leq i \leq k$. But if we apply a substitution δ, which replace a ground term with another ground term, to $\sigma(Msg)$, the result message $\delta(\sigma(Msg))$ may not be allowed in an execution, i.e. maybe it is impossible to have a substitution σ', that replaces variables with ground terms, and that $\delta(\sigma(Msg)) = \sigma'(Msg)$.
- There is a claim in the proof Lemma 4 in [1]. Why this claim is needed? Can the key steps of the proof of Lemma 4 be abstracted out into separate observations, so that the observations can be more generally used to prove other results?

By the above motivations we provide a different proof of Lemma 4. We prove Observation* 1, 2, 3, and 4, and and Lemma* 1, which are useful to prove Lemma 4. Note that the observations, lemmas, corollaries and theorems marked with a '*' are provided by this article, not by [1] or [3]. The improved proof avoids some error and is general enough to prove another important result that the attack needs not to generate nonces, discussed in next section.

Given a term T, define $Atoms(T) = \{t \mid t \in Sub(T)$ and t is an atomic and ground term $\}$. Note that atoms means ground atomic terms in [1]. We can also call an atom as a **constant**.

Observation* 1: Given a set of terms E_0, and a derivation $D = E_0 \rightarrow_{r_1} E_1 \rightarrow_{r_2} \cdots \rightarrow_{r_h} E_h$, for some $h > 0$, for a term t such that $t \notin sub(E_0)$, and $t \in E_h$, then $t \in forge(E_0)$. Furthermore, there is a step in D using a compose derivation rule of the form $L_c(t)$.

Proof: This observation can be obviously proved by the induction of steps of the derivation. Only compose rules can generate new subterms, and each compose rule application, say r_j, $j > 0$, may generate at most one new subterm that is not included in $Sub(E_{j-1})$. ■

Observation* 2: Given a set of terms E and a term t such that $t \in forge(E)$, it is true that $Atoms(t) \subseteq Atoms(E)$.

Proof: There is a derivation $D = E_0 \rightarrow_{L_1} E_1 \rightarrow_{L_2} \cdots E_h$, starting from $E_0 = E$ with the goal t and $t \in E_h$ for $h \geq 0$. We can also prove by induction on the steps of D to show that for each E_i, $0 \leq i \leq h$, it is true that $Atoms(E_i) \subseteq Atoms(E)$. Then since $t \in E_h$ it is true that $Atoms(t) \subseteq Atoms(E)$. ■

Observation* 3: For a protocol P, and an execution of P with a substitution σ, and the set of terms $\mathcal{P}$ defined earlier for the session, it is true that for each variable x in $Var(\mathcal{P})$, $Atoms(\sigma(x)) \subseteq Atoms(\mathcal{P})$.

Proof: The attacker's initial knowledge S_0 is included in $\mathcal{P}$. S_0 includes all nonces that can be generated by the intruder, and every term in S_0 is an atom. It is obvious that $Atoms(\sigma(S_0)) = Atoms(S_0) \subseteq Atoms(\mathcal{P})$.

Then we can prove this observation by induction to show that for each step number i, $1 \leq i \leq k$, it is true that $Atoms(\sigma(R_i)) \subseteq Atoms(\mathcal{P})$ and $Atoms(\sigma(S_i)) \subseteq Atoms(\mathcal{P})$.

Base case: $i = 1$. Since $\sigma(R_1) \in forge(S_0)$, by Observation* 2, $Atoms(\sigma(R_1)) \subseteq Atoms(S_0)$. Since $Atoms(S_0) \subseteq Atoms(\mathcal{P})$, $Atoms(\sigma(R_1)) \subseteq Atoms(\mathcal{P})$. Since in a protocol all variables in a sent message must appear at some received message first, and by the definition of an execution the first executed rule in the session must be the first rule of some agent, i.e. $\pi^{-1}(1) = (A, 1)$ for some agent A, it must be true that $Var(S_1) \subseteq Var(R_1)$. $Atoms(S_1) = Atoms(\sigma(Var(S_1))) \cup Atoms(S_1) \leq Atoms(\mathcal{P}) \cup Atoms(S_1) = Atoms(\mathcal{P})$.

Induction step: For $i > 1$, suppose for all j such that $1 \leq j < i$, it is true that $Atoms(\sigma(R_j)) \subseteq Atoms(\mathcal{P})$ and $Atoms(\sigma(S_j)) \subseteq Atoms(\mathcal{P})$. Let $\mathcal{S}_{\leq j}$ be defined as $S_0 \cup \{S_1\} \cdots \cup \{S_j\}$. By the induction hypothesis, it is obvious that $Atoms(\sigma(\mathcal{S}_{\leq j}) \subseteq Atoms(\mathcal{P})$, for $1 \leq j < i$. Since $\sigma(R_i) \in forge(\sigma(\mathcal{S}_{\leq i}))$, and by the induction hypothesis $Atoms(\sigma(\mathcal{S}_{\leq i})) \subseteq Atoms(\mathcal{P})$, and by Observation* 2, it is true that $Atoms(\sigma(R_i)) \subseteq Atoms(\mathcal{P})$. For $\sigma(S_i)$, it is obvious that $Atoms(\sigma(S_i)) = Atoms(S_i) \cup Atoms(\sigma(Var(S_i)))$. Let $(A, h) = \pi^{-1}(i)$. By the specification of the protocol P, each variable $y \in S_{(A,h)}$ must appear in some received message $R_{(A,f)}$ for some $f \leq h$. Be the definition of a protocol execution, it must be true that $\pi(A, f) \leq \pi((A, h)) = i$; therefore, by the induction hypothesis, it must be true that $Atoms(\sigma(Var(S_i))) \subseteq Atoms(\sigma(\mathcal{P}))$. Since $Atoms(S_i) \subseteq Atoms(\mathcal{P})$, it is true that $Atoms(\sigma(S_i)) \subseteq Atoms(\mathcal{P})$. ■

Substitution and the tree of a term and are defined in Section IV. We consider two kinds of substitutions. One kind replace variables with ground terms, such as $\sigma = [x \leftarrow t]$ for a variable x and a ground term t. The substitution σ in a protocol execution belongs to this kind. The other kind replaces ground a general term t with another general term t', such as $\delta = [t_1 \leftarrow t_2]$ for two ground terms t_1 and t_2. This kind of more general substitutions will be used in the reasoning of the proofs. We have the following observation.

Observation* 4: Consider a term T that is not ground, a term t that is ground, and a substitution σ, and a substitution δ, and a substitution σ', such that the following conditions are satisfied:

1) σ is a substitution replacing variables with ground terms.
2) δ is a substitution from ground terms to ground terms, and $\delta = [t \leftarrow t']$ for some ground terms t and t', i.e. δ only replaces t.
3) There is no term g such that $g \in Sub(T)$ and g is not a variable and $\sigma(g) = t$.
4) σ' is a substitution replacing variables with ground terms, and is defined as follows. Let $V_t = \{x | x \in Var(T)$ and $t \in Sub(\sigma(x))\}$. For a variable y, if $t \notin Sub(\sigma(y))$, which means $y \notin V_t$, then $\sigma'(y) = \sigma(y)$, otherwise if $t \in Sub(\sigma(y))$, which means $y \in V_t$, then $\sigma'(y) = \delta(\sigma(y))$.

It is true that $\sigma'(T) = \delta(\sigma(T))$.

Proof: If the observation is very intuitive, a reader can skip the following proof. We try to prove that the tree of $\delta(\sigma(T))$ is the same as $\sigma'(T)$ by analyzing the nodes with different locations.

For each variable in T, say variable x, the tree of $\sigma(T)$ (respectively the tree of $\sigma'(T)$) replaces every leave node of the tree of T that is marked with x with a tree of $\sigma(x)$ (respectively by a tree of $\sigma'(x)$). The differences between the tree of $\sigma(T)$ and the tree of $\sigma'(T)$ are the replacement of the nodes of variables in V_t in the tree of T.

Suppose $t \in Sub(\sigma(T))$, we can discuss the possible location L of the node labeled with t in the tree of $\sigma(T)$. Note that L may or may not be the location of a node in the tree of T. The possible cases are the follows:

- L is the location of some node nd in the tree of T, and nd is labeled by a term g that is not a variable. Then $\sigma(g) = t$, which is impossible by the condition 3 assumed by the observation.
- L is the location of some node nd in the tree of T, and nd is labeled by a variable x. Then $\sigma(x) = t$, and it must be true that $x \in V_t$.
- L note the location of some node in the tree of T, then there must be some node labeled with a variable x with location L', such that L' is a prefix of L. And it must be true that $t \in Sub(\sigma(x))$. Then it must be true that $x \in V_t$.

The locations of the nodes in the tree of T can be categorized in three groups: L_1 is the locations of the nodes that are not labeled by variables. L_2 is the locations of the nodes in T that are labeled by variables in V_t. L_3 is the locations of the nodes in T that are labeled by variables not in V_t.

The locations of the nodes in the tree of $\sigma(T)$ can be categorized in three groups. First, $L'_1 = L_1$. Second L'_2 (and similarly L'_3) is the locations of the nodes of a subtree whose top node has a location in L_2 (similarly in L_3). Similarly the locations of the nodes in tree of $\sigma'(T)$ can be categorized in three groups L''_1, L''_2 and L''_3. and $L'_1 = L''_1 = L_1$, and $L''_2 = L'_2$.

It is true that in the tree of $\sigma(T)$ and the tree of $\sigma'(T)$, the nodes with locations in $L'_1 \cup L'_2$ are the same. The difference between the tree of $\sigma(T)$ and $\sigma'(T)$ are the nodes with locations in L''_3 and L'_3. In other words, the difference between the tree of $\sigma(T)$ and $\sigma'(T)$ is the subtrees that replace the nodes in the tree of T labeled with X.

By the above reasoning of the locations of the node labeled with t in the tree of $\sigma(T)$, we know that applying δ to $\sigma(T)$ will only change some nodes in the subtree that replaces the node in the tree of T labeled with x.

It is obvious that $\delta(\sigma(x)) = \sigma'(x)$ for a variable x; therefore, the node labeled with x in the tree of T have the same replacement in the tree of $\sigma(T)$ and in the tree of $\sigma'(T)$, which means $L''_3 = L'_3$, and for a location $L \in L''_3$, the node at L in the tree of $\sigma(T)$ has the same mark as the node at L in the tree of $\sigma'(T)$.

So, the two trees of $\delta(\sigma(T))$ and $\sigma'(T)$ are identical and $\delta(\sigma(T)) = \sigma'(T)$.

■

Lemma** 1: Given a set of terms E, and a term t, such that $t \in forge(E)$, and given a substitution $\delta = [w \leftarrow w']$, where w and w' are two terms (could be ground) and w is not an asymmetric key, and $w \in forge(E)$, it is true that $\delta(t) \in forge(\delta(E) \cup \{w'\})$.

Proof: We clarify the meaning of the statement first. The reason for requiring that w is not an asymmetric key is to avoid the situation of replacing an asymmetric key, which is an atomic term, with any term that is not an asymmetric key, which is not allowed for asymmetric encryptions.

The reason for saying $\delta(t) \in forge(\delta(E) \cup \{w'\})$, instead of saying $\delta(t) \in forge(\delta(E))$ is to avoid an error. It is possible that $w' \notin forge(\delta(E))$. $\delta(E)$ means to apply δ to terms in E, not the terms that are derived from E. For example, consider $E = \{a, b\}$, and $\delta = [\langle a, b\rangle \leftarrow c]$, it is obvious that $\langle a, b\rangle \in forge(E)$ but $\delta(\langle a, b\rangle) = c \notin forge(\delta(E)) = forge(E)$. For the reasoning of the proof, details are showed below, considering $forge(\delta(E \cup \{w'\}))$ is needed to handle the case that $L_c(w)$ appears in a derivation D from E to goal t.

The reason for saying $\delta(t) \in forge(\delta(E) \cup \{w'\})$ instead of $\delta(t) \in forge(\delta(E \cup \{w'\}))$ is to avoid an error for the cases where $w \in Sub(w')$. For example, consider $E = \{a, b\}$, and $\delta = [\langle a, b\rangle \leftarrow \{\langle a, b\rangle\}^{\leftrightarrow}_c]$, it is obvious that $\langle a, b\rangle \in forge(E)$ but

$$\delta(\langle a, b\rangle) = \{\langle a, b\rangle\}^{\leftrightarrow}_c \notin forge(\delta(E, \{\langle a, b\rangle\}^{\leftrightarrow}_c)) = forge(E, \{\{\langle a, b\rangle\}^{\leftrightarrow}_c\}^{\leftrightarrow}_c).$$

The reason for requiring that $w \in forge(E)$ is to show that there is a derivation from E with the goal w with the property that $L_d(w)$ does not appear in this derivation. Obviously the normal derivation from E to the goal w satisfies this property. Then by proposition 2, there is a derivation D from E to the goal t such that $L_d(w)$ does not appear in D. Let D have the following form

$$D = E_0 \rightarrow_{L_1} E_1 \rightarrow_{L_2} E_2 \cdots \rightarrow_{L_h} E_h,$$

where $E_0 = E$, $t \in E_h$, $h \geq 0$, L_i is an application of a derivation rule, and L_i is identified as $L_{c/d}(T_i)$, for $i \geq 1$, and some term T_i, $T_i \in E_{i-1}$. $L_{c/d}(T_i)$ corresponds to $LHS_i \rightarrow RHS_i$, where LHS_i and RHS_i are two sets of terms.

Now we want to show that there is a derivation D' starting from $\delta(E)$ with the goal $\delta(t)$. D' can be constructed as follows. D' has the same length as D, and has the form

$$D' = E'_0 \rightarrow_{L'_1} E'_1 \rightarrow_{L'_2} E'_2 \cdots \rightarrow_{L'_h} E'_h,$$

where $E'_0 = \delta(E) \cup \{w'\}$, and the i^{th} step of D', for $1 \leq i \leq h$, is defined as follows.

- If the i^{th} step of D, which is L_i, has the form $L_c(w)$, then the h^{th} step of D' is empty rule, i.e. $L'_i = L_\emptyset$, which means doing nothing and $E'_i = E'_{i-1}$.
- Otherwise, L_i is not $L_c(w)$, say L_i is identified as $L_{c/d}(T_i)$ for some term T_i, then $L'_i = \delta(L_i) = L_{c/d}(\delta(T_i))$. L'_i corresponds to $LHS'_i \rightarrow RHS'_i$ for two sets LHS'_i and RHS'_i.

In order to show that D' is a valid derivation, we have to show two facts that are satisfied at each step, say the i^{th} step, of D':

1) $LHS'_i \subseteq E'_{i-1}$. In order to show this, we prove a stronger result by induction: for each i, $0 \leq i \leq h$, $E'_i = \delta(E_i) \cup \{w'\}$.
2) The derivation rule is applicable. There are several cases that a derivation rule is not applicable. (1) The decomposition rule of an encryption (pair) cannot be applied to a pair (encryption). (2) The composition rule

of an asymmetric encryption is not applicable using a term that is not an asymmetric key as an asymmetric key.

We prove that D' is a valid derivation by induction on the derivation steps.

Normally the induction starts with $i = 1$ since there is no derivation rule applied for $i = 0$. However the reasoning for $i = 1$ is not simpler than the reasoning for the induction steps when $i > 1$. In order to move all the similar reasoning to the induction steps, we can equivalently consider D has a prefix $E_{-1} \rightarrow_{L_0} E_0$, where $E_{-1} = E_0$ and $L_0 = L_\emptyset$. Similarly D' also has a dummy prefix $E'_{-1} \rightarrow_{L'_0} E'_0$. Here $L_\emptyset$ corresponds to $\emptyset \rightarrow \emptyset$.

Base case: For $i = 0$. Obviously $E'_0 = \delta(E_0) \cup \{w'\}$ and the rule $L_\emptyset$ is applicable. So the two facts are satisfied at step 0.

Induction step: for $i \geq 1$, suppose for all j, $j < i$, the two facts are satisfied, which implies that $\delta(E_{i-1}) \cup \{w'\} = E'_{i-1}$. We want to prove that the two facts are also satisfied at step i. We check each possible derivation rule listed below.

- In D, L_i is identified as $L_c(T_i)$, and the following cases are possible.
  - $T_i = w$, then in D', L'_i is a dummy rule. $\delta(E_i) \cup \{w'\} = \delta(E_{i-1} \cup \{w\}) \cup \{w'\} = \delta(E_{i-1}) \cup \{w'\} = E'_{i-1} = E'_i$; therefore, fact 1 is true for step i. Fact 2 is trivially true for the dummy rule.
  - $T_i \neq w$. There are three cases.
    - $T_i = \langle \alpha, \beta \rangle$. Then $\alpha \in E_{i-1}$ and $\beta \in E_{i-1}$; therefore, $\{\delta(\alpha), \delta(\beta)\} \subseteq \delta(E_{i-1}) \subseteq E'_{i-1}$. L'_i is obviously applicable and fact 2 is obviously satisfied for i. $\delta(E_i) \cup \{w'\} = \delta(E_{i-1} \cup \{\langle \alpha, \beta \rangle\}) \cup \{w'\} = \delta(E_{i-1}) \cup \{\delta(\langle \alpha, \beta \rangle)\} \cup \{w'\} = E'_{i-1} \cup \{\delta(\langle \alpha, \beta \rangle)\} = E'_i$; therefore, fact 1 is true for i.
    - $T_i = \{\alpha\}^p_\beta$. Fact 1 is obviously true by the same reasoning for the case of $T_i = \langle \alpha, \beta \rangle$. For fact 2, we have to show that $\delta(\beta)$ is an asymmetric key. Since w is not an asymmetric key, and β is a symmetric key, $\beta \neq w$. Because β is atomic, it is true that $\delta(\beta) = \beta$, and $\delta(\beta)$ is an asymmetric key. So fact 2 is satisfied.
    - $T_i = \{\alpha\}^s_\beta$. Fact 1 and fact 2 are satisfied by the same reasoning for case of $T_i = \langle \alpha, \beta \rangle$.
- In D, L_i is identified as $L_d(T_i)$. Note that $T_i \neq w$ by the way D is chosen. There are three cases.
  - $T_i = \langle \alpha, \beta \rangle$. $T_i \in E_{i-1}$; therefore, $\delta(T_i) \in \delta(E_{i-1}) \subseteq E'_{i-1}$. It is true that $\delta(T_i) = \langle \delta(\alpha), \delta(\beta) \rangle$. Note that the only possibility that $\delta(\langle \delta(\alpha), \delta(\beta) \rangle) \neq \langle \delta(\alpha), \delta(\beta) \rangle$ is when $w = \langle \delta(\alpha), \delta(\beta) \rangle$, which can be justified easily by analyzing the tree of a term, but it is impossible by the way D is chosen. So L'_i is applicable and fact 2 is true. $\delta(E_i) \cup \{w'\} = \delta(E_{i-1}) \cup \delta(\{\alpha, \beta\}) \cup \{w'\} = E'_{i-1} \cup \delta(\{\alpha, \beta\}) = E'_i$. So fact 1 is proved.
  - $T_i = \{\alpha\}^p_\beta$. Then $T_i \in E_{i-1}$ and $\beta^- \in E_{i-1}$. Since β^- is an asymmetric key, it is true that β^- is atomic, $\beta^- \neq w$, and, $\delta(\beta^-) = \beta^-$, and $\delta(\beta^-)$ is also the inverse key for $\delta(\beta) = \beta$. Then following the same reasoning for the above case of $T_i = \langle \alpha, \beta \rangle$, fact 1 and fact 2 are true.
  - $T_i = \{\alpha\}^s_\beta$. Then following the same reasoning for the above case of $T_i = \langle \alpha, \beta \rangle$, fact 1 and fact 2 are true.

So we have proved that D' is a valid derivation from $\delta(E) \cup \{w'\}$. Since $t \in E_h$, $\delta(t) \in \delta(E_h) \subseteq E'_h$; therefore, $\delta(t) \in forge(E'_0) = forge(\delta(E) \cup \{w'\})$. Lemma* 1 is proved. ■

Corollary* 1: Given a set of terms E, and a term t, such that $t \in forge(E)$, and given a substitution $\delta = [K_g \leftarrow K_f;\ K_g^- \leftarrow K_f^-]$, where K_g and K_g^-, and K_f and K_f^-, are two pairs of asymmetric keys, it is true that $\delta(t) \in forge(\delta(E) \cup \{K_f, K_f^-\})$.

Proof: Note that the two pairs of asymmetric keys, K_g and K_g^-, and K_f and K_f^-, can represent public keys and private keys established before a run, where f and g are some agent names, or they can represent asymmetric keys dynamically generated in a run, where f and g are the unique identities of the two pairs of the keys. Since K_g and K_g^- are atomic terms, it is impossible to have a derivation D from E that includes a rule application represented as $L_d(K_g)$ or $L_d(K_g^-)$; therefore, we do not require that $\{K_g, K_g^-\} \subset forge(E)$, while in Lemma* 1 it is required that $w \in forge(E)$. The proof is very similar to Lemma* 1, and is even simpler. Given a derivation D from $E_0 = E$ to the goal t, which has the form $E_0 \rightarrow_{L_1} E_1 \rightarrow_{L_2} \cdots E_h$, we construct a derivation D' from $E'_0 = \delta(E) \cup \{K_f, K_f^-\}$, . D' and D have the same length h, $h \geq 0$. For the i^{th} applied in D labeled with $L_{c/d}(T_i)$, $1 \leq i \leq h$ when $h > 1$, the i^{th} rule applied in D' is labeled with $L_{c/d}(\delta(T_i))$. In order to show that D' is a valid derivation, we can prove by induction that $E'_i = \delta(E_i) \cup \{K_f, K_f^-\}$, and $L_{c/d}(\delta(T_i))$ is an applicable rule. Then since $t \in E_h$, it is true that $\delta(t) \in \delta(E_h) \cup \{K_f, K_f^-\} = E'_h$. ■

The Improved Proof of Lemma 4:

Proof: Now we prove Lemma 4 by contradiction. *Contradiction Assumption*: suppose to the contradiction of

Lemma 4, in a normal attack with substitution σ, there is a variable x such that for every term t that $t \sqsubseteq_\sigma \sigma(x)$, $t \notin Sub(\mathcal{P})$.

By Observation* 3, $Atoms(\sigma(x)) \subseteq Atoms(\mathcal{P})$. If $|\sigma(x)|_{DAG} = 1$, then $\sigma(x)$ is a constant and $\sigma(x) \in Atoms(\mathcal{P})$, which violates the above contradiction assumption of Lemma 4; therefore, $|\sigma(x)|_{DAG} > 1$, which means $\sigma(x)$ is a ground composed term.

Since all terms in S_0 are constants (atoms), it must be true that $\sigma(x) \notin S_0$, otherwise there is a σ match of $\sigma(x)$ in S_0 which is impossible by the contradiction assumption. Since $x \in Var(\mathcal{P})$, it is true that $\sigma(x) \in Sub(\sigma(\mathcal{P}))$, i.e., $\sigma(x) \in Sub(\sigma(S_j))$ or $\sigma(x) \in Sub(\sigma(R_j))$ for some $1 \leq j$. Furthermore, in order to satisfy the contradiction assumption, every occurrence of $\sigma(x)$ must be a subterm of $\sigma(y)$ for some variable y appearing in S_j or R_j, for $1 \leq j$, which can be justified by analyzing the possible location of the tree of $\sigma(x)$ in the tree of $\sigma(S_j)$ or $\sigma(R_j)$, as discussed in the proof of Observation* 4.

Suppose $\sigma(x)$ appears in a sent message S_j where $j > 0$, then there is a variable y appearing in S_j such that $\sigma(x)$ is a subterm of $\sigma(y)$, as explained in the above paragraph. By the way a protocol is specified, and by the definition of a session, if a variable y appears in S_j, then y must appear in some received message R_i, for some $i \leq j$. This feature on the appearance of variables has been discussed in 3; therefore, there is a smallest integer nx, $nx > 0$, such that $\sigma(x) \in Sub(\sigma(R_{nx}))$ and $\sigma(x) \notin Sub(S_0 \cup \{\sigma(S_1)\} \cdots \cup \{\sigma(S_{nx-1}\}$. Since $\sigma(R_{nx}) \in forge(S_0, \sigma(S_1), \cdots, \sigma(S_{nx-1}))$, by Observation* 1, it is true that $\sigma(x) \in forge(S_0, \sigma(S_1), \cdots, \sigma(S_{nx-1}))$.

Now we define a new substitution σ'. We choose any atom t in S_0. The idea is that $|t|_{DAG} = 1 < |\sigma(x)|_{DAG}$, since $\sigma(x)$ is a composed term. Note that $S_0 \neq \emptyset$ since the attacker always knowns its name I. In [1] the attacker's name is $Charlie$, we use I for simplicity. So we can always choose I as the t. Let $V_{\sigma(x)}$ be defined as $\{y | \sigma(x) \in Sub(\sigma(y))$, and $y \in Var(\mathcal{P})\}$. Let δ be a substitution $[\sigma(x) \leftarrow I]$. Note that δ is a substitution replacing a ground term $\sigma(x)$, not replacing a variable. $\delta(t)$ means to replace any occurrence of $\sigma(x)$ in t with I, i.e. in the tree of t, every subtree of $\sigma(x)$ (the tree whose top node is labeled with x) is replaced with a tree of I. σ' is defined as follows.

- For any variable u such that $u \in Var(\mathcal{P}) - V_{\sigma(x)}$, $\sigma'(u) = \sigma(u)$.
- For any variable y such that $y \in V_{\sigma(x)}$, $\sigma'(y) = \delta(\sigma(y))$. We can observe that $\sigma'(x) = I$.

Note that in the proof of Lemma 4 in [1] σ' is defined in a way that the only difference between σ' and σ is the substitution for x, which is incorrect since then the proof of [1] cannot go through by such a definition of σ'.

Now we show that σ' is associated with a correct execution. We want to show that for each R_i, $i \geq 1$, $\sigma'(R_i) \in forge(S_0, \sigma'(S_1), \cdots, \sigma'(S_{i-1}))$. Since σ is associated with an attack, which is a correct execution, it is true that

$$\sigma(R_i) \in forge(S_0, \sigma(S_1), \cdots, \sigma(S_{i-1})).$$

For each i such that $1 \leq i < nx$, it is true that $\sigma(x) \notin Sub(\mathcal{P}_{<i})$; therefore,

$$\delta(\sigma(R_i)) = \sigma(R_i) \in forge(S_0, \sigma(S_1), \cdots, \sigma(S_{i-1})) = forge(\delta(S_0, \sigma(S_1), \cdots, \sigma(S_{i-1}))).$$

For each i such that $nx \leq i \leq k$, we can apply Lemma* 1. Since $\sigma(x) \in forge(S_0, \sigma(S_1), \cdots, \sigma(S_{i-1}))$, and $\sigma(x)$ is not an asymmetric key, by Lemma* 1, it is true that

$$\delta(\sigma(R_i)) \in forge(\delta(S_0, \sigma(S_1), \cdots, \sigma(S_{i-1})), I) = forge(\delta(S_0, \sigma(S_1), \cdots, \sigma(S_{i-1})))).$$

By Observation* 4, it is true that

$$\sigma'(R_i) = \delta(\sigma(R_i)),$$

and

$$\sigma'(S_0, S_1, \cdots, S_{i-1}) = \delta(\sigma(S_0, S_1, \cdots, S_{i-1}).$$

For $1 \leq i \leq k$ it is true that

$$\sigma'(R_i) = \delta(\sigma(R_i)) \in forge(\delta(S_0, \sigma(S_1), \cdots, \sigma(S_{i-1}))) = forge(S_0, \sigma'(S_1), \cdots, \sigma'(S_{i-1})$$

and σ' is associated with a correct execution.

Now we show that σ' is an attack, i.e., the attacker can know a secret term after the execution of $\mathcal{P}$ with σ'. This aspect is not discussed in the proof of Lemma 4 in [1], but is needed. We consider the secret term is declared similarly to the secret term discussed by [12] for the public key Needham-Schroeder protocol, which is a nonce generated by a regular agent. In [1] all nonces are represented as unique atoms in a session $\mathcal{P}$; therefore, here we consider the secret term, call it $Secret$, is a constant and $Secret \in Atoms(P)$. Let $\mathcal{S}_{\leq k} = S_0, S_1, \cdots, S_k$ where S_k is the last message sent in the session. Since $Secret \in forge(\sigma(\mathcal{S}_{\leq k}))$, by Lemma* 1, we have

$$\delta(Secret) \in forge(\delta(\sigma(\mathcal{S}_{\leq k})) \cup \{I\}) = forge(\delta(\sigma(\mathcal{S}_{\leq k}))).$$

By Observation* 4, it is true that

$$\delta(\sigma(\mathcal{S}_{\leq k})) = \sigma'(\mathcal{S}_{\leq k});$$

therefore, $Secret = \delta(Secret) \in forge(\delta(\mathcal{S}_{\leq k})) = forge(\sigma'(\mathcal{S}_{\leq k}))$. So $Secret$ is also leaked in the execution $\mathcal{P}$ with σ'. Since the size of σ is strictly larger than $sigma'$, it σ cannot be associated with a normal attack, contradiction. So Lemma 4 is true.

■

VII. Proof of No Nonce and Key Generation by the Attacker

In proof of [1] assumes no nonce generation of the attacker, which is mentioned at the end of the summary of [1]. Since potentially the attacker can generate a huge number of nonces, and all of these nonces are included in S_0, in order to prove the NP result, proving the size of S_0 is polynomial of the size of the protocol is necessary. In [1] the statement is that if the attacker can generate multiple nonces, by replacing all these nonces by the attacker's name it is also an attack. Although the statement is intuitive, it is better to have a proof.

Furthermore, how about the asymmetric keys that the attacker can dynamically generate in a run? Nonces cannot be used as asymmetric keys since they belong to different type, while a nonce may replace any term that is used as a symmetric key. In [1] this issue is not handled.

Theorem** 1: Consider a protocol P, given a description of an execution of a session $\mathcal{P} = S_0 \cup \{R_1, S_1, R_2, S_2, \cdots R_k, S_k\}$ with a substitution σ , where initial knowledge of the attacker S_0 includes a nonce n_I that is an atom created by the attacker, and this execution is an attack to P. Define a substitution $\delta = [n_I \leftarrow I]$, where I is the attacker's name and $I \in S_0$. Define $S_0' = \delta(S_0) = S_0 - n_I$. Define σ' as follows. Define $V_{n_I} = \{y | y \in Var(\mathcal{P})$ and $n_I \in Sub(\sigma(y))\}$. For a variable $y \in Var(\mathcal{P})$, if $y \notin V_{n_I}$, then $\sigma'(y) = \sigma(y)$, otherwise $\sigma'(y) = \delta(\sigma(y))$. Let $\mathcal{P}'$ be the same as $\mathcal{P}$ except that S_0 is replaced by S_0'. It is true that $\mathcal{P}'$ with σ' is also an attack to P.

Proof: Note that $\sigma'(S_0) = \delta(S_0) = S_0 - n_I = S_0'$, let $\mathcal{S}_{\leq h} = S_0 \cup \{S_1\} \cdots \cup \{S_h\}$, for $0 \leq h \leq k$. And similarly $\mathcal{S}'_{\leq h} = S_0' \cup \{S_1\} \cdots \cup \{S_h\}$. Let $Secret$ be the secret term, which is a constant, and $Secret \in Atoms(R_1, S_1, R_2, S_2, \cdots R_k, S_k)$.

We have to prove two facts. First σ' and $\mathcal{P}'$ form a correct execution, which means we have to prove that $\sigma'(R_i) \in forge(\sigma'(\mathcal{S}'_{\leq i-1}))$, for $1 \leq i \leq k$. Since $\mathcal{P}$ and σ form a correct execution, it is true that $\sigma(R_i) \in forge(\sigma(\mathcal{S}_{\leq i-1}))$. Since $n_I \in S_0 \subset forge(\mathcal{E}_{i-1})$, and n_I is not an asymmetric key, by Lemma* 1 it is true that

$$\delta(\sigma(R_i)) \in forge(\delta(\sigma(\mathcal{S}_{\leq i-1})), I) = forge(\delta(\sigma(\mathcal{S}_{\leq i-1}))).$$

Call this result *fact 1*.

Now we want to apply Observation* 4 to show that $\sigma'(R_i) = \delta(\sigma(R_i))$, and $\sigma'(S_i) = \delta(\sigma(S_i))$, call this result *fact 2*. Particularly, we have to show that the condition 3 of Observation* 4 is satisfied, which requires that it is impossible to have a term g such that $g \in Sub(R_i)$ or $g \in Sub(S_i)$, for $1 \leq i \leq k$, and g is not a variable, and $\sigma(g) = n_i$. Since n_I is a nonce generated by the attacker, it is true that $n_i \notin Atoms(R_1, S_1, \cdots R_k, S_k)$. For any term t such that $t \in Sub(R_1, S_1, \cdots R_k, S_k)$, if t is a composed term then $\sigma(t)$ is also a composed term and $\sigma(t) \neq n_I$; therefore, the condition 3 is satisfied. Other conditions of Observation* 4 are obviously satisfied; therefore, *fact 2* is proved.

Based on fact 1 and fact 2, it is true that

$$\sigma'(R_i) = \delta(\sigma(R_i)) \in forge(\delta(\sigma(\mathcal{S}_{\leq i-1}))) = forge(\sigma'(\mathcal{S}_{\leq i-1})).$$

So $\mathcal{P}'$ and σ' form a correct execution.

Second, we have to prove that $Secret \in forge(\sigma'(\mathcal{S}'_{\leq k}))$. Note that $Secret \neq n_I$. We know that $Secret \in forge(\sigma(\mathcal{S}_{\leq k}))$. n_I is not an asymmetric key; therefore, we can apply Lemma* 1, and by *fact 2*, and it is true that

$$Secret = \delta(Secret) \in forge(\delta(\sigma(\mathcal{S}_{\leq k})), I) = forge(\delta(\mathcal{S}_{\leq k})) = forge(\sigma'(\mathcal{S}_{\leq k})).$$

■

The following corollary is very similar to Theorem* 1, but it indicates that for any pair of asymmetric keys, K_g and K_g^-, dynamically generated by the attacker in an attack, if they are replaced by K_I and K_I^-, it is still an attack.

Corollary** 2: Consider a protocol P, given a description of an execution of a session $\mathcal{P} = S_0 \cup \{R_1, S_1, R_2, S_2, \cdots R_k, S_k\}$ with a substitution σ , where initial knowledge of the attacker S_0 includes a nonce n_I that is an atom created by the attacker, and this execution is an attack to P. Define a substitution $\delta = [K_g \leftarrow K_I;\ K_g^- \leftarrow K_I^-]$, where K_g and K_g^- are a pair of asymmetric keys dynamically generated by the attacker in a run, and K_I and K_I^- are the attacker's public key and private key respectively, and $\{K_I, K_I^-\} \subset S_0$. Define $S_0' = \delta(S_0) = S_0 - \{K_g, K_g^-\}$. Define σ' as follows. Define $V_{n_I} = \{\ y | y \in Var(\mathcal{P})$ and $K_g \in Sub(\sigma(y))$ or $K_g^- \in Sub(\sigma(y))\ \}$. For a variable

$y \in Var(\mathcal{P})$, if $y \notin V_{n_I}$, then $\sigma'(y) = \sigma(y)$, otherwise $\sigma'(y) = \delta(\sigma(y))$. Let $\mathcal{P}'$ be the same as $\mathcal{P}$ except that S_0 is replaced by S_0'. It is true that $\mathcal{P}'$ with σ' is also an attack to P.

Proof: The proof is very similar to Theorem* 1. let $\mathcal{S}_{\leq h} = S_0 \cup \{S_1\} \cdots \cup \{S_h\}$, for $0 \leq h \leq k$. Since $\mathcal{P}$ and σ describe a correct execution, then for $1 \leq h \leq k$, $\sigma(S_i) \in forge(\sigma(\mathcal{S}_{\leq h}))$. By Corollary* 1 it is true that

$$\delta(\sigma(R_h)) \in forge(\delta(\sigma(\mathcal{S}_{\leq h}) \cup \{K_I, K_I^-\}) = forge(\delta(\sigma(\mathcal{S}_{\leq h}))).$$

Since K_g and K_g^- are two atomic terms that are generated by the attacker, it is obvious that there does not exist a composed term T such that $T \in Sub(R_1, S_1, \cdots R_k, S_k)$ and $\sigma(T) = K_g$ or $\sigma(T) = K_g^-$; therefore, by applying Observation* 4, we know that for $1 \leq i \leq k$, $\sigma'(R_i) = \delta(\sigma(R_i))$, and $\sigma'(S_i) = \delta(\sigma(S_i))$.

Based on the above reasoning, we know that $\mathcal{P}'$ and σ' form a correct execution. Note that $Secret \notin \{K_g, K_g^-\}$. Since $Secret \in forge(\sigma(\mathcal{S}_{\leq k}))$, then by Corollary* 2, it is true that

$$Secret = \delta(Secret) \in forge(\delta(\sigma(\mathcal{E}_{\leq k})), I) = forge(\delta(\mathcal{S}_{\leq k})) = forge(\sigma'(\mathcal{S}_{\leq k})).$$

■

***Corollary** 3:* Given a protocol P and a secret term $Secret$ such that $Secret \in Atoms(P)$, if there is an attack of P formed by $\mathcal{P} = S_0, R_1, S_1, \cdots R_k, S_k$ and σ such that $Secret$ is leaked, then there is an attack formed by $\mathcal{P}' = S_0', R_1, S_1, \cdots R_k, S_k$ and σ', for some substitution σ', such that in S_0' no nonce or asymmetric key that is generated by the attacker is included.

Proof: We can apply Theorem* 1 repeatedly, to update S_0 and σ of an attack, and each time a nonce generated by the attacker, which is included in S_0, is replaced with the attacker's name I, and finally when all nonces generated by the attacker are replaced with I, and the obtained S_0' and σ' still form an attack. Then we can apply Corollary* 2 repeatedly to replace every pair of asymmetric keys generated by the attack with K_I and K_I^-, and the obtained S_0' and σ' still form an attack ■

VIII. The Error

Theorem 1 If σ is the substitution in a normal attack then we have for all $x \in Var$ $|\sigma(x)|_{DAG} \leqslant |\mathcal{P}|_{DAG}$.

As a reminder, $\mathcal{P}$ is defined as $\{R_j | j = 1, \ldots, i\} \cup \{S_j | j = 0, \ldots, i\}$. More precisely, $\mathcal{P}$ is the rules of the protocol specification arranged in the order of execution, together with the attacker's initial knowledge. Remind that the notation of $\mathcal{SP}$ means $Sub(\mathcal{P}$, which is the set of subterms in $\mathcal{P}$.

The following is the proof of [1] in page 464 and 465.

Proof. Given a set of variable U, we shall write $\overline{U} = \{\sigma(x) | x \in U\}$.

Let us build by induction a sequence of sets $E_p \subseteq \mathcal{SP}$ and a sequence of sets V_p of variables such that $|\sigma(x)|_{DAG} \leqslant |E_p, \overline{V_p}|_{DAG}$, where p is a number starting from 0.

- Let (E_0, V_0) be $(\emptyset, \{x\})$. We have $|\sigma(x)|_{DAG} \leqslant |E_0, \overline{V_0}|_{DAG}$ and $E_0 \subseteq \mathcal{SP}$.
- Assume that we have built (E_p, V_p) such that $|\sigma(x)|_{DAG} \leqslant |E_p, \overline{V_p}|_{DAG}$ and $E_p \subseteq \mathcal{SP}$, let us define E_{p+1} and V_{p+1}: If $V_p \neq \emptyset$ let us choose $x' \in V_p$, for any x'. Then there exists $t \sqsubseteq_\sigma \sigma(x')$ such that $t \in \mathcal{SP}$, according to Lemma 4. Note that it is guaranteed that t is not a variable and $x \notin Var(t)$. We define $E_{p+1} = E_p \cup \{t\}$ and $V_{p+1} = Var(t) \cup V_p - \{x'\}$.

Since $t \in \mathcal{SP}$, $E_{p+1} \subseteq \mathcal{SP}$. Now the proof wants to show the following result.

$$|\sigma(x)|_{DAG} \leqslant |E_{p+1}, \overline{V_{p+1}}|_{DAG} \tag{1}$$

The reason to prove (1) is that this construction will terminate since there is only finite many variables. At the end, say at the e step, $V_e = \emptyset$, and then the last set of terms E_e will have the property that $|\sigma(x)|_{DAG} \leqslant |E_e|_{DAG}$. Since $E_e \subseteq \mathcal{SP}$, $|E_e|_{DAG} \leqslant |\mathcal{P}|_{DAG}$, and then theorem 1 is proved.

Now let's see how (1) is proved in [1]. Let $\delta = \{[y \leftarrow \sigma(y)]/y \in Var(t)\}$. We show the following result.

$$|E_p\delta, \overline{V_p}|_{DAG} \leqslant |E_p, \overline{V_p - x'}, t, \overline{Var(t)}|_{DAG} = |E_{p+1}, \overline{V_{p+1}}|_{DAG} \tag{2}$$

Here is how (2) is proved in [1]. By Corollary 1 on $E_p \cup \{t\} \cup \overline{V_p - x'}$ for the substitution δ. Remark: $t\delta = \sigma(x')$. We obtain the following result (elaborated by the authors of this article).

$$|E_p\delta|_{DAG} \leqslant |E_p, Var(t)\delta|_{DAG} \tag{3}$$

By Corollary 1 we also have $|t\sigma|_{DAG} \leqslant |t, \overline{Var(t)}|_{DAG}$. So we have the following result.

$$|\overline{V_p}|_{DAG} = |\overline{V_p - x'}, \overline{x'}|_{DAG} = |\overline{V_p - x'}, t\sigma|_{DAG} \leqslant |\overline{V_p - x'}, t, \overline{Var(t)}|_{DAG} \tag{4}$$

Then by adding (3) and (4) together we get exactly (2). So (2) is proved.

However having (2) is not enough to show (1). What is said in [1] is that after (2), we have the following result, (without any explanation):

$$|E_p, \overline{V_p}|_{DAG} \leqslant |E_p\delta, \overline{V_p}|_{DAG} \leqslant |E_{p+1}, \overline{V_{p+1}}|_{DAG} \tag{5}$$

Note that the second $\leqslant$ in (5) is (2). So in order to prove (1), after (2) is proved, what we need is simply the first $\leqslant$ of (5):

$$|E_p, \overline{V_p}|_{DAG} \leqslant |E_p\delta, \overline{V_p}|_{DAG} \tag{6}$$

Error: (6) cannot be proved.

But how (6) is obtained is not explained by [1]. There is no lemma or proposition or corollary can support (6). Then if (6) cannot be proved, (5) is not proved, and (1) is not proved, and Theorem 1 is not proved.

Actually (6) appears to be correct, if Corollary 1 is understood in the opposite way. Here is the ***illusion***:

Suppose Corollary 1 is understood in the reverse direction, i.e. if we assume the following result:

$$|E_p\delta|_{DAG} \geqslant |E_p, var(t)\delta|_{DAG} \tag{7}$$

Then obviously $Sub(Var(t)\delta) = Sub(Var(t)\sigma) \subseteq Sub(t\sigma) = Sub(x\sigma) \subseteq Sub(\overline{V_p})$, where $x \in V_p$ and t is chosen so that $\sigma(t) = \sigma(x)$. Note that the two notations of $x\sigma$ and $\sigma(x)$ are the same.

Then we have

$$|E_p, \overline{V_p}|_{DAG} \leqslant |E_p, Var(t)\delta, \overline{V_p}|_{DAG} \leqslant |E_p\delta, \overline{V_p}|_{DAG}.$$

And then (6) is proved. But it is wrong since it is based on the illusion of (7) which is the contradictory usage of Corollary 1, except that rare case that $=$ is covered by both $\leqslant$ and $\geqslant$.

Now we have seen that the proof of (5) is wrong. But on the other hand, is (5) actually wrong and cannot be proved at all? Here a counter example of (5) to show that $|E_p, \overline{V_p}|_{DAG} \leqslant |E_{p+1}, \overline{V_{p+1}}|_{DAG}$ is not true, i.e. the updating process for the proof is wrong.

Counter Example of (5):

The idea of designing the counter example is to imagine a case during the process to update E_p and V_p, when term t is introduced to E_p while a variable y is removed from V_p. Then if t is already a subterm of some term in E_p, adding t does not increase the DAG size. Then if by removing y from V_p some subterms will be removed from the $Sub(\overline{V_p})$, $|E_{p+1}, \overline{V_{p+1}}|_{DAG}$ will decrease from $|E_p, \overline{V_p}|_{DAG}$.

Suppose x_1 x_2 and x_3 be three variables. and let a b and c be three constants (atoms).

Suppose in a normal attack, the three variables has the following instantiation.

- $\sigma(x_1) = \{\langle a, b\rangle\}^s_{\langle a,b\rangle}$
- $\sigma(x_2) = \langle a, b\rangle$
- $\sigma(x_3) = b$

Suppose in the protocol we can find the following three terms .

- $t_1 = \{\langle a, x_3\rangle\}^s_{x_2}$.
- $t_2 = \langle a, x_3\rangle$
- $t_3 = b$.

Note that in a protocol both variables and constants can appear. See the example of Otway-Rees protocol in [1] (page 457). Since t_1 and t_2 and t_3 are not variables, and $\sigma(t_1) = \sigma(x_1)$ and $\sigma(t_2) = \sigma(x_2)$ and $\sigma(t_3) = \sigma(x_3)$. By the notation defined in Definition 6, we have $t_1 \sqsubseteq_\sigma \sigma(x_1)$ and $t_2 \sqsubseteq_\sigma \sigma(x_2)$ and $t_3 \sqsubseteq_\sigma \sigma(x_3)$.

Now let's start the process of updating E_p and V_p for the variable x_1. We calculate $|E_p, \overline{V_p}|_{DAG}$, which is the number of subterms in E_p and $\overline{V_p}$.

1) $E_0 = \emptyset$, and $V_0 = \{x_1\}$. Choose x_1 to be removed, and choose t_1 in the protocol for x_1 since $\sigma(t_1) = \sigma(x_1)$. The total set of subterms of E_0 and $\overline{V_0}$ is
$$\{\ \{\langle a, b\rangle\}_{\langle a,b\rangle}, \langle a, b\rangle, a, b\ \}.$$
So $|E_0, \overline{V_0}|_{DAG} = 4$.
2) $E_1 = \{t_1\} = \{\ \{\langle a, x_3\rangle\}_{x_2}\ \}$, and $V_1 = \{x_2, x_3\}$. $\overline{V_1} = \{\sigma(x_2), \sigma(x_3)\} = \{\langle a, b\rangle, b\}$. The total set of subterms of E_1 and $\overline{V_1}$ is

$$\{\ \{\langle a, x_3\rangle\}_{x_2}, \langle a, x_3\rangle, a, x_3, x_2, \langle a, b\rangle, b\ \}.$$

So $|E_1, \overline{V_1}|_{DAG} = 7$. Choose x_2 to be removed and choose t_2 for x_2 since $\sigma(t_2) = \sigma(x_2)$.

3) $E_2 = \{t_1, t_2\} = \{\ \{\langle a, x_3\rangle\}_{x_2}, \langle a, x_3\rangle\ \}$. $V_2 = \{x_3\}$. $\overline{V_2} = \{\sigma(x_3)\} = \{b\}$. The total set of subterms of E_2 and $\overline{V_2}$ is

$$\{\ \{\langle a, x_3\rangle\}_{x_2}, \langle a, x_3\rangle, a, x_3, x_2, b\ \}.$$

So $|E_2, \overline{V_2}|_{DAG} = 6$. Choose x_3 to be removed and choose t_3 for x_3 since $\sigma(t_3) = \sigma(x_3)$.

4) $E_3 = \{t_1, t_2, t_3\} = \{\ \{\langle a, x_3\rangle\}_{x_2}, \langle a, x_3\rangle, b\ \}$, and $V_3 = \emptyset$. The updating process finishes. The total set of subterms of E_3 and $\overline{V_3}$ is

$$\{\ \{\langle a, x_3\rangle\}_{x_2}, \langle a, x_3\rangle, x_2, x_3, a, b\ \}.$$

So $|E_3, \overline{V_3}|_{DAG} = 6$.

Note that $|E_1, \overline{V_1}|_{DAG} = 7 > 6 = |E_2, \overline{V_2}|_{DAG}$, since the subterm $\langle a, b\rangle$ appears in $Sub(E_1, \overline{V_1})$, but not in $Sub(E_2, \overline{V_2})$. The property of (5) is violated. An argument to invalidate the counter example may suggest that the setting of the counter example will not happen for a normal attack to a protocol. But it is rather non-obvious to justify this argument.

IX. Fixing The Error

In this section we provide our solution to prove a result that is stronger than Theorem 1. The proof uses Lemma 4.

We prove Lemma* 2 first, which is our intuition that leads to the breakthrough. Then the proof of Lemma* 2 can be generalized to prove Corollary* 4. Although Corollary* 4 covers Lemma* 2, we still present and prove Lemma* 2 first since it is more intuitive to us. Then we prove Lemma* 3, which is stronger than Lemma 4. We prove Theorem* 2 which is stronger than Theorem 1, and then the error is fixed.

Fixing the error does not need the results as strong as Lemma* 3 and Theorem* 2. For example during the research of this article we have proved a weaker result first, which is based on Lemma* 2 and Corollary* 4 :

$$|\sigma(Var(\mathcal{P}))|_{DAG} \leq |\mathcal{P}|_{DAG}.$$

This result is stronger than Theorem 1 and is enough to fix the error. However when we try to prove the tight linear derivation bound, discussed below in Section X, we realized that we need to prove the even stronger results, which are Lemma* 3 and Theorem* 2.

Lemma** 2: In a normal attack to protocol $\mathcal{P}$, if $Var(\mathcal{P}) \neq \emptyset$, there is a variable x such that $\sigma(x)$ has a σ match t (denoted as $t \sqsubseteq_\sigma \sigma(x)$), $t \in Sub(\mathcal{P})$, and t is ground.

Proof: Remind that notation $t \sqsubseteq_\sigma \sigma(x)$ means that $t \in Sub(\mathcal{P})$ and t is not a variable and $\sigma(x) = \sigma(t)$.

If $\mathcal{P}$ has no variable, then Lemma* 2 is trivially true. So we only need to consider the cases such that $Var(\mathcal{P})$ is not empty.

Suppose to the contradiction of the lemma, in a normal attack, for each variables x, for each t such that $t \sqsubseteq_\sigma \sigma(x)$, t is not ground (t has a variable as its subterm). Now we go through a process and at the last step of the process we will show that the contradiction assumption cannot be true. The same process can be used to find the variable x as described by Lemma* 2. At every step of the process, say the i^{th} step, $1 \leq i$, a variable x_i and a term t_i is considered.

At the first step, step 1, choose any variable, call it x_1, $x_1 \in Var(P)$. Choose any term, call it t_1, such that $t_1 \sqsubseteq_\sigma \sigma(x_1)$. By lemma 4, such a t_1 always exists. Since t_1 is not a variable and not ground, then t_1 is a composite term.

If for the i^{th} step, t_i is not ground, $i \geq 1$, then the $i + 1^{th}$ step continues this process by choosing any variable, call it x_{i+1} in $Var(t_i)$, and then by choosing t_{i+1} to be any term such that $t_{i+1} \sqsubseteq_\sigma \sigma(x_{i+1})$. Such a t_{i+1} exists by Lemma 4.

For step i, $1 \leq i$, if t_i is ground, then the contradictory assumption is violated, and the Lemma is true.

Following the contradictory assumption, to continue the reasoning, we consider that t_i is not ground. Since t_i is not a variable, t_i must be a composite term; therefore, at the $i + 1^{th}$ step x_{i+1} can always be chosen, which is a strict subterm of t_i. So $|\sigma(x_{i+1})|_{DAG} < |\sigma(t_i)|_{DAG}$. Since $\sigma(x_i) = \sigma(t_i)$, it is true that $|\sigma(x_i)|_{DAG} = |\sigma(t_i)|_{DAG}$; hence, $|\sigma(x_i)|_{DAG} => |\sigma(x_{i+1})|_{DAG}$. It means that $|\sigma(x_i)|_{DAG} = |\sigma(t_i)|_{DAG} > |\sigma(x_j)|_{DAG}$, for $i < j$. So x_i and x_j are two different variables, for $i \neq j$. For $1 \leq i < j$, since $|\sigma(x_i)|_{DAG} > |\sigma(x_j)|_{DAG}$, and $\sigma(x_j) = \sigma(t_j)$, it is true that $|\sigma(x_i)|_{DAG} > |\sigma(t_j)|_{DAG}$, and $x_i \notin Sub(t_j)$. Since set of variables in the protocol P is finite, the process cannot continue forever. Let j be the last step of this process. Then t_j is ground, since otherwise the process can

continue to the $j+1$ step. Since t_j is ground, the assumption contradictory to Lemma* 2 is violated, so Lemma* 2 is proved. ■

Corollary* 4: In a normal attack of a protocol described by $\mathcal{P}$ and σ, let G and G' be any two disjoint sets of variables and $G \cup G' = Var(\mathcal{P})$, assuming $Var(\mathcal{P})$ is not empty, and G is not empty, it is true that there is a variable x in G such that $\sigma(x)$ has a σ match t, i.e. $t \sqsubseteq_\sigma \sigma(x)$, such that $t \in Sub(\mathcal{P})$ and $Var(t) \subseteq G'$.

Proof:

Suppose Corollary* 4 is not true, then for every variable x in G, for every σ match t of $\sigma(x)$, i.e. $t \sqsubseteq_\sigma \sigma(x)$, it is true that $Var(t) \not\subseteq G'$, which means $Var(t) \neq \emptyset$ (otherwise the corollary is trivially satisfied) and $Var(t) \cap G \neq \emptyset$.

We do a process similar to the one in the proof of Lemma* 2 to derive a contradiction. The same process can be used to find the variable x as described by the corollary. Before the start of the process, choose a variable $x_0 \in G$ and a term t_0 such that $t_0 \in Sub(\mathcal{P})$ and $t_0 \sqsubseteq_\sigma \sigma(x_0)$. Such a t_0 exists by Lemma 4. Since we assume Corollary* 4 is not correct, $Var(t_0) \not\subseteq G'$, which means $Var(t_0) \neq \emptyset$ and $Var(t_0) \cap G \neq \emptyset$.

At the i^{th} step, $i \geq 1$, choose a variable, call it x_i, from $Var(t_{i-1})$ such that $x_i \in G$, and choose a term t_i such that $t_i \in Sub(\mathcal{P})$ and $t_i \sqsubseteq_\sigma \sigma(x_i)$. By Lemma 4 such a t_i exists.

By the same reasoning for proving Lemma* 2, no variable will repeatedly appear in this process, i.e. for $i \neq j$, $x_i \neq x_j$, and this process will end since G is finite; hence, at the end of this process, say at the w^{th} step, there is variable x_w, $x_w \in G$, such that for every term t_w such that $t_w \in Sub(\mathcal{P})$ and $t_w \sqsubseteq \sigma(x_w)$ (t_w always exists by Lemma 4), it must be true that $Var(t_w) \cap G = \emptyset$, which means $Var(t_w) = \emptyset$ or $Var(t_w) \subseteq G'$, and the contradictory assumption is violated; thus, Corollary* 4 is true. ■

Lemma* 3: In a normal attack to a protocol described by $\mathcal{P}$ and σ, for every variable x such that $x \in Var(\mathcal{P})$ the following fact is satisfied: For every term y such that $y \in Sub(\sigma(x))$, there is a term t such that $t \in Sub(\mathcal{P})$ and t is not a variable and $y = \sigma(t)$.

Proof: We use a process to enumerate the variables in $Var(\mathcal{P})$, and Lemma* 3 is proved by induction along this process. At every step of the process, say the i^{th} step for $i \geq 0$, A set of variables V_i is considered.

Before the start of the process, at step 0, $V_0 = Var(\mathcal{P})$.

At the i^{th} step, $i \geq 1$, if $V_{i-1} \neq \emptyset$, a variable in V_{i-1} is chosen, call it x_i, such that the following condition is satisfied:

- There is a term t_i, $t_i \in Sub(\mathcal{P})$, t_i is not a variable, $\sigma(t_i) = \sigma(x_i)$ (i.e, $t_i \sqsubseteq_\sigma \sigma(x_i)$), and $Var(t_i) \subseteq (Var(\mathcal{P}) - V_{i-1})$.

Then $V_i = V_{i-1} - x_i$. The process ends at the i^{th} step if $V_i = \emptyset$.

We prove by induction on the step number i, $i \geq 1$, that for x_i the fact stated in Lemma* 3 is true.

Base case: for $i = 1$. $Var(\mathcal{P}) - V_0 = \emptyset$. Consider $\emptyset \subseteq \emptyset$. By Lemma 1*, we can always find a variable x in $Var(\mathcal{P})$ such that there is a term t in $Sub(\mathcal{P})$ and t is not a variable and t is ground, which means $Var(t) = \emptyset$, and $\sigma(x) = \sigma(t)$. x is the chosen variable x_0. $Sub(\sigma(x)) = Sub(\sigma(t)) = Sub(t)$. Since t is ground and $t \in Sub(\mathcal{P})$, obviously $Sub(t) \subseteq \mathcal{P}$. So the base case is proved.

Induction step: Suppose at step i, $i > 1$, it is true that for each variable x_j chosen at steps j, $j < i$, the fact stated in Lemma* 3 is true. Note that for each variable Y in $Var(\mathcal{P}) - V_{i-1}$, Y has been enumerated in this process, and by the induction hypothesis each term in $Sub(\sigma(Y))$ has a σ match in $Sub(\mathcal{P})$. At step i, let x_i be a variable chosen from V_{i-1}, as described by the process, such that there is a term t_i, $t_i \in Sub(\mathcal{P})$, t_i is not a variable, $\sigma(x_i) = \sigma(t_i)$, and $Var(t_i) \subseteq (Var(\mathcal{P}) - V_{i-1})$. By Corollary* 4 such a t_i always exists. Let $G_i = \{g | g \in Sub(t_i)$ and g is not a variable$\}$. The we have the following result:

$$Sub(\sigma(x_i)) = Sub(\sigma(t_i)) = \sigma(G_i) \cup Sub(\sigma(Var(t_i))).$$

For any term w such that $w \in Sub(\sigma(x_i))$, there are two cases.

1) $w \in \sigma(G_i)$. Since for any term g in G_i, g is not a variable, and $g \in Sub(\mathcal{P})$, the fact of Lemma* 3 is satisfied in this case.
2) $w \in Sub(\sigma(Var(t_i)))$. Then there is a variable x_j for $j < i$ such that $w \in Sub(\sigma(x_j))$. By the induction hypothesis, there is a term t such that t is not a variable and $t \in Sub(\mathcal{P})$ and $w = \sigma(t)$. So the fact of Lemma* 3 is satisfied in this case.

So the induction step is proved, so is Lemma* 3. ■

Theorem* 2: $|\sigma(\mathcal{P}))|_{DAG} \leq |\mathcal{P}|_{DAG}$.

Proof: Let $G = \{g | g \in Sub(\mathcal{P})$ and g is not a variable$\}$. It is true that

$$Sub(\sigma(\mathcal{P})) = \sigma(G) \cup Sub(\sigma(Var(\mathcal{P}))).$$

By the Lemma* 3, for each term t such that $t \in Sub(\sigma(Var(\mathcal{P})))$, there is a term g such that $g \in Sub(\mathcal{P})$ and g is not a variable and $\sigma(g) = t$, which means that $g \in G$ and $t \in \sigma(G)$; therefore, $Sub(\sigma(Var(\mathcal{P}))) \subseteq \sigma(G)$; therefore, the above equation can be modified as

$$Sub(\sigma(\mathcal{P})) = \sigma(G) \cup Sub(\sigma(Var(\mathcal{P}))) = \sigma(G).$$

Now we can prove that

$$|\sigma(\mathcal{P}))|_{DAG} = |Sub(\sigma(\mathcal{P}))|_{card} = |\sigma(G)|_{card} \leq |G|_{card} \leq |Sub(\mathcal{P})|_{card} = |\mathcal{P}|_{DAG}.$$

■

Theorem* 2 is stronger than Theorem 1 and implies Theorem 1.

X. Justifying the Polynomial Bound on Derivation Length

There is another fact that is not proved by [1], but is needed for proving the NP-completeness result.

For each received message R_h, the attacker needs to construct it based on its initial knowledge S_0 and the set of messages sent earlier by regular agents, $\{S_1, S_2, \cdots S_{h-1}\}$, before R_h is received by a regular agent. Then there must be a derivation $E \rightarrow_{l_1} E_1 \rightarrow_{l_2} \ldots E_j$, where $E = \{S_0, S_1, S_2, \cdots S_{h-1}\}$, and $R_h \in E_j$. In order to show the NP-completeness result, it must be shown that the cost of applying these rules the $l_1, l_2, \ldots l_j$ must be polynomial in the size of the input of the problem $n = |\mathcal{P}|_{DAG}$. By Remark 2, it is obvious that to determine if a rule is applicable, and to apply a rule, need only polynomial time, actually linear time. So the remaining question is whether j is polynomial in n or not.

The authors of [1] consider the number of rules in each of the derivation is at most n, which is mentioned in the Non-deterministic procedure in Fig. 1, and the first paragraph in Section 3.4.1. There is no explanation for this. The reason seems to be Proposition 1. Proposition 1 shows that $m \leqslant |t, E|_{DAG}$, where m is the number of rules in the derivation for term t based on E. If Proposition 1 shows that the number of derivation rules is no more than n, then [1] must have considered that

$$|R_h\sigma, S_0\sigma, S_1\sigma, S_2\sigma, \cdots S_{h-1}\sigma|_{DAG} \leqslant n. \qquad \text{(Linear Derivation Length)}$$

However the above equation is not proved. Note that without applying the substitution σ, the above equation is obviously true. However, with the substitution σ, its proof may not obvious at the first glance.

If we try to prove the above equation of linear derivation length using Lemma 4, which shows that for every variable x $\sigma(x) \in Sub(\sigma(\mathcal{P}))$. However Lemma 4 does not show that $Sub(\sigma(x)) \subseteq Sub(\mathcal{P})$; therefore, we cannot directly prove a stronger result supporting the linear derivation bound equation that $|Sub(\sigma(\mathcal{P}))|_{card} < |Sub(\mathcal{P})|_{card}$.

Suppose Theorem 1 is true (although it is not proved in [1] due to the error mentioned in the above section). We can have

$$|R_h\sigma, \sigma S_0, \sigma S_1, \sigma S_2, \cdots \sigma S_{h-1}|_{DAG} \leqslant n^2.$$

The reason is that there are at most n variables appearing in the DAG representation of $(S_0, \mathcal{P})$, and also at most n variables in in $\{R_h, S_0, S_1, \ldots, S_{h-1}\}$, and the instantiation of each variable by σ has the DAG size at most n. Theorem 1 bound the size of the instantiation of single variable in a normal attack, not for all variables. For two different variables x and x', $Sub(x\sigma) \cap Sub(x'\sigma)$ could be empty, which implies the above n^2 bound.

By Theorem* 2 it is obvious that

$$|R_h\sigma, \sigma S_0, \sigma S_1, \sigma S_2, \cdots \sigma S_{h-1}|_{DAG} \leqslant |\mathcal{P}|_{DAG} = n.$$

Since there are at most n rule in a derivation, and there are at most n derivation needed for the attack, and the cost to apply a single rule is n, the total cost to generate all received messages by the non-deterministic algorithm is $O(n^3)$, as stated in [1] at the beginning of section 3.4.1.

XI. On the Related Paper [3]

The more recent paper [3] clarifies the modeling in several aspects comparing to [1].

- The definition of a normal attack is directly based on DAG size, while in [1] in addition to DAG size, a special size $|.|$ is defined, which is not necessary. Also the proof of the corresponding result of Lemma 4 in [3] does not depend on the special term $Charlie$, which is used in [1].

- The definition of a session is clarified. In a session only a subset of the protocol rules are included, while each rule can appear at most once.
- The idea that variables first appear in some received message is clarified.

However we have checked the proofs of [3] and we consider there is still some problem of justifying the corresponding result of Theorem 1 after the corresponding result of Lemma 4 is proved. We consider that the case of not allowing XOR operators is a special case of allowing XOR, and in the ideal situation the proofs of [3] should also prove the NP result of checking secrecy without XOR. With this special interest, just assuming the XOR operators are not allowed, a lot of details of the proofs of [3] can be omitted. But we think that the error of [1], as discussed by this article, is not fixed in [3]. The definition and the presentation of oracle rules in [3] need to be clarified.

XII. Summary

We humbly point out an error and the aspects that can be improved of [1]. Although [1] has proved a very important result, which is Lemma 4, there is an error in the proof of Theorem 1, which seemly is due to a wrong assumption that the DAG size of an instantiation of a term T is no less than the DAG size of the term, i.e. $|T|_{DAG} \leq |T\sigma|_{DAG}$, for a term T and a substitution σ, which is wrong. We provide a non-trivial solution to fix and prove a result that is stronger than Theorem 1. Besides, the proof of Lemma 4 is improved and the linear bound on the derivation length of driving a received message is justified.

We suggest that an improved proof of NP-completeness of checking secrecy can be established with the following preferred features.

- The setting is to bound the number of role instances, instead of sessions. As discussed earlier, the NP-complete result of assuming a bound of the number of sessions does not imply exactly the corresponding complexity result of assuming a bound on the number of role instances. We believe that considering a bound on role instance, the proving methods of this paper can be adapted to prove the NP result.
- A normal attack is defined directly using DAG size, similar to [3], as demonstrated in the improved proof of Lemma 4.
- The initial knowledge of the attacker is clarified. Also the cases allowing the attacker to generate nonces are explicitly considered, or the fact the attacker does not need to generate any nonce is justified, as showed by this article.
- A deterministic model checker, with soundness and completeness, is proposed. And the NP proof is to check the shortest possible branch of the deterministic checker that can find an attack. This kind of presentation can have several advantages as follows.
  - The idea of normal attack and its practical meaning for designing a deterministic model checker is clarified.
  - Unification is introduced while the DAG data structure can be used for linear unification cost. The motivation of using DAG representation is to avoid exponential cost to apply substitution to terms. Similarly using DAG can to avoid exponential cost of unifying of two terms. However in [1] there is no argument where to unify two terms is required. In the papers addressing the deterministic algorithms of checking cryptographic protocols such as [9] and [8] unification is used extensively.
  - Lemma 4, which is the most close step for proving Theorem 1 by [1], can be proved more intuitively. Imagine running a model checker like Athena [9] or constraint solving [8]. For a variable x received in a message, the instantiation of x must be used in some way in the attack. x may either be unified with a term sent in a role instance or be unified with an atom initially known to, or created by, the attacker. The instantiation of x should not be more complex than the requirements of the unifications related to x in a normal attack. Then lemma 4 may be proved following this kind of argument.
- The key features of the NP-completeness proof are clarified so that the proof can be extended, following a general template, to cover other cases such as when XOR is allowed.
- The NP-hardness proof should directly address the protocols as communication sequences while the secret term is declared practically such as the secret term specified in the well-known attack to the public key Needham-Shroeder protocol [13]. In the reduction of the NP-hardness proof in [1] the secret term is sent in a message that is not encrypted, which means such a secret term will always be leaked if the protocol is a communication sequence. This kind of non-satisfying aspects have been discussed in [14] and [15].

We especially appreciate the discussion with Dr. Vitaly Shmatikov on the research of this article.

REFERENCES

[1] 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.

[2] M. Turuani, Weg page, http://www.loria.fr/~turuani/. [Online]. Available: http://www.loria.fr/~turuani/

[3] 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.

[4] R. M. Amadio, D. Lugiez, and V. Vanackère, "On the symbolic reduction of processes with cryptographic functions," *Theor. Comput. Sci.*, vol. 290, no. 1, pp. 695–740, 2003.

[5] 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.

[6] 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.

[7] Ferucio L. Tiplea and C. Enea and C. V. Birjoveanu, "Decidability and complexity results for security protocols," in *Verification of Infinite-State Systems with Applications to Security*. IOS Press, 2006, pp. 185–211.

[8] 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.

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

[10] H. Comon-Lundh and V. Cortier, "Security properties: two agents are sufficient." *Sci. Comput. Program.*, vol. 50, no. 1-3, pp. 51–71, 2004.

[11] G. Lowe, "Towards a completeness result for model checking of security protocols," in *Journal of Computer Security*. Society Press, 1998, pp. 96–105.

[12] ——, "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR," in *TACAS*, 1996, pp. 147–166.

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

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

[15] Z. Liang and R. M. Verma, "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).