# Real Cryptographic Protocol with an Insider Attacker: Improving Techniques for Proving Undecidability of Checking Security Goals

Zhiyao Liang and Rakesh M. Verma

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

Technical Report Number UH-CS-08-02

February 27, 2008

**Keywords:** Cryptographic protocols, secrecy, authentication, insider, undecidability, formal method.

## Abstract

Existing undecidability proofs of checking secrecy of cryptographic protocols have the limitations of not considering protocols common in literature, which are in the form of communication sequences, since only protocols as non-matching roles are considered, and not considering an attacker who is an insider since only an outsider attacker is considered. Therefore the complexity of checking the realistic attacks, such as the attack to the public key Needham-Schroeder protocol, is unknown. The limitations have been observed independently and described similarly by Froschle in a recently published paper [1], where two open problems are posted. This paper investigates these limitations, and we present a generally applicable approach by reductions with novel features from the reachability problem of 2-counter machines, and we solve the two open problems. We also prove the undecidability of checking authentication which is the first detailed proof to the best of our knowledge. A unique feature of the proof is to directly address the secrecy and authentication goals as defined for the public key Needham-Schroeder protocol, whose attack has motivated many researches of formal verification of security protocols. This report covers our workshop paper [2] and provide more details of modeling and proofs.

*Research supported in part by NSF grants CCF 0306475 and DUE 0313880.

# Real Cryptographic Protocol with an Insider Attacker: Improving Techniques for Proving Undecidability of Checking Security Goals

Zhiyao Liang and Rakesh M. Verma

**Abstract**

Existing undecidability proofs of checking secrecy of cryptographic protocols have the limitations of not considering protocols common in literature, which are in the form of communication sequences, since only protocols as non-matching roles are considered, and not considering an attacker who is an insider since only an outsider attacker is considered. Therefore the complexity of checking the realistic attacks, such as the attack to the public key Needham-Schroeder protocol, is unknown. The limitations have been observed independently and described similarly by Froschle in a recently published paper [1], where two open problems are posted. This paper investigates these limitations, and we present a generally applicable approach by reductions with novel features from the reachability problem of 2-counter machines, and we solve the two open problems. We also prove the undecidability of checking authentication which is the first detailed proof to the best of our knowledge. A unique feature of the proof is to directly address the secrecy and authentication goals as defined for the public key Needham-Schroeder protocol, whose attack has motivated many researches of formal verification of security protocols. This report covers our workshop paper [2] and provide more details of modeling and proofs.

**Index Terms**

Cryptographic protocols, secrecy, authentication, insider, undecidability, formal method.

## I. Introduction

SINCE networks are indispensable to all kinds of communications nowadays, checking and analyzing cryptographic protocols are especially important. A significant research direction is to check secrecy and authentication of protocols against a dominating attacker, first introduced by Dolev and Yao [3], assuming the cryptographic primitives cannot be broken. Since Lowe discovered an attack (listed in Fig. 2 of Appendix VI-A) on the public key Needham-Schroeder protocol (PKNS protocol) [4] 17 years after it was published [5], many papers have focused on the topics in this area using formal methods. An essential part of these researches is to investigate the complexity of checking security protocols. In this paper we focus on the undecidability results.

Undecidability results are important practically since they are helpful to understanding the problem and to find scenarios that are decidable. Provided with precise undecidability results, people can focus on more promising directions to find decidable or semi-decidable subcases, or to avoid the trouble by following some prudent engineering approach to design protocols so that checking the security is decidable, or to understand better the limitations of some automatic verifier or model checkers, such as why their termination cannot be proved, or why they detect false attacks. In general the more precise and specific undecidability results are stronger and more helpful.

Two questions need to be asked in order to judge and improve the quality of proving undecidability. 1) Do the obtained undecidability results exactly cover the realistic problems of practical interest? If not, the complexity of solving the target problem may still remain unknown, and researchers could spend more effort on proving decidability without progress, if the problem is actually undecidable. For example, Froschle has considered to decidability of the two open problems in [1] as possible future research directions, if they are not undecidable. 2) Is there a general and powerful approach so that the undecidability of different cases can be proved similarly, or even semi-automatically by some programs? Since there are different security goals, different protocols have

*Research supported in part by NSF grants CCF 0306475 and DUE 0313880.

different requirements, and researchers are analyzing problems with very specific requirements, inevitably there will be different tasks of investigating the undecidability of checking security protocols. It is ideal to have a common proving approach so different proofs could be discovered similarly while only the exact meaning of the parameters of the proving template need to be customized. The idea agrees in spirit with some techniques of artificial intelligence, and automatic theorem proving which take advantages of reusable patterns to provide solutions. Therefore this research has the potential to utilize these techniques for automation of proofs. This paper is devoted to the answers of the two questions.

## A. Limitations of Existing Undecidability proofs

Undecidability of secrecy checking has been mentioned by other researchers in several papers [6] [7] [8] [9] [10] [11] [12], and [8] [9] [11] provide proofs with details. In [9] and [8] the authors used MSR (multi-set rewriting) to analyze protocols. The proof is by a two-stage reduction from the halting problem of Turing machine with the style of Turing machine tableau to Horn clause theories without function symbols and then from Horn clause theories to protocols specified as a set or roles. In [11] the authors showed that the undecidability result of [8] can be proved more directly by a reduction from the reachability problem of a 2-counter machines to the secrecy checking problem of protocols as sets of roles (we call them role-oriented protocols).

The above proofs of undecidability have three limitations. **First**, all of the undecidability proofs, except [7], do not consider a protocol as a sequence of message exchanges, which we call a ***communication sequence***, or ***CS*** for short. The published protocols we have noticed (see the protocol library [13]) are all in the form of communication sequences. These undecidability proofs directly consider a protocol as a set of roles, we call this kind of protocols ***Role Oriented*** or ***RO*** for short, where each role is a sequence of message sends and receives executed by a principal (also called an agent). Usually the first step to analyze a protocol as a communication sequence is to translate it into a set of roles, where the actions executed by the same agent are organized into a role, and the relative order of actions in the communication sequence is kept. Every step of the communication sequence implies two actions, one is the message sending, marked with a '$+$', from the sender's role, and the other is the corresponding message receiving, marked with a '$-$', in the receiver's role. In a CS we only list the sender's actions but without the $+$ marks, while the corresponding receiver's actions are implicit. As an example, the CS and roles of the core of the public key Needham-Schroeder protocol [5] (PKNS) is listed in Appendix I.

However, there is a difference between an RO protocol translated from a CS, and a RO protocol directly designed without considering the corresponding CS. The reason is that the first one is ***matching***, in the sense that every message sending (or receiving) action in a role can always be matched with (be unified with) a unique message receiving (or sending) action in another role. The second one could be ***non-matching***. We call a $RO$ protocol non-matching if for some message received (sent) in a role, the other corresponding role in the protocol where this message is sent (received) does not exist. In other words, a set of non-matching roles are impossible to be obtained by parsing any CS. In the proofs of undecidability of secrecy checking mentioned above, except [7], the protocols considered are non-matching RO.

In [7] the authors showed a proof of the undecidability of secrecy checking by a reduction from the reachability problem of Petri nets. The protocol constructed in the reduction is called 'real' and has at least one honest run. A 'real' protocol is equivalent to what we call a communication sequence in this paper. However, as noticed by Froschle in [1], the proof of [7] depends on messages with unbounded size in a run, which is a part of the motivation of the first open problem of [1] (quoted later in this paper). Bounding message size in a run (or only consider runs where message size is bounded) is a condition making the undecidability stronger and more interesting as proposed by [8]. The proof of [7] has another limitation of considering an outsider attacker (the third limitation, described later).

**Second**, improper secrecy declaration. This limitation, described below, is directly related to the first limitation and can show further why the proofs are not quite suitable for practically designed protocols. The proofs (except [7]) have some role where a term, which is declared as the secret one, is sent out in a message, not encrypted or trivially encrypted so that the attacker can know the secret term once he can get the message. Suppose this role belongs to a set of roles which are translated from some practically designed protocol as a CS, we can see that an honest run of the protocol (running the CS) will inevitably send the message containing a secret term and the attacker can always know the secret term. In other words, secrecy checking for protocols in the form of a CS with this 'trivial' secrecy declaration is always decidable, and the undecidability proofs will not work.

Froschle has independently noticed the above two limitations in a recently published paper [1]. She mentioned protocols having an honest run or ‘real’ protocols (notions first used by [7]), and proposed the first open problem, as quoted below.

“A protocol has an *honest run* if all of its rules can be played in the given orders: the first rule, then the second, and so on. ”

*Problem 1.* “Is Insecurity decidable for protocols with honest runs when the message size is bounded?”

**Third**, considering the attacker as an outsider. Gollmann mentioned in [14] that in the attack to the PKNS protocol [4] the attacker should be considered as an insider. The reason is that in order for the attacker $C$ to carry out the attacker, $C$ has to wait for an honest agent $A$ to send a message to $C$ first, while $A$ knows $C$’s name and public key. Why $A$ can contact $C$ actively? Saying that $C$ is an insider that $A$ knows could be a convincing answer.

We further explain the intuition of the insider attacker by emphasizing that the insider attacker should belong to a group of agents together with other regular agents such that the group is established in the initial stage of a run, which is an assumption of the protocol run, and the attacker shares the same initial knowledge pattern with the other agents in this group. Every protocol assumes a perfect *initial knowledge establishing stage* at the beginning of a run. Note that if the initial knowledge of agents are not established perfectly and securly, there is no way to guarantee any security of the protocol. In this stage, keys and other terms will be distributed among a group of agents following some initial knowledge policy required by the protocol. We call this group as the ***legitimate agents group***. We call every agent belonging to this group as an ***insider*** to this group, and every other agent as an ***outsider*** to this group. The agents in this group knows each other and can contact each other actively.

All of the proofs, cited earlier, assume there is some term, say a key $K$, which is known (initially) to all agents (other than the attacker) participating a run of the protocol, but the attacker does not know the term. This restriction makes the reasoning of the proofs easier since for any encryption appearing in a run where the encryption key is $K$, it is guaranteed that the encryption is not created by the attacker but by some regular agent. Also in these proofs a regular agent will only try to contact other regular agents, not the attacker. It is clear that the proofs assume that the attacker does not participate in the initial knowledge establishing stage as other agents did, and the attacker is an outsider, while all other agents in the run, who are honest, are insiders.

In practice the terminologies of “insider” and “outsider” are used to describe some real security cases. Despite some difference of the details among cases, essentially the behind meaning is the same. So using this two terminologies here is quite appropriate. Further discussion is presented in Appendix II. Actually to consider an insider attacker has special importance, since in practice, the majority of security failures in a system may be due to an insider attacker [14].

Froschle also independently noticed that in the existing undecidability proofs the attacker is different from any regular agents in terms of initial knowledge, which is not very “natural”. She described the notion of natural key policy, and the second open problem in ([1]) as follows.

“A protocol has a *natural key policy* if in the specification of the initial key knowledge the Intruder is treated like all other principals.”

*Problem 2.* “Is Insecurity decidable for protocols with a natural key policy when the message size is bounded?”

Note that the existing undecidability results cannot cover the two open problems of [1], therefore their solutions are unknown. Furthur discussion on the two open problems are presented in Appendix III.

A unique feature of our proof is that we directly address the secrecy goal and the attacker as in the public key Needham-Schroeder protocol, which has been a phenomenal research focus, therefore we exactly address the real potocols and an insider attacker.

Our reduction scheme using 2-counter machine is inspired from [11]. However, there are some key differences. First, the proofs in [11] assume non-matching RO protocols and an outsider attacker and we need new ideas of reduction to avoid the limitations. Furthermore, we have found and fixed two minor errors in the work of [11]: a counter can be negative, and zero can be used as a positive number. [15]. The proof of correctness of the reduction in [11] is sketchy and consequently misses the two errors. Details of the errors and our fixes in the Appendix of [15]. We have adapted the approach of using 2-counter machine in [15] to prove the undecidability of an open problem pinpointed by [8]. Our work in [15] give us some confidence and motivation in finding a general and powerful proof scheme. However the proof in [15] also has the limitations of assuming non-mathing RO protocols and an outsider attacker.

## II. A General Template for Proving Undecidability of Checking Security Protocol

We present a proof template here. Section IV demonstrate in details how this template is fulfilled into actual proofs.

1) Modeling. The protocol is a set of matching roles which can be translated from a communication sequence. We prefer a modeling which clarifies a set of concepts: communication sequence versus roles; matching roles versus non-matching roles; role versus role instance; regular agent versus attacker; the agents participate in a run versus the agent names appearing in the protocol; term template versus term instance; the initial knowledge patterns of the attacker and the regular agents. Each role is a sequence of action templates. A protocol run consists a set of role instances. The actions of these role instances can either be considered as in a linear trace (no two actions can occur at the same time) by interleaving the role instances, or equivalent the earlier relationship between two actions is a partial order while two actions are allowed to occur at the same time. We will discuss the two options later in Section III. The definition of a security goals such as secrecy or authentication should be the same as what are defined in the literature for realistic protocols independent to the choice of proof techniques. We incorporate the PKNS protocol and its secrecy and authentication goals into the proof.

2) Encoding. A configuration of a 2-counter machine in a computation (defined later in Section IV) is represented as a term in a special form in a protocol run. Also a counter value is represented by a specific term in a protocol run. We suggest that the encoding scheme for a configuration and a counter value can be very flexible. For example, in the published proofs two consecutive locations of a turing machine tape or two consecutive counter values of a 2-counter machine are represented by a term $\{X_1, X_2\}_K^{\leftrightarrow}$, where $X_1$ and $X_2$ are atomic terms, usually nonces. However, a working encoding scheme can be in other forms, i.e., $X_1$ and $X_2$ could even be composite terms, as long as the correctness of the reduction can be justified. The flexibility of designing the encoding schemes is a source of the power of the proving system.

3) Translating a 2-counter machine into a protocol. Each transition rule of a deterministic 2-counter machine $M$ is translated into a distinct role (or even a distinct set of roles) of the corresponding protocol $Pro$. There are possibly 25 different combinations of a transition rule of $M$, depending to whether a counter will increase, decrease or remain the same. There is no need to explicitly describe the 25 different ways of translation. Instead, a set of rewriting rules, and a general role template are provided for the translation. When a transition rule of $M$ is translated, the rewrite rules will be applied as much as possible to the general role template to remove some terms and the result is the translated role. In addition, a set of starting roles and a set of final roles are needed. The starting roles are used to provide the initial configuration term which encodes the initial configuration of $M$. The final roles are used to show that when a final configuration term is produced in a run, which encodes a final configuration reached by $M$ in a computation, the security goal is violated. The design of the final roles are flexible, but should capture the target security goal precisely. We incorporate the PKNS protocol into the final roles. A set of helper roles may also be needed to reproduce a term into a different form, such as the copy roles in the proof of this paper (introduced later).

4) The two-directional proof. Direction 1: to prove that if $M$ can reach a final configuration, there is an attack of $Pro$ that violates the security goal. The attack can be constructed step by step following the computation of $M$. I.e., when a transition rule $t$ is applied during the computation of $M$, suppose $t$ is translated into a role $R$ of $Pro$, then a role instance of $R$ is executed. The configuration term generated from the earlier step is used as a input in the current step. When the final configuration term is generated, it triggers the security failure, e.g., the attack to the PKNS protocol is carried out. A lemma can be used to show that for each message received by a regular agent in a run, the attacker can construct it before it is received.

Direction 2, to prove that if $Pro$ has an attack where the security goal is violated, then the two counter machine can reach a final configuration in a computation. Several observations need to be justified to show the soundness of the encoding scheme: every term can only encode a unique number; it is impossible that a term can encode a negative number; A term encoding zero (or a positive number) cannot be used to encode a positive number (or zero). Then we need to show that the security goal can be violated only if a final configuration term can be produced. This can be proved by a stronger lemma showing that for each configuration term $T$ generated in a run of $Pro$, $T$ encodes a reachable configuration of the 2-counter machine $M$. Since a final configuration term of $Pro$ can only encode a final configuration reachable to $M$, this direction is proved.

The "parameters" of this proving system are the encoding schemes and the translation scheme, and the design of initial and final roles (aspects 2 and 3). To prove the undecidability of a different problem, we only need to provide the exact choices for these parameters, while the modeling is the same, and the 2-directional proof should be largely identical. Choosing the arguments of the parameters may require a little creativity from user and full automation seems to be not easy now. However we do notice that there are considerable reusable pattern for customization of the arguments to these parameters. For example, the logic behind the rewrite rules are the same for different problems so it is straightforward to adjust the representation of the rewrite rules for different problems. The 2-directional reasoning requires a lot details, especially for complex problems, and consumes most of the space of a proof. However the 2-directional proof is suitable for using mechanical tools thus manual mistakes and labors could be avoided.

## III. Modeling

### A. Notations

Notations are chosen in a style that is commonly used in the literature, e.g., [16]. The notations for asymmetric keys and the idea of symbols are new. Notations are described by the following grammar, followed by some explanations.

$$
\begin{array}{lll}
String & ::= & Lowercaseletter \mid Uppercaseletter \mid Number \mid StringString \\
Symbol & ::= & String \mid String^{String} \mid String_{String} \\
Asymmetrickey & ::= & k^1_{Symbol} \mid k^0_{Symbol} \\
Atomicterm & ::= & Symbol \mid Asymmetrickey \mid k_{Symbol:Symbol} \\
Term & ::= & Atomicterm \mid [Term, Term] \mid \{Term\}^{\rightarrow}_{Asymmetrickey} \mid \{Term\}^{\leftrightarrow}_{Term}
\end{array}
$$

A ***symbol*** is a string of letters and numbers, possibly with a superscript or subscript, which is also a string of letters and numbers. A ***variable*** is a symbol with at least one uppercase letter, such as $N_A$, $A2$, $A$. The meaning of a symbol will be clear in the context of the analysis of a protocol, for example a set of agent names will be defined in a run of a protocol. A symbol could be an agent, usually in the form of $A$, $A1$, $B$, or a nonce, usually in the form of $N_A$, $n_b$.

A pair of asymmetric keys is represented as $k^1_X$ and $k^0_X$. $X$ is the unique ID (UID) of the asymmetric key pair. This notation can also describe the asymmetric keys generated during a run. When $X$ is the name of an agent, $k^0_X$ and $k^1_X$ represent the established private key and public key of the agent $X$, respectively. When $X$ is not a name of an agent, it must be a unique symbol representing the UID of the key pair, and $X$ should not be used alone or known by any agent as an explicit term. Then the superscripts 0 and 1 only indicate that they are inverse to each other, and the choice of 0 or 1 is arbitrary and they do not mean which one is public or private.

A ***constant*** is an atomic term with no uppercase letter, such as $a$, $n_a$, $r_1$.

An ***atomic term*** is a symbol, an asymmetric key, or an established symmetric key. An established symmetric key shared by $A$ and $B$ is represented as $k_{A:B}$, where $A$ and $B$ should represent some agent names. The lowercase $k$ is reserved as a special notation to describe keys. When a new symmetric key is dynamically generated in a run, it can be any term and should not be described with the form $k_{A:B}$.

A ***term*** is an atomic term, or a list, or an asymmetric encryption, or a symmetric encryption. A list has the form of $[X, Y, \cdots]$, where $X$ and $Y$ are terms and the list contains finite number of member terms. A list is a simpler representation of a sequence of nested pairs. For example $[W, X, Y, Z]$ is the same as $[W, [X, [Y, Z]]]$. When a message is a list, the top level enclosing [ ] is omitted. An asymmetric encryption, has the form of $\{T\}^{\rightarrow}_{k^i_A}$, $i \in \{0, 1\}$, where $T$ is the encrypted term, and $k^i_A$ is the atomic encryption key. A symmetric encryption has the form of $\{T\}^{\leftrightarrow}_{Y}$, where $T$ is the encrypted term and $Y$ is the term working as the encryption key. For both asymmetric or symmetric encryption, when a list, say $[X, Y, Z, \cdots]$ is encrypted, the enclosing square brackets are removed from within "{ }". The word ***ground*** means variable free.

An atomic term is the smallest indecomposable term. A symbol is the smallest unit to construct a representation or a name of a term. The notation $k^0_a$ is still an atomic term, and the subscript/superscript are only used to describe some attributes of the key.

When a term is not an atomic term, it is called a ***composite term***. A composite terms is allowed to be the key (composite keys) for a symmetric encryption, but a key for an asymmetric encryption should be an atomic term. This assumption about composite keys and atomic keys can show the difference between different encryption algorithms, and is also assumed by other papers including [17] and [18].

It is helpful to clarify the difference between term templates and term instances. Term templates, or simply terms, can contain variables, while term instances are ground terms. How a variable is instantiated will be clear when we describe a run of a protocol. A ***message*** is a term. Every message appearing in a run is a ground term.

Often, constants can appear in a protocol, which can be understood as special terms that have some fixed value, such as the fixed name of the server. We will explain the meaning of a term when necessary. A special constant is the upper case letter $I$ that is reserved as the name of the attacker. $I$ commonly stands for “intruder”, which is proper since in all other papers we have noticed analyzing the complexity of checking security protocol the attacker is an outsider in the reductions. The terminologies of insider and outsider will be explained later.

We assume the free term algebra, which means that there is only one way to construct a term. In other words, given two terms $T_1$ and $T_2$ that are constructed differently, the bit-string that $T_1$ represents must be different from the one that $T_2$ represents. This assumption is commonly made in papers based on the Dolev-Yao model.

## *B. Protocol, Attacker, and Protocol Run*

***Definition*** *1:* An ***agent*** is a tuple $\langle name, init\rangle$, where $name$ is the unique name of the agent, and $init$ is the initial knowledge of the agent. It is a set of terms that are built in an assumed *initial knowledge establishing stage* (explained earlier), and are considered known to the agent initially before a run. If an agent always acts according to the description of the protocol, it is called a ***regular agent***. If the name of an agent is $A$, the $init$ field of the agent is referred as $A.init$.

The initial knowledge patterns of agents will be specified in the protocol. A role (defined later) may specify some requirements of $init$ for the agents who can execute the role. An agent can participate in a run several times, each time executing a different role instance.An agent is called a principal in some papers. The definition of agent can be expanded to address more specific situations, such as to add a memory field [19] which indicates the set of terms an agent has “seen” in a run.

***Definition*** *2:* A ***role type*** or ***role template*** or a ***role*** for short, is a tuple: $\langle agent,\ acts,\ conds\rangle$.

- $agent$ is the term representing the name of the agent who will execute the role. If the agent name is a constant, say $s$, then it means that the role can only be executed by a fixed agent, whose name is $s$, usually a special server. If this field is a term $P$, by convention, we can call this role as ***P*'s role**.
- $acts$ is the sequence of actions of message sends and receives. Each action of a role is a structure of symbols and has a message number assigned by the ***communication sequence*** ($CS$) of the protocol. If an action in $P$'s role sends a message $Msg$, then it has the form
$$n.\ \#_P(T_1, T_2\cdots)\ +(P \Rightarrow B) : Msg \qquad ,$$
where $n$ is the message number (or equivalently step number), $B$ is the intended receiver agent, and the fresh term generation part $\#_P(T_1, T_2\cdots)$ is optional. If an action in $P$'s role receives a message $Msg$, then it has the form
$$n.\ -(B \Rightarrow P) :\ Msg$$
where $n$ is the message number, $B$ is the supposed sender agent of the message. The $acts$ of a role implies a set of variables that appear in $acts$, and they are the symbols with at least one uppercase letter.
- $conds$ reprsents the required conditions on terms in the $acts$ that are not implicitly expressible by the $acts$. These conditions represent some computation features of $P$ at the step of $P$'s program. The common forms of a condition include, but not restricted to: $X \neq Y$ (values of the two terms $X$ and $Y$ are different); $X \notin Q$ (term $X$ is not included in the set $Q$, where $Q$ is defined in the context of the protocol). The $acts$ of a role can be parsed from the $CS$ of a protocol, as showed in Appendix I

***Definition*** *3:* A ***role instance*** is a tuple $\langle rid, agent, role, end, vmap, events\rangle$.

- $rid$ is the unique ID of the role instance. It is for reasoning purpose only and is not necessarily a term.
- $agent$ is the name (a ground term) of the agent who executes the role instance.
- $role$ is the name of role template for this role instance. By convention it is a term representing an agent name.

- $end$ is the largest (latest) message number of the events of the role instance. Suppose the largest message number of $role$ is $n$, then $end \leq n$.
- $vmap$ is a function (a substitution) that maps a variable to a ground term. To apply $vmap$ to a term $T$ can be denoted as $vmap(T)$ or $T \cdot vmap$. For a constant or a composite ground term $T$, $vmap(T) = T$. $vmap$ can also be obviously extended as a substitution to map a term or a condition to its ground instance. Note that $vmap(role.conds)$ should be satisfied.
- $events$ is a sequence of ground events, which instantiate the prefix of the sequence of steps of the $role$, up to message number $end$. Each event is a pair $\langle act, time\rangle$. For an action of $role$ with message number $m$, $m \leq end$, call it $Act_m$, the corresponding event in the role instance has the form $[act_m, time_m]$. $act_m = vmap(act_m)$ is a ground structure of symbols that instantiates $Act_m$. $time_m$ is a positive real number. $time_m < time_{m'}$ if and only if $m < m'$.

A role instance does not need to cover all steps of the corresponding role. Since a role instance should always be discussed in a context of a run, every event is associated with a $time$ field representing its occurring time in a run.

***Definition** 4:* An ***event*** is a tuple $\langle act, time\rangle$, where $act$ is a ground structure of symbols representing an action, and $time$ is a positive real number representing when the even occurs after the start of the run. An event could be of two possible kinds. A ***regular event***, which is an event executed by a regular agent that belongs to a role instance. If a regular event sends a message, then the $time$ field represent the time that the message is sent (the time when the sending action is finished) in the run. If a regular event receives a message, the $time$ field represents the time that the message is received by the agent (the time when the receiving action is finished). An ***attacker event***, which is an event executed by the attacker and does not belong to any role instance. In this paper the $act$ field of an attacker event has only one form: $\#_I(T_1, \cdots T_n)$, which represent the computation of the attacker $I$ to generate the fresh terms $T_1$ to $T_n$. The $time$ field represent the time when $I$ finishes generating these fresh terms in the run.

In the real world in a run of a protocol, the time when message is sent or received is physical. The time field is used to simulate, but not to exactly represent, the physical time of an event. In a symbolic run for our reasoning purpose, we only care about who is earlier or later between two events. The attacker's actions of sending or receiving messages are not recorded explicitly as events in a run since they are all implicit in a run: the attacker, who controls the network, records immediately all message sent by regular agent, and send all messages that are received by regular agents immediately before they are received.

***Definition** 5:* A ***protocol*** $Pro$ is a tuple $\langle PID,\ CS,\ roles,\ AN,\ rsts\rangle$

- $PID$: The unique ID of the protocol, for reasoning purpose only.
- $CS$: The communication sequence.
- $roles$: A set of role templates. $roles$ should be obtained by parsing $CS$ as discussed earlier, and $roles$ are matching.
- $AN$: The set of names of a group of agents (legitimate agents group) who have participated in the initial knowledge establishing stage and ready to participate in a run of $Pro$. If an agent name $X$ is included in $AN$, then $X$ is called an ***insider***, otherwise $X$ is an ***outsider***. Sometimes $AN$ can be split into several smaller groups depending on the situation. $AN$ will be instantiated by a set of agent names in a run. Note that in a protocol $AN$ may only be the name of a set to be referred in $rsts$, and the exact members of this set are not specified until a run of the protocol is discussed.
- $rsts$: The restrictions describing the patterns of the initial knowledge of the agents in $AN$. Note that usually $AN$ are referred in $rsts$ in order to define these patterns. The $rsts$ field also includes the definitions of related sets, such as a set of terms known to a certain group of agents.

***Definition** 6:* The behavior of a ***Dolev-Yao attacker*** [3], or an ***attacker*** for short, is a tuple $[name, init, know_I]$.

- $name$ : By convention $name = I$.
- $init$: A set of ground terms that is the initial knowledge of the attacker. This field can be referred as $I.init$. When a run is discussed the exact terms included in $I.init$ will be specified according to the attacker's initial knowledge pattern specified in the run.
- $know_I$: A function. After a set $E$ of events have occurred in a run (before a certain time), $know_I(E)$ is the set of terms that the attacker can obtain, which will be explained later in this definition.

The behavior of $I$ in a protocol run can be summarized by three aspects:

1) $I$ records every message immediately when the message appears in the network.
2) $I$ can prevent a principal from receiving a message that has been sent.
3) $I$ has the freedom to send out any term to any agent as a message at any time of the run, as long as $I$ can obtain the term before sending it out.

Given a set $E$ of events that have occurred, $know_I(E)$ is calculated as the closure of applying the following rules, each has the form $condition \mapsto Result$

- Initial knowledge: $X \in init_I \quad \mapsto \quad X \in know_I(E)$
- Sent message recording:
  $msg$ is a message sent in a regular event in $E \quad \mapsto \quad msg \in know_I(E)$
- Fresh term generation:
  $X_i$ is a fresh term generated in an attacker event $e$, $e \in E$, i.e., $e$ has the form $[\#_I(X_1, \cdots X_n), time] \quad \mapsto$ $X_i \in know_I(E)$, for all $i$, $1 \leq i \leq n$.
- Synthesis:
  - List construction: $X, Y, Z, \ldots \in know_I(E) \quad \mapsto \quad [X, Y, Z, \ldots] \in know_I(E)$
  - Asymmetric key encryption: $X \in know_I(E)$ and $k_G^i \in know_I(E)$, $i \in \{0, 1\} \quad \mapsto \quad \{X\}^{\rightarrow}_{k_G^i} \in know_I(E)$
  - Symmetric key encryption: $X \in know_I(E)$, and $Y \in know_I(E) \quad \mapsto \quad \{X\}^{\leftrightarrow}_Y \in know_I(E)$.
- Analysis:
  - List breaking: $[X, Y, \cdots] \in know_I(E) \quad \mapsto \quad \{X, Y, \cdots\} \subset know_I(E)$
  - Asymmetric key decryption: $\{X\}^{\rightarrow}_{k_G^m} \in know_I(E)$, $k_G^{1-m} \in know_I(E)$, $m \in \{0, 1\} \quad \mapsto \quad X \in know_I(E)$
  - Symmetric key decryption: $\{X\}^{\leftrightarrow}_Y \in know_I(E)$, $Y \in know_I(E) \quad \mapsto \quad X \in know_I(E)$

***Definition** 7:* A ***run*** of a protocol $Pro$ with an attacker $D$ is a tuple: $\langle Pro,\ D,\ R,\ AN,\ E, conds\rangle$. $Pro$ is the protocol. $D$ is the initial knowledge pattern of the attacker who is involved in the run. $R$: is a set of role instances that are executed honestly by regular agents. $AN$ is the set of ground names of the agents who can participate in the assumed perfect initial knowledge establishing stage of the run, including all of the regular agents and sometimes the attacker, involved in the run. The agents in $AN$ are insiders. $AN$ instantiates the $AN$ field of $Pro$. $E$ is A set of events that have occured. $conds$ is the conditions required, listed as the following.
1) $Pro.rsts$ should be satisfied. That is, for each agent $P$ in $AN$, $P.init$ is assigned with a set of ground terms according to $Pro.rsts$.
2) If $I \in AN$, $I$ is an ***insider attacker***, then its initial knowledge pattern $D$ should be the same as other regular agents described in $Pro.rsts$. Otherwise if $I$ is an ***outsider attacker***, then $D$ could be different from the initial knowledge patterns of other agents. Usually an outsider attacker knows less than an insider, and $D$ may specify that $init.I$ include the names and public keys of all agents in $AN$, and some constants that are known to all agents.
3) The set of ***regular*** events in $E$ include, nothing more and nothing less, all the events of all the role instances in $R$. That is, $\bigcup_{r \in R} r.events$.
4) For any regular event $e = [act, t]$ in $E$, if $e$ receives a message $msg$, let $E_e = \{d|\ d \in E,\ d.time < t\}$, then $msg \in know_I(E_e)$.
The set of all possible runs of a protocol $Pro$ and with some specific initial knowledge pattern $D$ of the attacker, is denoted as $Runs^{D:Pro}$.

Since the attacker's intention to follow the protocol is not clear, only regular agents' events are organized into role instance. The attacker's message sending and receiving behaviors are not described in a run, since they are implicit by the definition. We assume a powerful attacker who can do any amount of computations, as described for $know_I(E)$, in extremely short amount of time, i.e, shorter then the closest time distance between any two events. The attacker controls the network in a superb way such that the attacker can record or send a message in no time.

In general there are two kinds of modeling in terms of describing a run. The first kind, we call trace based modeling, such as [20] and [18], is that a run is considered as a linear sequence (a trace) of all the events, which may imply that no two events occurs at the same time. The second kind, we call partial-order based, such as the strand space model [21] [22], does not require a total order, and the earlier relationship between two events is a partial order, which suggest that two events can occur at the same time. The two kinds of model are equivalent. In [19] and [2] we used trace based model for the proofs. Here we define a run using the partial order based model, since we found that induction based proof can be established on partial order based model without additional complexity, which is interesting to show. Further more we consider traced is more natural than the trace based one and could be more convenient to design checking algorithms (the topics out of the scope of this paper).

***Definition** 8:* Given a protocol $Pro$, a specific pattern $D$ of the attacker's initial knowledge, and a set of secret terms $SEC$ (declared in a certain way), the ***secrecy problem*** is to check the validity of the following statement.

$$\exists run, \exists X, run \in runs^{D:Pro}, X \in SEC : X \in know_I(run.E)$$

It is obvious that assuming more than one attacker is not more dangerous than assuming just one attacker, since when every attacker can control the network, and they can communicate with each other and share information, the result is the same as one powerful attacker. See [23] for more discussion on this topic. We are not considering the special case that possibly there are several attackers and they cannot directly communicate and share information with each other. So it is justified that we only consider one attacker.

In general the number of agents who can participate in a run should be allowed to be unbounded. In [24] the author proved that the number of agents can be considered bounded for analysis of security protocols. However we do not need to bound the number of agents for the proof in this paper.

Since we address the undecidability of analyzing a protocol, the number of role instances should not be bounded. It has been proved that, e.g. [17], if the number of role instances is bounded, secrecy is decidable.

## IV. Proving Undecidability of Checking Secrecy and Authentication

We want to prove that, an arbitrary deterministic 2-counter machine [25] with no input, call it $M$, can reach its final configuration, if and only if the corresponding protocol $Pro$ has a secrecy attack. $Pro$ is a matching RO protocol. The well-known definition of a deterministic 2-counter machine is presented in Appendix IV. The attack is a run of $Pro$ where the attacker, who is an insider, finally knows some secret term. The secret term is declared by the same way $N_b$ is declared as the secret term for the well-known attack to the PKNS protocol [4]. Since the 2-directional proof can be carried out in a straightforward style, we focus on presenting the "parameters" of the proof template, which are the encoding schemes and protocol design. The remaining technical details are included in Appendix VI.

***Theorem** 1:* Secrecy checking of matching RO protocols is undecidable while considering an insider attacker, and runs of protocols with bounded message size.

*Proof:* Given a 2-counter machine $M = (Q, \delta)$, where $Q$ is a set of states and $\delta$ is a set of transition rules. let $Q = \{q_0,\ q_{final},\ q_1,\ q_2,\ \cdots,\ q_m\}$ and $\delta = \{T_1,\ T_2,\ \cdots,\ T_n\}$. The proof follows the template presented in Section II.

Encoding is a correspondence between a term and its designed meaning in a reduction. In the reductions of undecidability [7] [8] [11] and NP-hardness [17] [12] encoding is implemented by encryptions using some symmetric key shared by agents. Using shared symmetric key could make the proofs easier and we could also design the encoding scheme using symmetric keys. However we implement encoding in the reduction using only public keys and private keys, for three reasons . First, it is an interesting challenge since we have not noticed a reduction proof using only initially distributed public keys and private keys. Second, the result could be more practical noticing that in the well-known attack to the PKNS protocol, where the attacker is an insider, there is no shared symmetric key between two agents. Third, the attack is more convincing. For a term encrypted by a symmetric key unknown to $I$, $I$ cannot construct it nor decrypt it, which leaves a question of how can the attacker $I$ know its actions in the attack while so many terms are blind to $I$. But when a term is encrypted by an asymmetric key unknown to $I$, while $I$ knows the decryption key, $I$ still cannot construct the term, but $I$ can decrypt it and understand the message, and so has a clear view of the (more convincing) attack.

A ***configuration term*** has the form $\{5, B, e, q, C_1, C_2\}_{\overrightarrow{k_A^0}}$, where $A$ and $B$ are two different regular agent names, which means $\{A, B\} \subset AN$ ($AN$ is the set of insider names specified in a run), $A \neq I$, $B \neq I$, $A \neq B$. $C_1$ and $C_2$ could any terms. $e$ is a special constant used in the reduction as an evidence to distinguish a real configuration term from a term of the form $\{5, B, N_A, q, C_1, C_2\}_{\overrightarrow{k_A^0}}$, where $N_A$ is a nonce generated by $A$ in the first message of the role $S_f$ (introduced later), $1 \leq f \leq n$. An honestly generated nonce $N_A$ cannot be $e$, due to the assumption of unbounded fresh nonce generation. $q$ is a constant that could be any state in $Q$ (The set of states of the two counter machine $M$). The regular agent name $B$ included in a configuration term is also designed to prevent the $I$ from getting a (final) configuration term trivially. Note that $I$ can easily get a term $\{5, I, e, q_{final}, C_1, C_2\}_{\overrightarrow{k^0_A}}$, which is not a configuration term ($I$ is inside, not $B$), when $A$ plays the role $S_{final}$ (introduced later) and $A$ talks with $I$. Note that $I$ cannot construct a configuration term since $I$ does not know $k^0_A$. The Design of using $B$ and $e$ for a configuration term is to ensure that $I$ cannot get a configuration term trivially, except the starting configuration term $\{5, B, e, q_0, z, z\}_{\overrightarrow{k^0_A}}$ corresponding to $(q_0, 0, 0)$, and $I$ have to do a run to simulate a computation of $M$ in order to get non-trivial configuration terms which include the final configuration terms.

A ***connection term*** has the form of $\{7, B, C_1, C_2\}_{\overrightarrow{k^0_A}}$, where $\{A, B\} \subset AN$, $A \neq B$, $A \neq I$, $B \neq I$. Connection terms are used to build the encoding of a counter value described later. The encryptions containing constant 5 are distinguished from, and cannot be unified with, the encryptions containing constant 7. A ***secret term*** is a term $N_B$ generated by a role instance of $R_{final}$ (introduced later) which is executed by a regular agent $B$, $B \neq I$, where $B$ tries to send $N_B$ to agent $A$, $A \neq I$. The secret term is declared in the same way as in the public key Needham-Schroeder protocol [4].

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

$$\forall P, P \in AN : P.init = AN \cup \{k^1_B | B \in AN\} \cup Q \cup \{e, z, 1, 2, 3, 5, 7, k^0_P\}$$

The set of roles of $Pro$ is the follows.

$$\{S_0, R_0\} \cup \{\ S_1, R_1, \cdots, S_n, R_n\} \cup \{S_{copy1}, R_{copy1}, S_{copy2}, R_{copy2}, S_{final}, R_{final}\}$$

Note that there are $n$ transition rules in $\delta$. $AN$ is the names of the set of insiders in a run.

For each $run$ of $Pro$, we consider $I$ is an insider, which means that $I \in run.AN$ and that $I.init$ has the same pattern as other agents, described above, just instantiate $P$ with $I$.

We describe a matching pair of roles $S_i$ and $R_i$, $i \in \{0, 1, \cdots n, final\}$ by the CS between them. The actions of two matching roles can be straightforwardly parsed from the CS between them. We choose different variable names in different matching pairs of roles so there is no confusion when the CSs of these pairs of roles are concatenated to form the CS of the whole protocol. The CS of the whole protocol can be chosen as a sequence starting with the CS between $S_0$ and $R_0$, followed by the one between $S_1$ and $R_1$, and then continues until the one between $S_n$ and $R_n$ is appended, and then the one between $S_{final}$ and $R_{final}$ is appended. Actually any interleaving of actions steps of the pairs of roles will be a corresponding CS of the protocol.

Two roles $S_0$ and $R_0$ are designed to generate the initial configuration term. $S_0$ and $R_0$ are executed by $A_0$ and $B_0$ respectively. The CS between them is the following.

1. $+(A_0 \Rightarrow B_0) : \quad A_0, B_0, \{5, B_0, e, q_0, z, z\}_{\overrightarrow{k^0_{A_0}}}$

Call the message appearing in the above CS between $S_0$ and $R_0$ as $Msg$. Then $S_0$ contains one action of sending $Msg$ and $R_0$ contains one action of receiving $Msg$.

The two roles $S_{copy1}$ and $R_{copy1}$ are used to rewrite a configuration term, they are executed by agents $P1$ and $G1$ respectively. The CS (containing only message sending actions) between them is the follows.

1. $\#_{P1}(N1_{P1}, N2_{P1}, N3_{P1})$
   $+(P1 \Rightarrow G1) : P1, G1, \{5, G1, N1_{P1}, N2_{P1}, N3_{P1}\}_{\overrightarrow{k^0_{P1}}}$
2. $+(G1 \Rightarrow P1) : G1, P1, \{5, P1, N1_{P1}, N2_{P1}, N3_{P1}\}_{\overrightarrow{k^0_{G1}}}$

The two roles $S_{copy2}$ and $R_{copy2}$ are used to rewrite a connection term, they are executed by agents $P2$ and $G2$ respectively. The CS between them is the follows.

1. $\#_{P2}(N1_{P2}, N2_{P2})$
   $+(P2 \Rightarrow G2) : P2, G2, \{7, G2, N1_{P2}, N2_{P2}\}_{\overrightarrow{k^0_{P2}}}$
2. $+(G2 \Rightarrow P2) : G2, P2, \{7, P2, N1_{P2}, N2_{P2}\}_{\overrightarrow{k^0_{G2}}}$

The two roles $S_{final}$ and $R_{final}$ carry out an adjusted version of the public key Needham-Schroeder protocol, they are executed by agents $A$ and $B$ respectively. The communication sequence between $A$ and $B$ is the follows.

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

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

1. $\#_{A_f}(C1_f, C2_f, C1_f^{-1}, C2_f^{-1}, N_f) \quad + (A_f \Rightarrow B_f):$
   $A_f, B_f, \{5, B_f, N_f, q, C1_f, C2_f\}_{k^0_{A_f}}^{\rightarrow},\ \{7, B_f, C1_f^{-1}, C1_f\}_{k^0_{A_f}}^{\rightarrow}, \{7, B_f, C2_f^{-1}, C2_f\}_{k^0_{A_f}}^{\rightarrow}$
2. $\#_{B_f}(C1_f^{+1}, C2_f^{+1}) \quad + (B_f \Rightarrow A_f):$
   $B_f, A_f, \{5, A_f, N_f, q', C1'_f, C2'_f\}_{k^0_{B_f}}^{\rightarrow},\ \{7, A_f, C1_f, C1_f^{+1}\}_{k^0_{B_f}}^{\rightarrow},\ \{7, A_f, C2_f, C2_f^{+1}\}_{k^0_{B_f}}^{\rightarrow}$

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

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

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

The counter value 0 must be represented by $z$. When a counter $h$ should be positive, the term $\{7, B_f, Ch_f^{-1}, Ch_f\}_{k^0_{A_f}}^{\rightarrow}$ is needed in $Msg_1$, which shows that $Ch_f^{-1}$ encodes a number one less than the number encoded by $Ch_f$. Verbose explanation of the rewrite rules are presented in Appendix V.

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

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

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

A special representation $\{7, A/B, z, X\}_{k^0_{A/B}}^{\rightarrow}$ represents either the term $\{7, A, z, X\}_{k^0_B}^{\rightarrow}$ or $\{7, B, z, X\}_{k_{A^0}}^{\rightarrow}$, $A \neq B$. Similarly $\{5, A/B, e, X, Y\}_{k^0_{A/B}}^{\rightarrow}$ can represent either $\{5, A, e, X, Y\}_{k^0_B}^{\rightarrow}$ or $\{5, B, e, X, Y\}_{k^0_A}^{\rightarrow}$.

After running a set $E$ of events in a $run$ of $Pro$, we say a term $X$ is the ***encoding*** of a positive integer $N$, if and only if there is a sequence of terms :

$$\{7, A/B, z, X_1\}_{k^0_{A/B}}^{\rightarrow}, \{7, A/B, X_1, X_2\}_{k^0_{A/B}}^{\rightarrow}, \cdots, \{7, A/B, X_{N-2}, X_{N-1}\}_{k^0_{A/B}}^{\rightarrow}, \{7, A/B, X_{N-1}, X\}_{k^0_{A/B}}^{\rightarrow}$$

such that ($A$ and $B$ are two different regular agent names) $\{A, B\} \subset run.AN$, $A \neq B$, $A \neq I$, $B \neq I$, and the attacker knows each element $T$ of this sequence ($T \in know_I(E)$). Note that in this sequence the connection terms can have differnt encryption keys, some are encrypted by $k^0_A$ and $B$ appears in the encrypted text, while some are encrypted by $k^0_B$ and $A$ appears in the encrypted text. Here $X$ and $X_j$, for some integer $j$, $1 \leq j \leq N-1$, are different variables that can represent any terms (could be composite terms). We call $N$ the ***i_value*** of $X$ (i stands for integer), or $X$ is the ***encoding*** of $N$, or $X$ ***encodes*** $N$, denoted as $N = \underline{X}$. The above term sequence is called the ***encoding sequence*** of $X$. The encoding sequence of $z$ is $z$.

Since all the agents (including $I$) are symmetric, i.e., they have the same intial knowledge pattern, it is obvious that if there is a run where a secret $n_d$ is leaked which is generated by $d$ for $c$, then there is a run where a secret $n_b$ is leaked where $n_b$ is generated by $b$ for $a$. Without loss of generalization, we consider only the secret term $n_b$ between two agents $b$ and $a$. For this reason, the rest of the proof only considers configuration terms of the form $\{5, a/b, e, X, Y\}_{k^0_{a/b}}^{\rightarrow}$ and connection terms of the form $\{7, a/b, X, Y\}_{k^0_{a/b}}^{\rightarrow}$.

The encoding of 0 is the constant $z$. So $0 = \underline{z}$. The encodings of numbers are connected in a connection term to show the consecutive order between numbers. For example $\{7, B, X, Y\}_{k^0_A}^{\rightarrow}$ in an encoding sequence means that $\underline{X} = \underline{Y} - 1$.

***Direction 1***: Suppose there is a computation of $M$ call it $Comp$, and $Comp$ has $u$ transition steps, $1 \leq u$, such that after $Comp$, $M$ can reach a final configuration $(q_{final}, \_, \_)$ from the initial configuration, we prove that there exists a $run$, such that $run \in Runs^{D:Pro}$ where $D$ is the initial knowledge patter of the insider attacker described earlier in this proof, and some secret term is included in $know_I(run.E)$.

We prove this direction by constructing a $run$ simulating $Comp$. $Pro$ is the protocol just described. $run.AN = [I, a, b]$. Only three agents are enough here to instantiate the sender and receiver variables in each role instance. $run$ starts with a role instance of $S_0$ executed by $a$, to generate the initial configuration term $\{5, b, e, q_0, z, z\}_{\overrightarrow{k_a^0}}$. Then for each transition step of $Comp$ of applying a transition rule $t$, a role instance, say $r$, of the corresponding role $R_f$ which is translated from $t$, is executed. $r$ receives a configuration term encoding the previous configuration, and produce a configuration term encoding the next configuration of $Comp$. In addition, some role instances of $R_{copy1}$ and $R_{copy2}$ are executed to swap the positions of $a$ and $b$ in the newly generated configuration term and number connection terms, in order to maintain the encoding scheme. A lemma is proved by induction on each step of $Comp$ to show that the attacker can always obtain the message needed for the next step of the attack. Finally, when the final configuration term is generated, the attacker use it to carry out an attack between $S_{final}$ and $R_{final}$, which is the same as the well-known attack to the PKNS protocol. When we show the role instances to the constructed run we do not need to specify the $time$ associated with each event, since the role instances, and their events, are executed consecutively and no two events need to be executed at the same time, therefore $time$ fields can be assigned like consecutively 1, 2, and so on according to the order that the event is introduced in the run. Details are presented in Appendix VI-A.

***Direction 2***: We have to show that for any $run$, $run \in Runs^{D:Pro}$, if there is a secret term $n_b$ such that $n_b \in know_I(run.E)$, then the 2-counter machine $M$ can reach a final configuration $(q_{final}, \_, \_)$.

The proof design is the follows. The only way the attacker can know $n_b$ is by a Needham-Schroeder attack showed in the finishing events in Direction 1. The only way for $I$ to execute such a Needham-Schroder attack is to know a term $\{5, b, e, q_{final}, C1, C2\}_{\overrightarrow{k_a^0}}$. And the only way to know this term is to carry out a run of $Pro$ which simulates a computation of $M$ reaching a final configuration.

We can observe that in a run every term can encode at most one number, and 0 can only be encoded by $z$. We can use stronger lemma to show that every configuration term generated in a run of $Pro$ encodes a configuration reachable to $M$. And it is obvious that the secret term is leaked if and only if the attacker can obtain a final configuration term. Details of the proof of Direction 2 are included in Appendix VI-B. ■

## *A. Undecidability of checking authentication*

Formal definition of authentication goals for security protocols have been discussed by by several papers. The most clarified and widely used definitions of authentiction goals are presented by Lowe in [26] as the correspondence between role instances with shared data. An example of the authentication goal of the PKNS protocol is discussed in [26] as follows.

For PKNS protocol, the role of A can be represented by a schema of four parameters: $Role_A(A, B, N_A, N_B)$. Similarly, role of B can be represented as $Role_B(A, B, N_A, N_B)$. According to the protocol as a communication sequence, some atomic terms should appear in both roles, this atomic terms are considered being shared by both roles. In PKNS, $Role_A$ and $Role_B$ share all of the parameters. In every role instance of $Role_A$ or $Role_B$, the parameters are instantiated by some specific values. According to the PKNS protocol, $Role_A$ and $Role_B$ share all the parameters. Recentness is another requirement to define a strict authentication goal, which means for a role instance $r$ finished in a run there can only be a unique corresponding role instance $r'$ appeared in the run. $r'$ is only required to finish a prefix up to the last message that is supposed to be sent from $r'$ and received by $r$. Note that for a goal of $Role_1$ to authenticate $Role_2$, both the agents who execute $Role_1$ and $Role_2$ should be a regular agent, otherwise the goal is not defined, since the behavior of the attacker cannot be organized into role instances.

***Definition*** *9:* Given a protocol $Pro$ as a communication sequence, and two roles $Role_1$ and $Role_2$ parsed from $Pro$. Suppose $S$ is a set of parameters shared by $Role_1$ and $Role_2$ according to $Pro$. Assume $S$ always include the two agent names who execute $Role_1$ and $Role_2$. Let $M$ be the last message that is sent by $Role_2$ and received by $Role_1$ according to $Pro$. The authentication goal of $Role_1$ to authenticate $Role_2$, denoted as $Role_1 \rightarrow Role_2$, means that for a role instance $r$ of $Role_1$ in a run, there is one and only one role instance $r'$ of $Role_2$ in the run which has been executed up to $M$, and each variable in $S$ is instantiated by the same value in $r$ as in $r'$.

***Theorem** 2:* Authentication checking of matching RO protocols is undecidable while considering an insider attacker, and runs of protocols with bounded message size.

Same as the attack to PKNS, the leaked secret term triggers an authentication attack, which is the proof idea. Further details are provided in Appendix VI-C.

We did not notice any existing proof of undecidability of authentication. An authentication goal can only be defined based on a CS. Since the existing proofs undecidability of secrecy are based on (non-matching) roles, not on a CS, they cannot be adapted to show the undecidability of authentication, which could explain why the proof is missing.

## V. Summary

In order to prove the complexity results of checking security protocols, the protocol and security goals of secrecy and authentication in the proofs should be described exactly as in the realistic cases. We present a thorough analysis of the problems of checking secrecy and authentication for security protocols and prove the undecidability which is first to be directly applicable to protocols common in literature (matching RO protocols) with an insider attacker. We directly address the secrecy and authentication goals of the PKNS protocol. The proof of authentication is the first to our best knowledge. Our approach is powerful enough to solve the two open problems posted in [1]. Our approach has the advantage of using a clarified modeling and very flexible reduction techniques.

The proving approach is general and can be reused to deal with problems beyond the theorems of this paper. We have adapted this approach to prove the undecidability of a complex problem of secrecy checking [15], and a special kind of replay attack (thanks to our undecidability proof of authentication).

Our vision is that an undecidability proof for checking cryptographic protocols can be built with only a small amount of human input. Although the remaining reasoning of the proof needs a lot of details, it could be carried out as routines by some mechanical reasoning tools. The techniques of artificial intelligence, such as case-based reasoning, could be used to provide customized proofs based on a common general proof template with further convenience.

## References

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

[2] Z. Liang and R. M. Verma, “Improving Techniques for Proving Undecidability of Checking Cryptograhpic Protocols,” in *Workshop on Privacy and Security by means of Artificial Intelligence (PSAI)*, March 2008.

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

[4] G. Lowe, “Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR,” in *TACAS*, 1996, pp. 147–166.

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

[6] 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.

[7] H. Comon and V. Cortier, “Tree automata with one memory, set constraints and cryptographic protocols,” Laboratoire Spécification et Vérification, ENS Cachan, France, http://www.lsv.ens-cachan.fr/Publis/RAPPORTS˙LSV/PS/rr-lsv-2001-13.rr.ps, Tech. Rep. LSV-01-13, December 2001, 98 pages. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/RAPPORTS˙LSV/PS/rr-lsv-2001-13.rr.ps

[8] 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.

[9] N. A. Durgin, “Logical Analysis and Complexity of Security Protocols,” Ph.D. dissertation, Computer Science Department, Stanford University, March 2003. [Online]. Available: http://www-cs-students.stanford.edu/˜nad/papers/thesis.ps

[10] S. Even and O. Goldreich, “On the security of multi-party ping-pong protocols,” in *IEEE Symposium on Foundations of Computer Science*, 1983, pp. 34–39.

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

[12] 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.

[13] J. Clark and J. Jacob, “A survey of authentication protocol literature: Version 1.0,” Department of Computer Science, University of York, UK, Tech. Rep., 1997. [Online]. Available: www.cs.york.ac.uk/˜jac/papers/drareviewps.ps

[14] D. Gollmann, “Insider fraud (position paper).” in *Security Protocols Workshop*, ser. Lecture Notes in Computer Science, B. Christianson, B. Crispo, W. S. Harbison, and M. Roe, Eds., vol. 1550. Springer, 1998, pp. 213–219.

[15] Z. Liang and R. M. Verma, “Secrecy Checking of Protocols: Solution of an Open Problem,” Computer Science Department, University of Houston, Texas, USA, http://www.cs.uh.edu/preprint, Tech. Rep., April 2007, uH-CS-07-04.

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

[17] 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.

[18] 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.

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

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

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

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

[23] P. Syverson, C. Meadows, and I. Cervesato, “Dolev-Yao is no better than Machiavelli,” in *Proceedings of the First Workshop on Issues in the Theory of Security*, P. Degano, Ed., Geneva, Switzerland, 2000, pp. 87–92. [Online]. Available: http://theory.stanford.edu/˜iliano/papers/wits00.ps.gz

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

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

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

[27] J. D. Guttman, F. J. Thayer, and J. C. Herzog, “Strand spaces: Why is a security protocol correct?” in *1998 IEEE Symposium on Security and Privacy*, May 1998, pp. 160–171. [Online]. Available: “url–http://www.mitre.org/work/tech˙papers/tech˙papers˙00/guttman˙stran%ds/index.html˝

## Appendix I
## The communication sequence and roles of the PKNS protocol

The core of the public key Needham-Schroeder protocol [5] (PKNS) is described as the communication sequence (CS) of three steps listed in Fig. 1, together with the two roles $Role_A$ and $Role_B$ that can be parsed from the CS.

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

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

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

## Appendix II
## Insiders vs. Outsider

The usually meanings of insiders and outsiders are the follows. 1) There is a common organization which defines the background group for the insiders. 2) Insiders can recognize each other or at least knows the identities or names of each other. So normally an insider will try to start a conversation for the internal business of the organization only to another insider, not to anybody in the world. 3) Insiders have some privilege that the outsiders do not have. The scenario of the checking security protocols discussed in this paper is exactly the same. 1) The agents who participated the initially knowledge establishing stage implies a group of legitimate agents. 2) The names of legitimate agents are known to each other. When $A$ start a communication with $B$, $B$ must be known to $A$ and $B$ should also be an insider. 3) In the published proofs, regular agents knows some special terms, say a key $K$, where other agents do not know, including the attacker. So using the terms “insider” and “outsider” for the agents in a security protocol run is quite appropriate.

## Appendix III
## Why the two problems are open

Existing undecidability results cannot cover the two open problems posted in [1]. The complexity of checking secrecy and authentication for realistic protocols with an insider attacker is unknown, due to the following two reasons. 1. In general it is not clear whether the problem of checking security always becomes easier or harder assuming a more powerful attacker than assuming an attacker with less privileges. When the attacker has more choices to do attacks, the situation could be more complex to check whether the system will fail, but sometimes it can also be obvious to see that the strong attacker can easily penetrate the protection. Proving the undecidability of checking secrecy assuming an outsider attacker does not imply the same undecidability assuming an insider attacker. 2. we can see that the set of protocols represented as any set of roles (non-matching or matching) covers two disjoint sets of protocols, the protocols as non-matching set of roles and the protocols as matching roles. While the undecidability result for protocols as non-matching roles implies the undecidability for protocols as any set of roles, it does not imply the undecidability for protocols as matching roles, the realistic ones.

## Appendix IV
## Deterministic 2-counter machine

***Definition** 10:* A ***deterministic 2-counter machine*** [25] with empty input is a pair $(Q, \delta)$, where $Q$ is a set of states including the starting state $q_0$ and the accepting state $q_{final}$ and $\delta$ is a set of transition rules. A configuration of a 2-counter machine is a tuple $(q, C_1, C_2)$, where $q$ is the current state and $C_1$ and $C_2$ are two non-negative integers representing the two counters. The 2-counter machine can detect whether a counter is 0 or not. A transition rule, (call the rule $T \in \delta$) is of the form $[q, i_1, i_2] \to [q', j_1, j_2]$, where $q, q' \in Q$; $i_1, i_2 \in \{0, 1\}$; $j_1, j_2 \in \{-1, 0, +1\}$. An application of $T$ can be described as $(q, C_1, C_2) \longrightarrow^T (q', C_1', C_2')$, where LHS and RHS are the configuration before and after the transition respectively. For $h \in 1, 2$, when $i_h = 0$, it means that $C_h = 0$. When $i_h = 1$, it means that $C_h > 0$. When $j_h = +1$ ($j_h = 0$, $j_h = -1$), it means that after the transition, $C_h' = C_h + 1$ ($C_h' = C_h$, $C_h' = C_h - 1$). Especially, when $j_h = -1$, $i_h$ must be 1, since decrementing 0 is not allowed. The reachability problem of such a 2-counter machine is to decide that, starting from the initial configuration $(q_0, 0, 0)$, after applying some applicable transition rules, whether some final configuration $(q_{final}, \_, \_)$ can be reached, where $\_$ represents an arbitrary possible value. We assume (for convenience) that $q_0 \neq q_{final}$ and, for nontriviality, that $\delta$ is not empty.

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

## Appendix V
## More Explanation of the Rewrite Rules for Theorem 1

Here is some explanation of the rewrite rules presented in the proof of Theorem 1 arranged by the rule numbers.

1) Counter value 0 must be represented by $z$. There is no previous counter value for 0. So the term $\{7, B_f, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k^0_{A_f}}}$, which shows that $Ch_f^{-1}$ encodes a number one less than the number $Ch_f$ ($Ch_f = z$) encodes, should not be required.
2) When a counter is positive, we emphasize that there must be evidence showing the counter has a preceding nonnegative value. This rule is redundant since it is covered by the default rule.
3) When a counter is incremented, the variable $Ch_f'$ is replaced by a nonce $Ch_f^{+1}$ generated by $B_f$ . The history records showing that the new counter value is incremented from its precedent is represented by the connection term $\{7, A_f, Ch_f, Ch_f^{+1}\}_{\overrightarrow{k^0_{B_f}}}$.
4) When a counter is kept the same in the transition, neither the new nonce nor the record of counter value increment is needed.
5) When a counter is decremented, the variable $Ch_f'$ is replaced by the preceding counter representation $Ch_f^{-1}$. The new nonce which should represent an incremented counter and the record of the counter increment are not needed. Note that when $j_h = -1$, $i_h$ must be 1 for a valid transition rule $T_f$, and rule 2 applies.

## APPENDIX VI
## DETAILED PROOFS

### A. Proving Direction 1 for Theorem 1

We focus on describing $run.E$, while the set of role instances $run.R$ will be clear once $run.E$ has been described. $run.E$ is described below.

We need to prove that the constructed $run$ is a run, or $run \in Runs^{D:Pro}$. More specificly, we have to show that the attacker can construct every message before it is received by a regular agent.

$run.E$ can be divided into three parts: the starting actions, and the transition actions, and the finishing actions. A secret term is obtained by the attacker at the end of the ending actions. We build $run.E$ by appending actions to $run.E$, starting from an empty sequence.

The ***starting actions***. $run.E$ starts with the events of a role instance of $S_0$, call it $s^0$, which is executed by $a$. So $s^0 \in run.R.$ , $s^0.acts$ includes one event

$$+(a \Rightarrow b) : a, b, \{5, b, e, q_0, z, z\}_{\overrightarrow{k^0_a}} \quad ,$$

and this event is appended in $run.E$.

The ***transition actions***: Suppose the $w^{th}$ step in $Comp$ is $(q, V_1, V_2) \longrightarrow^t (q', V_1', V_2')$, where $1 \leq w \leq u$ and $t \in \delta$. By the construction of $Pro$, the transition rule $t$ corresponds to a pair of transition roles in the protocol, say $S_f$ and $R_f$, where $1 \leq f \leq n$. The $w^{th}$ step in $Comp$ corresponds to one role instance of $R_f$, call it $r^w$, one role instance of $R_{copy1}$, call it $p^w$, and two role instances of $R_{copy2}$, call them $p1^w$ and $p2^w$. So $\{r^w, p^w, p1^w, p2^w\} \subset run.R$, and they are executed by $b$, $a$, $a$, and $a$ respectively. The events of the four role instances, totally (at most) eight events, are appended to $run.E$. According to the protocol, these events can be described in the general template as below, where the exact form of these events depends on the specific role $R_f$. The events of a role instance $r$ is denoted as $r.acts$.

$$
\begin{array}{lll}
r^w.acts = & & \\
 & & -(a \Rightarrow b) : a, b, \{5, b, e, q, C1_f, C2_f\}_{\overrightarrow{k^0_a}}, \\
 & & \{7, b, C1_f^{-1}, C1_f\}_{\overrightarrow{k^0_a}}, \{7, b, C2_f^{-1}, C2_f\}_{\overrightarrow{k^0_a}} \\
\#_b(C1_f^{+1}, C2_f^{+1}) & & +(b \Rightarrow a) : b, a, \{5, a, e, q', C1'_f, C2'_f\}_{\overrightarrow{k^0_b}}, \\
 & & \{7, a, C1_f, C1_f^{+1}\}_{\overrightarrow{k^0_b}}, \{7, a, C2_f, C2_f^{+1}\}_{\overrightarrow{k^0_b}} \\
p^w.acts = & -(b \Rightarrow a) : & b, a, \{5, a, e, q', C1'_f, C2'_f\}_{\overrightarrow{k^0_b}} \\
 & +(a \Rightarrow b) : & a, b, \{5, b, e, q', C1'_f, C2'_f\}_{\overrightarrow{k^0_a}} \\
p1^w.acts = & -(b \Rightarrow a) : & b, a, \{7, a, C1_f, C1_f^{+1}\}_{\overrightarrow{k^0_b}} \\
 & +(a \Rightarrow b) : & a, b, \{7, b, C1_f, C1_f^{+1}\}_{\overrightarrow{k^0_a}} \\
p2^w.acts = & -(b \Rightarrow a) : & b, a, \{7, a, C2_f, C2_f^{+1}\}_{\overrightarrow{k^0_b}} \\
 & +(a \Rightarrow b) : & a, b, \{7, b, C2_f, C2_f^{+1}\}_{\overrightarrow{k^0_a}}
\end{array}
$$

Note that whether we need $ph^w$, $h \in \{1, 2\}$, depends on whether $\{7, A_f, Ch_f, Ch_f^{+1}\}_{\overrightarrow{k^0_{B_f}}}$ appears in the second message of $R_f$ or not. So we may need 4, 6, or 8 actions steps when we need neither of, one of, or both of, $p1^w$ and $p2^w$ respectively.

We need to specify the instantiation of the variables in the above template of steps. Note that the attacker impersonates $a$ , constructs and sends the first message received by $b$ in $r^w$. The nonce variable $N_f$ in $R_f$, which should normally be instantiated by a fresh nonce, is instantiated by the constant $e$ in $r^w$. Intuitively we want the configuration term $\{5, b, e, q, C1_f, C2_f\}_{\overrightarrow{k^0_a}}$ encodes the configuration $(q, V_1, V_2)$ of $M$. Let $E_w$ be the prefix of $E$ which ends immediately before the first event of $r^w$. We require that the instantiation of $Ch_f$ must encode $V_h$, denoted as $V_h = \overline{Ch_f}$, for $h \in \{1, 2\}$, after running $E_w$. Whether $\{7, b, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k^0_a}}$ appears in the first message of $r^w.acts$, $h \in \overline{\{1, 2\}}$ depends on the specific $R_f$. If $Ch_f^{-1}$ appears in $R_f$, we require the instantiation of $Ch_f^{-1}$ satisfys that that $V_h - 1 = \overline{Ch_f^{-1}}$. No need to specify the values of the two variable $C1'_f$ and $C2'_f$, since they will be replaced by other variables according to $R_f$. The variable $Ch_f^{+1}$, $h = 1or2$, if it will appear in $R_f$, is instantiated by a nonce freshly generated by $b$ in the second event of $r^w$.

Now we need to show that every message received by $b$ or $a$ in the above sequence can be obtained by the attacker before it is received. We only need to show that for the first message, call it $Msg$, received by $b$ in $r^w$ can be obtained by $I$, denoted as $Msg \in know_I(E_w)$, which can be proven by the following stronger lemma.

***Lemma 1:*** In the constructed run, for the role instance $r^w$, the $w^{th}$ transition step of $M$, and $E_w$ just described, the following three facts are true.

1) There exists $\{5, b, e, q, C_1, C_2\}_{\overrightarrow{k_a^0}} \in know_I(E_w)$, such that $V_1 = \underline{C_1}$ and $V_2 = \underline{C_2}$.
2) For $h \in \{1, 2\}$, if $V_h > 0$, then $\{7, b, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k_a^0}} \in know_I(E_w)$, such that after running $E_w$, $V_h - 1 = \overline{Ch_f^{-1}}$.
3) After running the events listed earlier, we call the set of of executed events in the run so far as $E'_w$. After $\overline{E'_w}$, $V'_h = \underline{C'_h}$, for $h \in \{1, 2\}$, and $\{5, b, e, q', C'_1, C'_2\}_{\overrightarrow{k_a^0}} \in know_I(E'_w)$.

*Proof:* This lemma can be proven by induction on the length of the computation steps of $M$. When we say a term $T$ encodes a number $N$, we can find at least one encoding sequence of $T$ formed by $N$ connection terms appeared in the run. A CS of $T$ may have have connection terms where $a$ and $b$ are in different positions, some like $\{7, b, X, Y\}_{\overrightarrow{k_a^0}}$, and some like $\{7, a, X, Y\}_{\overrightarrow{k_b^0}}$. To prove this lemma we only need to consider a CS where connection terms are in the form $\{7, b, X, Y\}_{\overrightarrow{k_a^0}}$, thanks to the role $R_{copy2}$.

***Base case***: The first transition step of $Comp$ ($w = 1$) must have the form $(q_0, 0, 0) \longrightarrow^t (q', V'_1, V'_2)$, for some $q' \in Q$, and $V'_1, V'_2 \in \{0, 1\}$. Suppose $t$ is translated into $S_f$ and $R_f$ of the protocol, for some $f$, $1 \le f \le n$, and $r^w$ is a role instance of $R_f$. Proving the first fact: Obviously $\{5, b, q_0,\ z,\ z\}_{\overrightarrow{k_a^0}} \in know_I(E_1)$ since $\{5, b, q_0, z, z\}_{\overrightarrow{k_a^0}}$ is just produced by the starting event. Since $0 = \underline{z}$, the first fact is proved. Proving the second fact: Since $V_1 = V_2 = 0$, the second fact is trivially true. Proving the third fact: In the first transition step of $Comp$, for $h \in \{1, 2\}$, either $V'_h = V_h + 1 = 1$ or $V'_h = V_h = 0$. If $V'_h = 1$, then according to the design of $Pro$ and the run of $Pro$ in direction 1, the second message of $r^1$ which is a role instance of $R_f$ executed by $b$ must include a term $\{7, a, z, Ch_f^{+1}\}_{\overrightarrow{k_b^0}}$. Then in the run this term is sent by $I$ to the role instance $ph^1$ which is a role instance of $R_{copy2}$ executed by $a$, and $a$ will send $\{7, b, z, Ch_f^{+1}\}_{\overrightarrow{k_a^0}}$ in the second event of $ph^1$, and this single connection term forms an encoding sequence of $Ch_f^{+1}$. So after running $E'_1$, $V'_h = 1 = \underline{Ch_f^{+1}} = \underline{Ch'_f}$. If $V'_h = 0$, then $V'_h = 0 = \underline{Ch'_f}$, where $Ch'_f = Ch_f = z$. So it must be true the $V'_h = \underline{Ch'_f}$. A term of the form $\{5, a, e, q', C1'_f, C2'_f\}_{\overrightarrow{k_b^0}}$ is generated in the second event of $r^1$, while $e$ is carried along in $r^1$. This term is sent to the role instance $p1^1$ executed by $a$, and then $a$ will send in the second event of $p1^1$ the term $\{5, b, e, q', C1'_f, C2'_f\}_{\overrightarrow{k_a^0}}$. Also by the design of $Pro$, $q'$ and $q$ in $R_f$ are always the same as the $q'$ and $q$ in $t$. The third fact is proved. The base case is proven.

***Induction step***: Suppose for the $w - 1^{th}$ step, the lemma is true, while for a stronger consideration we only consider the CS of a term where every connection term is encrypted by $k_a^0$. We need to prove that the lemma is also true for the $w^{th}$ transition step.

Fact 1. The $w^{th}$ transition step of $Comp$ is of the form $(q, V_1, V_2) \longrightarrow^t (q', V'_1, V'_2)$. The configuration $(q, V_1, V_2)$ must be reached by the $w - 1^{th}$ step. By the induction hypothesis, for the $w - 1^{th}$ step, which just occurred, the lemma is satisfied, so there must be a term $\{5, b, e, q, X_1, X_2\}_{\overrightarrow{k_a^0}}$ already generated and collected into $know_I(E'_{w-1})$, where $V_1 = \underline{X_1}$ and $V_2 = \underline{X_2}$. Since $E'_{w-1}$ is the same as $E_w$, The first fact is proven.

Fact 2. If $V_h > 0$, then according to the design of $Pro$, the instance of $\{7, b, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k_a^0}}$ will appear in the first message of $r^w$, which is a role instantce of $R_f$. By Fact 1 above, $Vh_f = \underline{Ch_f}$ after running $E_w$. By the induction hypothesis, there must be a encoding sequence of $Ch_f$, where every connection term is encrypted by $k_a^0$, and the last term of this sequence has the form $\{7, b, X, Ch_f\}_{\overrightarrow{k_a^0}}$, and $\underline{X} = Vh_f - 1$. The instance $\{7, b, X, Ch_f\}_{\overrightarrow{k_a^0}}$ must be included in $know_I(E_w)$, and is used to instantiate $\{7, b, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k_a^0}}$. The second fact is proven. We can see that given a specific $Ch_f$, the term $\{7, b, X, Ch_f\}_{\overrightarrow{k_a^0}}$ is unique is a run, by Observation 3 in the proof of Direction 2.

Fact 3. We need to show that $V'_h = \underline{Ch'_f}$ after executing $E'_w$. There are three possible cases for the value of $j_h$ in $t$, for $h \in \{1, 2\}$.

- $j_h = -1$ and $V'_h = V_h - 1$. According to the design of $R_f$, $Ch'_f = Ch_f^{-1}$. By the proven fact 2, $V'_h = V_h - 1 = \underline{C_h^{-1}} = \underline{C'_h}$.
- $j_h = 0$ and $V'_h = V_h$. According to the design of $Pro$, $Ch'_f = Ch_f$. By the proven fact 1, $V'_h = V_h = \underline{Ch_f} = \underline{Ch'_f}$.
- $j_h = +1$ and $V'_h = V_h + 1$. Then according to the design of $R_f$, $Ch'_f = Ch_f^{+1}]$. According to the design of the protocol and the run, in the second message of $r^w$, there must be a term $\{7, a, Ch_f, Ch_f^{+1}]\}_{\overrightarrow{k_b^0}}$ generated in the second message of $r^w$, where $C_h^{+1}$ is a fresh nonce generated by the b. Then this term is sent to role instance $ph^w$ executed by $a$ and in the second event of $ph^w$ a term $\{7, b, Ch_f, Ch_f^{+1}\}_{\overrightarrow{k_a^0}}$ is generated and

is obviously collected in $know_I(E'_W)$. By the proven fact 1, $V_h = \underline{C_h}$. Then by the definition of encoding, after running $E'_w$, $V'_h = V_h + 1 = \underline{Ch_f^{+1}} = \underline{Ch'_f}$. Now we finish proving fact 3. In the second event of $r^w$ the term $\{5, a, e, C1'_f, C2'_f\}_{\overrightarrow{k^0_b}}$ is generated. Note that $e$ is forwarded from the message received by $r^w$. Then according to the design of $run$, the attacker send this term to the role instance $p1^w$ executed by $a$, then the term $T = \{5, b, e, C1'_f, C2'_f\}_{\overrightarrow{k^0_a}}$ is generated in the second event of $p^w$. $T \in know_I^{E'_w}$. Fact 3 is proved.

Lemma 1 is proved. ■

The ***finishing events***: The well-known attack to the PKNS protocol discovered by [4] is listed in Fig. 2, side by side with the finishing action steps. The receiving event of $B$ in the form of $-(I(A) \Rightarrow B) : Msg$ means that $B$ considers the message is sent from $A$ but actually is sent from $I$, while $I$ impersonates $A$. The finishing events are carried out to two role instances $s^{finish}$ and $r^{finish}$, of the two roles $S_{final}$ and $R_{final}$ respectively, executed by $a$ and $b$ respectively. $s^{finish}$ and $r^{finish}$ are included in $run.R$. The events of $s^{finish}$ and $r^{finish}$ are interleaved in a sequence exactly as the attack to the PKNS protocol, and this sequence is appended to $run.E$.

In the sequence of final events, 1) 4) and 5) belong to $s^{finish}$, while 2) 3) and 6) belong to and $r^{finish}$. In 2, the attacker impersonates $a$ and constructs a fake message where the term $\{5, b, e, q_{final}, C1, C2\}_{\overrightarrow{k^0_a}}$ is instantiated by any final configuration term $I$ can obtain from running the transition events. Since $M$ will reach a final configuration, by Lemma 1, it is guaranteed that a term of the form $\{5, b, e, q_{final}, C1, C2\}_{\overrightarrow{k^0_a}}$ is included in $know_I(E')$ where $E'$ is the prefix of $E$ up to the last transition events. Then there is no problem that the attacker can obtain the messages of 2) and 4), before they are received, and then $n_b \in know_I(E)$ after 5), and $n_b$ is a secret term. The event 6) is not necessary for the attacker to steal the secrecy $n_b$, but it is needed to finish an authentication attack to $b$, discussed later in this section. Direction 1 is proven.

| | | |
|---|---|---|
| 1) | $\#_A(N_A)$ | |
| | $+(A \Rightarrow I)$ : | $\{A, N_A\}_{\overrightarrow{k^1_I}}$ |
| 2) | $-(I(A) \Rightarrow B)$ : | $\{A, N_A\}_{\overrightarrow{k^1_B}}$ |
| 3) | $\#_B(N_B)$ | |
| | $+(B \Rightarrow I(A))$ : | $\{N_A, N_B\}_{\overrightarrow{k^1_A}}$ |
| 4) | $-(I \Rightarrow A)$ : | $\{N_A, N_B\}_{\overrightarrow{k^1_A}}$ |
| 5) | $+(A \Rightarrow I)$ : | $\{N_B\}_{\overrightarrow{k^1_I}}$ |
| 6) | $-(I(A) \Rightarrow B)$ | $\{N_B\}_{\overrightarrow{k^1_B}}$ |

Fig. 2. The attack to PKNS protocol

| | | |
|---|---|---|
| 1) | $\#_a(n_a, c1_a, c2_a)$ | |
| | $+(a \Rightarrow I)$ : | $\{1, a, n_a, \{5, I, e, q_{final}, c1_a, c2_a\}_{\overrightarrow{k^0_a}}\}_{\overrightarrow{k^1_I}}$ |
| 2) | $-(I(a) \Rightarrow b)$ : | $\{1, a, n_a, \{5, b, e, q_{final}, C1, C2\}_{\overrightarrow{k^0_a}}\}_{\overrightarrow{k^1_b}}$ |
| 3) | $\#_b(n_b)$ | |
| | $+(b \Rightarrow I(a))$ : | $\{2, n_a, n_b\}_{\overrightarrow{k^1_a}}$ |
| 4) | $-(I \Rightarrow a)$ : | $\{2, n_a, n_b\}_{\overrightarrow{k^1_a}}$ |
| 5) | $+(a \Rightarrow I)$ : | $\{3, n_b\}_{\overrightarrow{k^1_I}}$ |
| 6) | $-(I(a) \Rightarrow b)$ | $\{3, n_b\}_{\overrightarrow{k^1_b}}$ |

Fig. 3. The final action steps of the run construct in the proof of Theorem 1.

## *B. Proving Direction 2 for Theorem 1*

**Proving the Observations in Direction 2** The following five observations are helpful to prove direction 2.

*Observation 1*: First, every connection term of the form $\{7, a/b, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k^0_{a/b}}}$, and every configuration term of the form $\{5, a/b, e, q, X, Y\}_{\overrightarrow{k^0_{a/b}}}$ must be constructed by a regular agent $a$ or $b$, $a \neq I$ $B \neq I$. The reason is that the attacker does not know $k^0_a$ or $k^0_b$. Second, two encrypted terms appearing in $run$ with different format cannot be unified and cannot be used interchangeably, due to the constants 1, 2, 3, 5 and 7.

*Observation 2*: A term of the form $\{7, a/b, X, z\}_{\overrightarrow{k^0_{a/b}}}$ will never be generated in the run. If it can be generated, it must be an instance of the term $\{7, A_f, Ch_f, Ch_f^{+1}]\}_{\overrightarrow{k^0_{B_f}}}$, for $h \in \{1, 2\}$. But then $z$ is a freshly generated nonce by $a$ or $b$, impossible due to the assumption of unbounded fresh terms.

*Observation 3*: Given any term $X$, for all of the terms appearing in a run of $Pro$ in the form of $\{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$, $Y$ is the same. In other words, $X$ can only appear after a unique term $Y$ in any connection term. By the construction of the protocol, $\{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$ can only be generated as a connection term in three cases: 1) in the first message of role instance of $S_f$; 2) in the second message of a role instance of $R_f$, $1 \leq f \leq n$; 3) in the second message of a role instance of $S_{copy2}$; 4) in the second message of a role instance of $R_{copy2}$.

We show if there is a term $T$ of the form $\{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$ generated in case 4), there must be a term of the form $\{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$ which is generated earlier in case 1) or 2) or 3). Note in case 4) a role instance $r$ of $R_{copy2}$ will not change the order between $Y$ and $X$, in the sense that if in the second message of $r$ a term $T = \{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$ is sent, then in the first message of $r$ a term $T'$ of $\{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$ must be received while the position of $a$ and $b$ are swapped. The question is how $T'$ is generated. If we consider $T'$ again is generated in case 4), then we can find another earlier generated term again of the form $\{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$. This backward reasoning cannot be infinite since the run has finite many events. So case 4) can be reduced to case 1) or 2) or 3).

Suppose to the contrary there are two terms $T_1 = \{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$ and $T_2$=$\{7, a/b, Y', X\}_{\overrightarrow{k^0_{a/b}}}$ where $Y \neq Y'$. There are three situations. First, $T_1$ and $T_2$ are both generated in cases 1) or 2) or 3). It is impossible, since it means $X$ will be generated as a fresh nonce twice. Second, one of $T_1$ and $T_2$ is generated in case 1) or 2) or 3), while the other one is generated in case 4). Third, both $T_1$ and $T_2$ are generated in case 4). The second and third situations both imply the first situation, by the reasoning of the above paragraph, and are impossible.

*Observation 4*: For every term $X$, $X$ can only encode at most one number, and especially $z$ can only encode 0.

We can see Observation 4 directly by Observation 3. If $X$ is $z$, then by Observation 2, there can only be one encoding sequence of $z$, which is $z$ itself. For an encoding sequence of $X$, if $X \neq z$, then by the definition of encoding sequence, there must be a term $\{7, a/b, Y, X\}_{\overrightarrow{k^0_{a/b}}}$ appearing at the end of a encoding sequence of $X$. By Observation 3, the term $Y$ preceding to $X$ is fixed. By the same reasoning, the term preceding to $Y$ in a encoding sequence of $X$ is also fixed. The same reasoning can be applied recursively backwards, until the term $z$, which is the starting point of a encoding sequence, and it is impossible for $z$ to appear in the middle of the encoding sequence, by Observation 2.

On the other hand, it is possible that there exist two different terms of the form $\{7, a/b, X, Y_1\}_{\overrightarrow{k^0_{a/b}}}$, and $\{7, a/b, X, Y_2\}_{\overrightarrow{k^0_{a/b}}}$, where $Y_1 \neq Y_2$. The reason is that during the run of the 2-counter machine, a counter can reach a number (encoded by $X$) several times, and then incremented multiple times from $X$, corresponding to the run of the protocol, each time a different nonce $Ch_f^{+1}$ is used as the incremented value. In other words, a number can be encoded by several different terms, while each term can only encode one number. If we connect the encoding terms together where $X$ is the parent of $Y$ if there is a term $\{7, a/b, X, Y\}_{\overrightarrow{k^0_{a/b}}}$ appearing in the run, then we can form a tree, whose top node is $z$. Every node (a term) of the tree, can have several children nodes, but can only have one parent node. Each term can appear at most once as a node in the tree.

*Observation 5*: The number 0 can only be encoded by $z$.

By Observation 2, it is impossible for $z$ to encode any positive number. One concern is that if $X$ appears in $\{7, b, X, Y\}_{\overrightarrow{k^0_a}}$ where $Y$ encodes 1, and $X \neq z$, then $X$ could be used as a term encoding 0. But since 1 is the $i\_value$ of $Y$, there must be a term $\{z, Y\}_{\overrightarrow{k^1_{g2}}}$ by the definition of $i\_value$. It is impossible by Observation 3.

We prove direction 2 by proving a stronger result below.

***Lemma 2:*** For an run of the protocol $Pro$, call it $run$, considering an insider $I$ as described earlier. For every configuration term $T$ generated in the form $\{5, a/b, e, q, C_1, C_2\}_{\overrightarrow{k^0_{a/b}}}$, and $T \in know_I(run.E)$, $T$ encodes a reachable configuration, say $(q, V_1, V_2)$, of the two counter machine $M = (Q, \delta)$, in the sense that $V_h = \underline{C_h}$, for $h \in \{1, 2\}$.

*Proof:* This lemma is proved by induction on the sequence of the configuration terms that $I$ can obtain in a $run$ of $Pro$.

For a term $T$ in the form of $\{5, a/b, e, q, C_1, C_2\}_{\overrightarrow{k^0_{a/b}}}$, according to the design of $Pro$, there are four situations that $T$ can be generated in a $run$. 1) $T$ is generated in a role instance of $S_0$ executed by $a$ or $b$. 2) In the second event of a role instance of $R_{copy1}$ executed by $a$ or $b$. 3) In the first message of a role instance of $S_{final}$ exeucted by $a$ or $b$. 4) In the second message of a role instance of a role $R_f$, for some $f$, $1 \leq f \leq n$. Note that $T$ cannot be generated by $I$ since $I$ will never know $k_a^0$ or $k_b^0$. $T$ cannot be generated in the first event of some role instance of $S_f$ or $S_{copy1}$ executed by $a$ or $b$, since a fresh nonce honestly generated by $a$ or $b$ will take the position of $e$ in $T$, and it is impossible to be $e$. For $T$ genenerated in situation 3), it is impossible for $I$ to know $T$ since $T$ cannot appear in another message, and the only possible appearence of $T$ in a run is being encrypted using $k_a^1$ or $k_b^1$, while $I$ does not know $k_a^0$ or $k_b^0$ and cannot decrypt the message to know $T$. So we only need to consider $T$ generted in situation 1) 2) or 4). In situations 2) or 4), $T$ is generated after the role instance receiving a term also in the form $\{5, a/b, e, q, C_1, C_2\}_{\overrightarrow{k^0_{a/b}}}$, which must be generated and known to $I$ earlier than $T$. So the first $T$ known to $I$ must be generated in situation 1), which is the base case of the induction.

***Base case***: $T$ is generated in a role instance of $S_0$, and $T$ must be $\{5, a, e, q_0, z, z\}_{\overrightarrow{k^0_b}}$, or $\{5, b, e, q_0, z, z\}_{\overrightarrow{k^0_a}}$. since $z$ encodes 0, and the starting configuaration $(q_0, z, z)$ is a reachable one to $M$, base case is proven.

***Induction step***: Suppose for a configuration term $T$ known to $I$ in $run$, all of the earlier generated configuration terms (note that they are known to $I$ immediately after they are generated) satisfy the lemma, we need to prove that $T$ also encodes a reachable configuration of $M$.

There are three possible cases. First, $T$ is generated in situation 1). The lemma is true by the same reason for the base case.

Second, $T$ is generated in situation 2) by a role instance $r$ of $R_{copy1}$. Say $T$ is $\{5, b, e, q, X, Y\}_{\overrightarrow{k^0_a}}$, and $r$ is executed by $a$. Then the term $\{5, a, e, q, X, Y\}_{\overrightarrow{k^0_b}}$, call it $T'$, must be received earlier in $r$ before $r$ sends $T$. By the induction hypothesis, $T'$ encodes a reachable configuration $(q, V_1, V_2)$ of $M$, where $V_1 = \underline{X}$ and $V_2 = \underline{Y}$. Then $T$ should encode the same configuration, snce $q$, $X$, and $Y$ appear in the same in $T$ as in $T'$. So the lemma is true in the second case.

Third, $T$ is generated in situation 4) by a role instance $r$ of $R_f$, $1 \leq f \leq n$. According to the design of $R_f$, $T$ must be the term, say $\{5, a/b, e, q', C1'_f, C2'_f\}_{\overrightarrow{k^0_{a/b}}}$. Then $r$ must receive in its first event a term $\{5, a/b, e, q, C1_f, C2_f\}_{\overrightarrow{k^0_{a/b}}}$, which is a configuration term, call it $T'$. By the induction hypothesis, $T'$ must encode some reachable configuration $(q, V_1, V_2)$, where $V_1 = \underline{C1_f}$ and $V_2 = \underline{C2_f}$. Suppose $R_f$ corresponds to a transition rule $t$ of $M$, we want to show that $T$ encodes the configuration $(q', V_1', V_2')$, which is reached by applying $t$ to $(q, V_1, V_2)$ in a computation of $M$.

First, we show that $t$ is applicable to $(q, V_1, V_2)$. $t$ must have the form of $[q, i_1, i_2] \rightarrow [q', j_1, j_2]$. There are different cases to consider based on the possible values of $i_h$, for $h \in \{1, 2\}$.

- If $i_h = 1$, we need to show that $V_h > 0$. By the construction of $R_f$, there is a term $\{7, B_f, Ch_f^{-1}, C_h\}_{\overrightarrow{k^0_{A_f}}}$ included in the first message of $R_f$. This term must be instantiated by a ground connection term in $r$, as showed by Observation 1. By Observation 2, $Ch_f \neq z$. Since $Ch_f$ must encode either a positive number or 0, while 0 can only be encoded by $z$ as showed by Observation 5, so $\underline{Ch_f} > 0$. By the induction hypothesis, $V_h = \underline{Ch_f}$. So $V_h > 0$.
- If $i_h = 0$, then we need to show that $V_h = 0$. By the construction of $R_f$, $Ch_f = z$. Obviously $0 = V_h = \underline{Ch_f}$.

So $t$ is applicable to $(q, V_1, V_2)$.

Second, we need to show that after applying $t$ to $(q, V_1, V_2)$, the new reachable configuration $(q', V_1', V_2')$ is encoded by $T$. By the construction of $R_f$, the state constants $q$ and $q'$ in $r$ match with states of $t$, so we only need to show that $V_h' = \underline{C_h'}$. There are different cases to consider for the possible values of $j_h$, for $h \in \{1, 2\}$.

- If $j_h = 0$, then $V_h' = V_h$. Then, by the construction of $R_f$, it must be true that $Ch_f' = Ch_f$. Since $V_h = \underline{Ch_f}$ by the induction assumption, $V_h' = \underline{Ch_f'}$.
- If $j_h = +1$, then $V_h' = V_h + 1$. By the construction of $R_f$, In the second message of $r$, there must be a term $Ch_f^{+1}$ freshly generated, and a connection term $\{7, A_f, Ch_f, Ch_f^{+1}]\}_{\overrightarrow{k^0_{B_f}}}$ is generated, and $Ch_f' = Ch_f^{+1}$. Note that in $r$ $A_f$ and $B_f$ are instantiated either by $a$ or $b$. Then by the definition of encoding, $\underline{Ch_f} + 1 = \underline{Ch_f^{+1}} = \underline{Ch_f'}$. Since $V_h = \underline{Ch_f}$, by the induction hypothesis, and $Ch_f$ can only encode a unique number by Observation 4, $V_h' = V_h + 1 = \underline{Ch_f} + 1 = \underline{Ch_f'}$.
- If $j_h = -1$, then $V_h' = V_h - 1$. By the construction of $R_f$, there must be a term $\{7, B_f, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k^0_{A_f}}}$ included in the first message of $r$. $Ch_f' = Ch_f^{-1}$. By the induction hypothesis, $V_h = \underline{Ch_f}$, and $V_h > 0$. By Observation 3, $Ch_f \neq z$, since $z$ can only encode 0. Then by the definition of encoding sequence, there exists a term

$\{7, a/b, Y, Ch_f\}_{\overrightarrow{k^0_{a/b}}}$ in the encoding sequence of $Ch_f$, which has appeared in the run and known to $I$, where $V_h - 1 = \underline{Ch_f} - 1 = \underline{Y}$. By Observation 3, the $Y$ appearing in $\{7, a/b, Y, Ch_f\}_{\overrightarrow{k^0_{a/b}}}$ is unique, Then, the attacker can only use $\{7, a/b, Y, Ch_f\}_{\overrightarrow{k^0_{a/b}}}$ as the term $\{7, B_f, Ch_f^{-1}, Ch_f\}_{\overrightarrow{k^0_{A_f}}}$. So $V'_h = V_h - 1 = \underline{Ch_f^{-1}} = \underline{Ch'_f}$.

So the induction step is proved. Lemma 2 is proved. ■

***Lemma** 3:* Considering a $run$ of the protocol $Pro$ with an insider attacker, for a secret term $n_b$ generated by $b$ for $a$, $nb \in know_I(run.E)$ if and only if there is a configuration term $T$ of the form $\{5, b, e, q_{final}, C1,\ C2\}_{\overrightarrow{k^0_a}}$ such that $T \in know_I(run.E)$ before $I$ knows $n_b$.

*Proof:* This lemma follows the analysis of the behavior of the PKNS protocol and its attack, which has been extensively studied [4] [27]. Details of the attack are listed in Fig. 2. For the PKNS protocol, $I$ can steal the secret term $N_B$ if and only if $I$ can construct a message $\{A, N_A\}_{\overrightarrow{k^0_B}}$ and then send it to $B$ (showed above in event 2), where $I$ got $N_A$ from a message $\{A, N_A\}_{\overrightarrow{k^0_I}}$ sent earlier from $A$ to $I$.

Although in the communication sequence between $S_{final}$ and $R_{final}$ the NS protocol is adjusted, but the same reasoning can show that $I$ can steal $n_b$ if and only if $I$ can construct a message $a, b, \{a, na, \{5, b, e, q_{final}, C1, C2\}_{\overrightarrow{k^0_a}}\ \}_{\overrightarrow{k^1_b}}\}$, where $n_a$ is sent from $a$ to $I$. Since $I$ can always get such a $n_a$, it is clear that $I$ can steal $n_b$ if and only $I$ can get $\{5, b, e, q_{final}, C1, C2\}_{\overrightarrow{k^0_a}}$. The constants 1, 2, 3, 5, and 7 makes different kinds of encryption terms not pair-wisely unifiable, which makes the proof more obvious. Lemma 3 is proved. ■

Now we finish the proof of direction 2. By Lemma 3 and Lemma 2, the attacker can obtain a secret term if and only if $M$ can reach a final configuration. It is crucial for the attacker to be an insider to carry out the attack to the NS protocol, otherwise $A$ will not send a message to $I$ and $I$ cannot get $N_a$ to trigger the attack. Therefore $I$ has to be an insider to steal $n_b$. The message size (the number of atomic terms appearing in a message) can be bounded by 19 in a run, which is the size of the longest message which may appear in the role $R_f$, $1 \leq f \leq n$. The number of agents can be bounded by 3. The number of events in a role can be bounded by 2. Theorem 1 is proved.

## C. Proving Theorem 2

*Proof:* The secrecy attack to PKNS protocol listed in Fig. 2 is also an authentication attack, since the goal $Role_B \rightarrow Role_A$ with shared data $\{A, B, N_A, N_B\}$ is violated. When a role instance of $Role_B(A, B, N_A, N_B)$ finishes in the attack which is a run, the corresponding role instance of $Role_A(A, B, N_A, N_B)$ does not exist, while only a role instance of $Role_A(A, I, N_A, N_B)$ exists. This attack has been extensively studied. The attacker can launch this attack if and only if $I$ can obtain the needed $N_A$ to construct the message of 2) to be received by $B$. In order for $I$ to obtain $N_A$, $I$ have to be an insider so that $A$ recognize $I$ and is willing to start a conversation with $I$.

We choose the goal of $R_{final} \rightarrow S_{final}$. According to the CS between the two roles listed earlier, the two roles share all of the parameters $\{A, B, N_A, N_B, C1_{final}, C2_{final}\}$. In the attack showed in Fig. 2, this goal is violated since when a role instance $R_{final}(A, B, N_A, N_B, C1_{final}, C2_{final})$ finishes, the corresponding role instance of $S_{final}(A, B, N_A, N_B, C1_{final}, C2_{final})$ does not exist. By the same reasoning, $I$ can launch the attack if and only if $I$ can obtain the terms $N_A$ and a configuration term of the form $\{5, B, e, q_{final}, C1_{final}, C2_{final}\}_{\overrightarrow{k^0_A}}$, which means $I$ has to be an insider to chosen by $A$ as the interlocutor, and the 2-counter machine has to reach a final configuration, showed by the proof of Theorem 1. In conclusion, the authentication goal $R_{final} \rightarrow S_{final}$ can be violated if and only if $M$ can reach a final configuration. ■