Secrecy Checking of Protocols: Solution of an Open Problem [*]

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-07-04

April 9, 2007

Keywords: Cryptographic protocols, secrecy, undecidability.

Abstract

This paper proves the undecidability of an open problem on the complexity of checking secrecy of cryptographic protocols due to Durgin, Lincoln and Mitchell. Our presentation clarifies several features of the Dolev-Yao model including a few notions that are commonly glossed over or vaguely described. The proof is by a reduction scheme from 2-counter machines to protocols, and we prove both directions of the reduction in detail for rigor and correctness. The modeling and proof method are generally applicable and can be easily adapted to solve other problems about the complexity analysis of checking properties of protocols.

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

Secrecy Checking of Protocols: Solution of an Open Problem [*]

Zhiyao Liang and Rakesh M Verma

Abstract

This paper proves the undecidability of an open problem on the complexity of checking secrecy of cryptographic protocols due to Durgin, Lincoln and Mitchell. Our presentation clarifies several features of the Dolev-Yao model including a few notions that are commonly glossed over or vaguely described. The proof is by a reduction scheme from 2-counter machines to protocols, and we prove both directions of the reduction in detail for rigor and correctness. The modeling and proof method are generally applicable and can be easily adapted to solve other problems about the complexity analysis of checking properties of protocols.

Index Terms

Cryptographic protocols, secrecy, undecidability.

I. Introduction

Checking and analyzing the security properties of cryptographic protocols has been one of the most contentious and important challenges nowadays when networks are ubiquitous. A significant research direction in this area is to check secrecy and authentication of the protocols against a Dolev-Yao attacker [1] assuming that the cryptographic protocols cannot be broken. Since Lowe discovered an attack on the public key Needham-Schroeder protocol [2], [3] 17 years after it was published [4], many papers have been published on this topic. To check secrecy failure or correctness is a very hard problem. One bound on the complexity of secrecy problem is that when the number of role instances in the protocol run is bounded, the secrecy problem is NP-complete [5], even when composite keys are allowed [6].

Secrecy checking is undecidable assuming unbounded number of role instances in a protocol run (together with other specific assumptions). Undecidability of secrecy checking is mentioned by several papers [7] [8] [9] [10] [11] [12] [13] [14], and [9] [10] [11] [13] provide proofs with details. The survey paper [14] is partly motivated by the work of [13] and partly to clarify the sketched proof in [10] on showing undecidability of secrecy by reduction from PCP (Post Correspondence Problem).

In [11] and [10] the authors use MSR (multi-set rewriting) to analyze protocols. In these papers the focus is on bounding the symbolic size of each message instance that can appear in a run of the protocol, and the number of messages in every role of the protocol is bounded. When the total number of distinct nonce instances that can be generated by regular principal instances is unbounded, and the symbolic message size of all messages instances are bounded by a number, secrecy verification is undecidable. The proof is by a two-stage reduction from the halting problem of a Turing machine with the style of Turing machine tableau to Horn clause theories without function symbols and then from Horn clause theory to protocols specified as a set of roles. When the attacker can generate unbounded number of nonces and the regular agents can record nonces and check the uniqueness of each received nonce in a run, and a run can have unbounded number of role instances, the complexity of checking secrecy is an *open problem* [10] [11]. The open problem is stated more precisely in theorem 1, after our modeling has been presented.

In [13] the authors show that the undecidability result of [10] can be proved more directly by a reduction from the reachability problem of a 2-counter machine to the secrecy checking problem of a protocol as a set of roles (we call the protocol role-oriented). In addition, by replacing unique nonces with unique composite terms, [13] proved the undecidability of secrecy checking when the symbolic size of message instances are unbounded, while the nonce instances generated in the run are bounded (in fact, no nonce generation is required).

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

It is fair to say that the source of undecidability of secrecy checking is the unbounded number of terms that can appear in the messages sent or accepted by the regular agents in a run of the protocol. Note that the closure of the attacker's knowledge is always a set of infinite number of terms, which is not the source of the undecidability. This description of the source of undecidability uniformly covers the cases proved by [11] [10] and [13] where unbounded number of nonces can be generated and message instances have bounded size, and the case proved by [13], where no nonce is generated, but a message can be arbitrarily large.

Two factors are crucial for us to solve the open problem. First, we model the problem carefully and second, we utilize an improved and more direct reduction scheme, from 2-counter machines to security protocols. We give a rigorous and complete proof of correctness of the reduction. Our scheme is also applicable beyond the open problem.

A key idea in the work of [10], [11] and [13] is to use term encoding. In [10] and [11] a nonce is used to encode a position in a Turing machine tableau. When two nonces N_1 and N_2 are encrypted in a term $\{N_1, N_2\}_K^{\leftrightarrow}$, for some symmetric key K, it means that the two positions encoded by N_1 and N_2 are neighbors. In [13] a nonce is used to encode a counter which is a number, and similarly $\{N_1, N_2\}_K^{\leftrightarrow}$ means that N_2 encodes a number that is 1 larger than the number encoded by N_1.

Our reduction scheme using 2-counter machine is inspired from [13]. However, there are some key differences. Because of the constraints of the open problem, we cannot use their scheme directly and need a new idea of stamping nonces with agent id's. Moreover, we have found and fixed two errors in the reductions of [13]: a counter can be negative, and zero can be used as a positive number. Details of the errors and our fixes are included in Appendix H. The proof of correctness of the reduction in [13] is sketchy and consequently misses the two errors.

The organization of this paper is as follows. In Section II, we present our notations and the definitions of protocol model. In Section III we present the main result of this paper, which solve the the open problem in theorem 1, together with a stronger consideration of the open problem in theorem 2. Section IV concludes the paper. The Appendix presents additional information about the open problem and the design of the proofs.

II. Modeling

Commonly the Dolev-Yao model [1] refers to the assumption of a dominating attacker, which means the attacker can record and intercept every message appearing in the network, and the assumption of perfect encryption, which means an encrypted term can not be decrypted by an agent unless the agent has the corresponding decryption key.

There are different formalization systems to describe the Dolev-Yao model, such as (not a comprehensive list) the model defined by Paulson [15], Multiset Rewriting (MSR) [10] and Strand Space [16]. We feel that the different models are essentially equivalent, since the assumption of Dolev-Yao model is the same. There are papers that compare the technical difference between different models. Our model is somewhat similar to Paulson's. For example, the actions in the definition of run is close to what Paulson defined as a trace, and attacker's knowledge is defined as the closure of applying a set of rules to an initial term set. However, our presentation clarifies several features of the Dolev-Yao model including a few notions that are commonly glossed over or vaguely described.

We introduce notations before defining a protocol and a protocol run.

A. Notations

Notations are chosen in a style that is commonly used in the literature, e.g., [2]. 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 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***. Composite terms are allowed to be the keys (composite keys) for symmetric encryption, but a key for 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 [6] and [17].

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 which 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 modeling of this paper can be extended to deal with an insider attacker. We will use D to refer to the attacker, Def. 6, as an object with several features, and $D.name = I$.

We assume the free term algebra, which means that there is only one way to construct a term. This assumption is commonly made in papers based on the Dolev-Yao model.

B. Protocol, Protocol Run, and Secrecy

A published protocol is commonly presented as a sequence of message exchanges, which describes a normal run, such as those collected in the survey [18]. We call this sequence *communication sequence*. In all of the papers doing complexity analysis of protocols that we are aware of, such as [9] [10] [13] [14] [5] [6], a constructed protocol is presented directly as a set of roles. We call this kind of protocol as *role-oriented*, or *RO* for short. Usually the first step of analyzing a protocol is to translate it into a set of roles. An example of a protocol as a communication sequence and the RO protocol translated from it, is presented in Appendix N. A RO protocol may be *non-matching*, that is, it does not correspond to any communication sequence, usually due to the fact that for some receiver's role, the corresponding sender's role is missing. The protocols constructed in the proofs of [9] [10] [13] [14] [5] [6] are all non-matching RO protocols. For compatibility with other papers and especially with [11] and [10], which describe the open problem, in this paper a protocol is defined in Def. 4 in RO style, and in the proof of the theorem solving the open problem, the protocol constructed is RO and non-matching. Proofs for matching RO protocols can be derived from the non-matching RO proofs. We present further discussion of the protocol specifications in Appendix I.

Definition** 1:* An ***agent is a tuple $[name, init, mem]$.

- $name$: The unique name of the agent.
- $init$: The initial knowledge of the agent. It is a set of (ground) terms that are initially known by the agent before running a protocol.
- mem: The set of terms that the agent have remembered so far in the current protocol run.

Agents can be categorized into different kinds depending on their initial knowledge patterns, which are defined in the protocol, and thus they can do different tasks. In many cases, there can be two kinds of agents, the a ***special agents*** who execute special and usually critical tasks, such the server roles, and the ***common agents*** who execute the roles other than the server roles. If an agent always acts according to the description of the protocol, it is called a ***regular agent***.

A role (defined later) will specify the requirement of $init$ for the agent who can execute the role. The mem field means that the agent may remember terms which have occurred in every action of every earlier role instance in addition to the current role instance executed by the agent in the current run. An agent can participate in a run several times, each time executing a different role instance. How the memory of an agent is initialized and updated will be described in the definition of the protocol.

An agent is called a principal in some papers. Note that in common situations it is not required that an agent should remember any term of earlier role instances, But the mem field of an agent is exactly what is needed for the open problem, which will be solved in Theorem 1.

Definition** 2:* An ***action can be an ***internal action*** or an ***external action***. Let P, A, and B be agent names. The internal action of fresh term generation is denoted as $\#_P(t1, t2, \cdots)$, where $t1, t2, \cdots$ represent the fresh terms (nonces) generated by agent P before P sends a message that contains these fresh terms. An external action can be a message sending or a message receiving. The action of agent A to send a message Msg, when the intended receiver is B, $A \neq B$ is denoted as $+(A \Rightarrow B) : Msg$. The action of agent A to receive a message Msg from a supposed sender B, $A \neq B$ is denoted as $-(B \Rightarrow A) : Msg$.

An ***action step*** is a sequence of actions. It has four forms. 1) $\#_I(term1,\ term2, \cdots)$; 2) $+(A \Rightarrow B) : Msg$; 3) $-(B \Rightarrow A) : Msg$; 4) $\#_A(t1, t2, \cdots)\ \ +(A \Rightarrow B) : Msg$;

We assume that $A \neq B$, since in practice an agent should not or need not to exchange messages with herself.

An internal action could be a value assignment, an equivalence check, a decryption, a type check, etc. In the papers on Dolev-Yao attacks, usually internal actions are not expressed explicitly in the protocol code or in the corresponding roles, which makes it more difficult to formally describe security protocols and their runs. Some internal action can be implicitly described by the protocol code, such as equivalence checking for the values of the same term. For example, if the term X appears both in message 1 and 2, and these two messages are sent and received, respectively, by the same agent A, then A needs to check the equivalence of the different occurrences of X during A's execution of a role instance of the protocol. However, some other internal actions cannot be expressed implicitly. In this paper, the only kind of internal actions explicitly expressed in the action code are the fresh term generations. Some other internal actions, such as disequality check of terms, when they are required to be expressed , such as those required by the open problem, are described in the conditions of a specific role of the protocol.

For the 4^{th} kind of action step, a fresh term generation action (by a regular agent) is showed immediately before the corresponding message sending action where the sent message contains the generated fresh terms, as if they are a single action. For a nonce generation action X, X will be a stand alone action step only if it is executed by the attacker, described as the 1^{st} kind of action step. The 4^{th} kind of action step is a sequence of two actions, while the other three kinds are sequences with a single action.

This paper assumes the uniqueness of nonce creation, which is also assumed by all of the papers we have noticed addressing the Dolev-Yao model. It means that when an action $\#_A(\cdots, X, \cdots)$ exists in a run (defined later), X is instantiated by a ground atomic term (or constant), which is different from any other term that has appeared in the run so far. Here A is variable representing any agent name, which could be I. Note that if the attacker, with name I, wants to recycle some old term as the nonce X, then the corresponding action of $\#_I(\cdots, X, \cdots)$ will not appear in the run.

Definition 3: A ***role type*** or ***role template*** or a ***role*** for short, is a tuple: $[RID, agent, vars, conds, acts]$

- RID: The UID of the role, which is a constant.
- $agent$: The agent who will execute the role template.
- $vars$: The set of variables that appear in $acts$ or $conds$. Note that all of the atomic terms appearing in the role that are not variables are constants.
- $acts$: The sequence of action steps numbered sequentially starting from 1.
- $conds$: the internal actions that are not implicitly expressible by the action code. The notation $n.pre$: $(cond1, cond2, \cdots)$ represents the conditions that should be checked and satisfied before the n^{th} action step (before accepting a received message, or before sending a message) is executed, where n is an action step number. The statement $n.post$: $(cond1, cond2, \cdots)$ describes the conditions that must be satisfied after the n^{th} step of action is executed to update the properties of agents. Usually a condition of a role (expressed in this paper) can be one of the following forms: $X \neq Y$ (values of the two variables are different); $X \subseteq agent.init$; $X \subseteq agent.mem$; $X \notin agent.mem$; $agent.name = X$; $X \in Q$. X is a term or a set of terms depending on the context. Q is a set which should be defined in the description of the protocol or a protocol run. A "post" condition must be satisfied. For example $X \subseteq agent.mem$ means to add every element of the set X into $agent.mem$ if the element is not an element of $agent.mem$ yet.

The condition $agent.name = X$ is included in $1.pre$ to specify that X executes the role. If X 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 agent. Sometimes when SN and CN are specified in a run of the protocol as the names of special agents and common agents, the conditions $\{X\} \subset SN$ or $\{X\} \subset CN$ specifies X is special agent or a common agent. In the role if $agent.name \in CN$ or $agent.name \in SN$, then the role is called a ***special role*** or a ***common role*** respectively.

Definition 4: A ***protocol*** Pro is a tuple $[PID,\ roles,\ AN,\ pk,\ gk,\ rsts]$

- PID: The UID of the protocol, a constant.
- $roles$: A set of role templates.
- AN: Agent names. If an agent name X is included in AN, then X is called an ***insider***, otherwise X is an ***outsider***. Sometimes $AN = [SN, CN]$, where SN is the set of special agent names and CN is the set of common agent names.
- pk: A set of terms representing the public knowledge. This set of terms should be known by every one even an outsider.
- gk: A set of terms representing the group knowledge. The set of terms should be known by, and only by, all the insiders, but not by outsiders.
- $rsts$: The restrictions describing pk, gk, and the initial knowledge and initial memory (indicated as $mem^{initial}$) of the agents, and definitions of sets.

PID is only useful when several different protocols are considered together. The attacker knows the terms of pk even though the attacker may be an outsider. Commonly, the names and public keys of all regular agents are included in pk. If in the restrictions of a protocol and the conditions of its roles, mem fields of agents are not described, then it means that they are not needed or not defined. For a regular agent, its initial knowledge will be specified according its name (the AN field) and the restriction of the protocol (the $rsts$ field).

Definition 5: A ***role instance*** is a tuple $[agent, role, vmap, acts]$.

- $agent$: an agent who executes the role instance.
- $role$: The role template for this role instance.
- $vmap$: A function (a substitution) that maps a variable to a ground term. For a constant or a composite ground term c, $vmap(c) = c$. $vmap$ can also be obviously extended as a substitution to map a term or an action or a condition to its ground instance. A requirement of $vmap$ is that $vmap(role.conds)$ should be satisfied.
- $acts$: The sequence of (ground) action steps. $acts = vmap(role.acts)$.

Definition 6: The behavior of a ***Dolev-Yao attacker*** [1], or an ***attacker*** for short, call it D for dangerous, in a protocol run can be summarized by three aspects:

1) D records every message immediately when the message appears in the network.

2) D can prevent a principal from receiving a message that has been sent.
3) D has the freedom to send out any term to any agent as a message at any time of the run, as long as D can obtain the term before sending it out.

A Dolev-Yao attacker is a tuple $[name, init_I, know_I]$.

- $name$: By convention $name = I$.
- $init_I$: A set of ground terms which is the initial knowledge of the attacker.
- $know_I$: A function. After a sequence E of action steps has been executed, $know_I(E)$ is the set of terms that the attacker can obtain.

For an element X, and a sequence E, $X \in E$ means that (by abuse of notation) X appears as an element in E. Given a sequence of action steps E, $know_I(E)$ is calculated as the closure of applying the following rules.

- Initial knowledge: $X \in init_I \quad \mapsto \quad X \in know_I(E)$
- Sent message recording: $+(A \Rightarrow B)\ Msg \in E$, or $\#_A(terms) + (A \Rightarrow B)\ Msg \in E$, $A \neq I \quad \mapsto \quad Msg \in know_I(E)$
- Fresh term generation: $\#_I(\cdots X \cdots) \in E$, or $\#_I(\cdots X \cdots) + (I \Rightarrow B)\ Msg \in E \quad \mapsto \quad X \in know_I(E)$

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 $k_G^m \in know_I(E)$, $m \in \{0, 1\} \quad \mapsto \quad \{X\}_{k_G^m}^{\rightarrow} \in know_I(E)$
- Symmetric key encryption: $X \in know_I(E)$, and $Y \in know_I(E) \quad \mapsto \quad \{X\}_Y^{\leftrightarrow} \in know_I(E)$

Analysis:

- List breaking: $[X, Y, \cdots] \in know_I(E) \quad \mapsto \quad \{X, Y, \cdots\} \subset know_I(E)$
- Asymmetric key decryption: $\{X\}_{k_G^m}^{\rightarrow} \in know_I(E)$, $k_G^{1-m} \in know_I(E)$, $m \in \{0, 1\} \quad \mapsto \quad X \in know_I(E)$
- Symmetric key decryption: $\{X\}_Y^{\leftrightarrow} \in know_I(E)$, and $Y \in know_I(E) \quad \mapsto \quad X \in know_I(E)$

For a sequence W, the prefix of W including the first n items of W is denoted by $W^{1:n}$. Sequence U is a subsequence of W is denoted by $U \leqslant W$. The length of W, or the number of elements of W, is denoted as $len(W)$. The $\diamond$ operator concatenates a sequence and an element together to form a longer sequence.

Definition** 7:* A ***run of a protocol Pro with an attacker D is a tuple: $[Pro,\ D,\ R,\ AN,\ E,\ conds]$.

- Pro: The protocol.
- D: The attacker.
- R: A set of role instances that are executed honestly by regular agents.
- AN: The names of the agents who can legally participate in a run. AN instantiates $Pro.AN$. Sometimes $AN = [CN, SN]$ where CN is the set of common agent names and SN is the set of special agent names.
- E: A sequence of actions steps.
- $conds$: The conditions required.

1) $Pro.rsts$ should be satisfied. That is, $Pro.pk$, $Pro.gk$, and $P.init$, $P.mem^{initial}$, for each P in with the name in AN, execpt when P is the attacker, and other set of terms defined by the protocol, are built according to $Pro.rsts$.

2) For each role instance $r \in R$, $r.agent.name$ appears in AN, and $r.agent.name \neq I$, all action steps of $r.acts$ are included in $run.E$ preserving the relative order.

3) For each $X \in E$ executed by some regular agent, $X \in r.acts$, for some $r \in R$.

4) If I appears in AN, then D is an insider (note that $D.name = I$), then $D.init_I$ will be instantiated as other regular agents with the same initial knowledge pattern according to $Pro.rsts$. Otherwise, D is an outsider, and $D.init_I = Pro.pk$.

5) If $Y \in E$, and E' is the prefix of E that ends immediately before Y, and Y is the $m + 1^{th}$ element of $r.acts$, for some $r \in R$ then $r.acts^{1:m} \leqslant E'$

6) Suppose $W \diamond X$ is a prefix of E, where W is a sequence of action steps, $X \in r.acts$ for some $r \in R$. If $X = -(A \Rightarrow B)msg$, where B is the name of a regular agent , then $msg \in know_I(W)$.

7) When D is an outsider, the only actions executed by D that are explicitly included in $run.E$ are the fresh nonce generation actions of the form $\#_I(terms)$.

8) When D is an insider, the actions of the attacker that are explicitly included in $run.E$ are described in 3 cases. 1. Actions of the form $\#_I(terms)$. 2. If an action, in the form of $+(A \Rightarrow I)MSG$, is included in $run.E$, then immediately after this action, an action $-(A \Rightarrow I)MSG$ is included in $run.E$. 3. If an action in

the form of $-(I \Rightarrow A)MSG$, then immediately before this action, an action of $+(I \Rightarrow A)MSG$ is included in $run.E$.

The set of all possible runs of a protocol Pro and an attacker D with some specific initial knowledge pattern of D, is $Runs^{Pro:D}$, and if in addition AN is fixed, is $Runs^{Pro:D:AN}$.

Some explanation for the conditions may be helpful. For 1), the attacker D is the only agent who may not follow the restrictions of the protocol. For 2) and 3), only the regular agents' actions are organized into role instances. For each action Y executed by a regular agent, there are two implicit properties of Y, the role instance r in which Y appears, and the number m of the action step Y. For 4), we assume the attacker will not be a special agent, which usually is a trustworthy server doing critical computation in the protocol. For 5), the causal earlier relationship between two actions X and Y in the same role instance r is preserved in $run.E$. Note that we do not require all actions of $r.acts$ to be included in $run.E$, only a prefix of $r.acts$ including Y. For 6), before a regular agent receives a message, the attacker should have been able to construct it. For 7), the attacker's behaviors of sending or receiving messages are implicitly described in a run, since every message received (sent) by a regular agent is considered to be sent (received) directly by the attacker. When the attacker is an outsider, his fresh term generation actions are explicitly recorded for teh convenience of expressing the attacker's knowledge of terms. For 8), to record some of the attacker's message sending or receiving actions explicitly is for the convenience of specifying other security goals we are investigating, e.g., some special authentication-related goals which can deal with an insider attacker. Since the open problem assumes an outsider attacker. So 8) is not directly relevant to this paper. More explanation of the modeling, discussion of the Dolev-Yao model and the attacker are included in the Appendices.

We assume every run has an implicit stage to distribute keys and establish the initial knowledge of agents.

Definition *8:* Given a protocol Pro, an attacker D, and a set of secret terms SEC, a ***secrecy goal*** is an assertion:

$$\forall run\ (run \in Runs^{Pro:D}) : \forall X\ (X \in SEC) : X \notin D.know_I(run.E)$$

A ***secrecy problem*** is to check whether a secrecy goal of a protocol can be violated. A ***secrecy attack*** on Pro with the attacker D is a run in $Runs^{Pro:D}$ such that some secret X is revealed ($X \in D.know_I(run.E)$).

Given a protocol Pro, when we talk about an arbitrary run, call it run, there are some fixed considerations and flexible considerations.

The required knowledge patterns for a special agent or a common agent are fixed as specified by $Pro.rsts$. Also the attacker's initial knowledge pattern is fixed (either an insider or an outsider).

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 [19] 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 [20] 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. [6], if the number of role instances is bounded, secrecy is decidable. In addition to the agent names, some sets defined in the protocol can be instantiated by different sets of ground terms in different runs. For example, the set of secret terms SEC. Also the interleaving of the role instances, which is the action sequence $run.E$, is flexible.

III. Solution of the Open Problem

In this section, we present the solution to the open problem. We are considering the lower bound of complexity. In other words, we show that given a set of conditions, in the worst case the problem is undecidable. Although there are special kinds of protocols that are decidable, they are not relevant to the theme of this paper.

The proof structure used in the solution of the open problem is generally utilizable and it not specially customized just for the open problem. It can be used to prove other theorems which are also challenging.

The open problem described in [10] and [11] is as follows. The protocols considered by [10] are bounded, which means two bounds. First, the number of messages in a role template (and also in a role instance), called the role length, is bounded. Second, the size of a message instance (the number of ground atomic terms appearing in a message, which is a term) that can appear in a run of the protocol is bounded. In other words, only the runs with bounded size of message instances are considered.

In [10], and all other papers showing proofs of undecidability or NP-completeness that we are aware of, cited in Section I, the attacker is considered as an outsider, since according to the proof of these papers, the attacker does not know a key that is known by all legal agents (based on our definitions, we may call the key K and $K \in Pro.gk$), and the attacker cannot legally participate in a protocol run as a regular agent.

The open question is on the complexity of checking secrecy when the attacker can generate unbounded number of nonces, the regular agents can only generate bounded number of nonces, while every regular agent will remember all nonces she has seen in the run and will only accept a term as a fresh nonce, and then record it, if the term is different from all of the recorded terms (we can consider the initial set of recorded terms is the initial knowledge of the agent).

The open problem is precisely stated in Theorem 1. Appendix A discusses it in more detail. When we consider all of the possible runs of a protocol, we have to cover the cases where type-flaw can occur. That is, a variable is instantiated by a ground term of different type. The most obvious case is that a variable representing a nonce (an atomic term) is instantiated by a composite ground term. In our proof to solve the open problem we also need to cover both the runs with or without type-flaws. More discussion on type-flaw is presented in Appendix G. Note that, in the scenario of the open problem, nonce generations depend on the attacker, and the attacker can always use a composite term as a nonce. So type flaw is not avoidable.

In the proof of Theorem 1 type flaws are allowed. More specifically, a variable is allowed to be instantiated by a composite term in a run. If only consider the runs of the protocol where type flaws do not occur, more specifically a variable is always instantiated by an atomic term (an agent name or a nonce) in the runs, the open problem is still undecidable. We only need to adjust the proof of Theorem 1 a little, and it is presented in Appendix L.

The essence of the proof is to reduce the halting problem (also called the reachability problem in [13]) of a deterministic 2-counter machine [21] with no input to the secrecy problem of a protocol, by translating every transition rule of the 2-counter machine into a transition role type of the protocol. We need to ensure that a 2-counter machine can reach its final state if and only if there is a run of the corresponding protocol in which the secret term is leaked.

Definition 9: A ***deterministic 2-counter machine*** with empty input is a pair (Q, δ), where Q is a set of states including the starting state q_0 and the accepting state q_{final} and δ is a set of transition rules. A configuration of a 2-counter machine can be described as 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, which are initially 0. The 2-counter machine can detect whether a counter is 0 or not. A transition rule of a 2-counter machine, (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 (q, C_1, C_2) is the configuration before the transition and (q', C_1', C_2') is the configuration after the transition. 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, which means C_h must be positive, since decrementing 0 is not allowed. The halting 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. Here _ represents an arbitrary possible value of the variable. In this case an arbitrary non-negative integer. Different occurrences of the _ symbols can represent different values. We assume (for convenience) that $q_0 \neq q_{final}$ and, for nontriviality, that δ is not empty.

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

Theorem 1: The open problem of [10] is undecidable. Specifically, checking secrecy is undecidable, assuming: (i) the protocol has bounded number of messages in a role (role length), (ii) considering only the runs where the sizes of messages are bounded, (iii) the number of role instances in a run of the protocol is unbounded, (iv) regular agents can generate only bounded number of nonces, (v) the attacker can generate unbounded many nonces, (vi) the internal action of disequality test on two terms is allowed, and (vii) when a term is supposed to be freshly generated nonce and is received by some regular agent, who records every nonce encountered, it must be different from all other terms the agent has recorded so far in the run, and then it is recorded by the agent.

Proof: We translate an arbitrary 2-counter machine into a protocol which fits in the scenario of the open problem. Implicitly, the open problem considers non-matching RO protocols and an outsider attacker. The same is considered in this proof. Every role has a different scope of variables. So a variable in role is independent of the variable with the same name in another role.

Given a 2-counter machine $M = (Q, \delta)$, let $Q = \{q_0,\ q_{final},\ q_1,\ q_2,\ \cdots,\ q_m\}$ and $\delta = \{T_1,\ T_2,\ \cdots,\ T_n\}$. The following is the description of the protocol Pro with a secrecy term sec, constructed according to M. We show that M can reach $(q_{final}, \_, \_)$ if and only if there is a run, call it run, $run \in Runs^{Pro:D}$ and $sec \in know_I(run.E)$, where D is an outsider, and sec is a secret term.

The messages received in a role is in the format of:

$$sender,\ receiver,\ receiver's\ role,\ \cdots \qquad ,$$

so the receiver of the message has clear hint to understand the message and know what she should do. The variables B, A_{final}, A_0, A_f, for $1 \le f \le n$ are agent names. We differentiate the namess of the variables representing the executors of different roles, including A_{final}, A_0, A_f, $1 \le f \le n$, for the clarity of the presentation, although a single variable A can be used in different roles. B represents the agent talking with the executor of every role.

In a role executed by an agent A, for the variables whose values are not determined by the agents other than A, we can categorize these variables in three kinds depending on A's different treatment to them. 1) The set of terms which A must check that they belong to the $A.init$, such as the agent names. 2) The set of terms which A does not care about whether A has seen them already or not, such as C_h and C_h^{-1}, no matter they should be nonces or not. 3) The set of terms which A must check its uniqueness (where the disequality $\neq$ applies), i.e., A has never seen it before, such as C_h^{+1}.

We will adapt the proof to show in theorem 2 that when the variables of kind 2) are not allowed, the open problem is still undecidable.

$Pro\ =\ [PID,\ roles,\ agents,\ AN,\ pk,\ gk,\ rsts]$

* PID is arbitrary.

* $roles = \{R_0,\ R_{final},\ R_1,\ R_2,\ \cdots,\ R_n\}$

- $R_0 = [RID,\ agent,\ vars,\ acts,\ conds]$
  - $RID = r_0$; $agent = [name,\ init,\ mem]$; $vars = \{A_0,\ B\}$
  - $acts = 1.\ +(A_0 \Rightarrow B):\quad A_0, B, \{q_0, z, z\}_{\vec{k}^0_{g1}}$
  - $conds = \{\ 1.pre: (q_0,\ A_0,\ r_0,\ B,\ k^0_{g1}\} \subseteq init,\ agent.name = A_0,\ \{A_0, B\} \subset CN, A_0 \neq B)\ \}$
- $R_{final} = [RID,\ agent,\ vars,\ acts,\ conds]$
  - $RID = r_{final}$; $agent = [name,\ init,\ mem]$; $vars = \{A_{final},\ X,\ Y,\ B,\ Sec\}$.
  - acts= 1. $-(B \Rightarrow A_{final}):\quad B, A_{final}, r_{final}, \{q_{final}, X, Y\}_{\vec{k}^0_{g1}}$

    2. $+(A_{final} \Rightarrow B):\quad A_{final}, B, Sec$
  - $conds =\ 1.pre:\quad (\{q_{final},\ A_{final},\ r_{final},\ B,\ k^1_{g1}\} \subseteq init,\ \ agent.name = A_{final}, \{A_{final}, B\} \subset CN,$
    $A_{final} \neq B)\ );$

    $2.pre:\quad (\ Sec \in init,\ \ Sec \in SEC\ )\ \}$

  SEC is defined in $Pro.rsts$. We do not specify the secret term Sec as a constant in the role. So even though the attacker knows the protocol code, the attacker does not trivially know the instance of Sec.
- For each $T_f \in \delta$, for some f, $1 \le f \le n$, suppose $T_f = [q, i_1, i_2] \to [q', j_1, j_2]$. $R_f \in roles$. R_f can be constructed according to T_f by the following description.

  $R_f = [RID, agent, vars, acts, conds]$
  - $RID = r_f$. $agent = [name, init, mem]$. $vars = \{A_f, B, C_1,\ C_2, C_1^{-1},\ C_1^{+1},\ C_2^{-1}, C_2^{+1}\}$.
  - $acts$: The following is the template of acts. The exact action sequence of each R_f will be adjusted by the specific T_f and a set of rewrite rules. Note that $q, q', i_1, i_2, j_1, j_2$ may represent different constants for different f, according to the 2-counter machine specification. The variables C_1' and C_2', which represent the new counter values, will only be used in the template and they will not appear in the actual code of the R_f, since they will be replaced by other terms after applying the rewrite rules.

    1. $-(B \Rightarrow A_f):\quad B, A_f, r_f, \{q, C_1, C_2\}_{\vec{k}^0_{g1}}, \{C_1^{-1}, C_1\}_{\vec{k}^0_{g2}}, \{C_2^{-1}, C_2\}_{\vec{k}^0_{g2}}, C_1^{+1}, C_2^{+1}$
    2. $+(A_f \Rightarrow B):\quad A_f, B, \{q', C_1', C_2'\}_{\vec{k}^0_{g1}}, \{C_1, [A_f, C_1^{+1}]\}_{\vec{k}^0_{g2}}, \{C_2, [A_f, C_2^{+1}]\}_{\vec{k}^0_{g2}}$

For $h \in \{1, 2\}$, the following restriction will hold. Described as "condition $\Mapsto$ effects". $W \rightarrowtail V$ means to replace

W with V in the above action code template of R_f. $W \rightarrowtail \varepsilon$ means to remove W. $W \in Msg_1$ means the assertion that the term W will appear in message 1. An implicit rule is that any term in the template of $R_f.acts$ which is not removed or changed will still appear in the code. We emphasize that a term will appear in a message in some rule, even without explicitly saying so, the fact should still hold. If in R_f some variables will not appear in the actions since they will be removed by applying the rules, then these variables will also be removed from other fields such as $R_f.vars$ and $R_f.conds$. If a rule is only applied to Msg_1, it is labeled with "in Msg_1". $h \in \{1, 2\}$.

1. $i_h = 0 \quad \Rrightarrow \quad C_h \rightarrowtail z;\ \{C_h^{-1}, C_h\}_{\overrightarrow{k_{g2}^0}} \rightarrowtail \varepsilon$
2. $i_h = 1 \quad \Rrightarrow \quad \{C_h^{-1}, C_h\}_{\overrightarrow{k_{g2}^0}} \in Msg_1$
3. $j_h = +1 \quad \Rrightarrow \quad C'_h \rightarrowtail [A_f, C_h^{+1}];\ C_h^{+1} \in Msg_1;\ \{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k_{g2}^0}} \in Msg_2$
4. $j_h = 0 \quad \Rrightarrow \quad C'_h \rightarrowtail C_h;\ \{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k_{g2}^0}} \rightarrowtail \varepsilon;$ In Msg_1 $C_h^{+1} \rightarrowtail \varepsilon$
5. $j_h = -1 \quad \Rrightarrow \quad C'_h \rightarrowtail C_h^{-1};\ \{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k_{g2}^0}} \rightarrowtail \varepsilon;$ In Msg_1 $C_h^{+1} \rightarrowtail \varepsilon$

Here is the explanation of the above rules.

1) Counter value 0 must be represented by z. There is no previous value for counter value 0 so no "number connection" term $\{C_h^{-1}, C_h\}_{\overrightarrow{k_{g2}^0}}$ is required in the role.
2) When a counter is positive, there must be evidence that it has a preceding nonnegative value.
3) When a counter is incremented, the variable C'_h is replaced by a new pair $[A_f, C_h^{+1}]$, where C_h^{+1} is a new nonce. The history records that the new counter value is incremented from its precedent is represented by the term $\{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k_{g2}^0}}$.
4) When a counter is kept the same, neither the new nonce nor the record of increment is needed.
5) When a counter is decremented, the variable C'_h is replaced by the preceding counter representation. The new nonce and the record of incremented counter are not needed. When $j_h = -1$, i_h must be 1 (and rule 2 applies) if T_f is a valid transition rule of M.

The rewrite rules are applied as much as possible. For example, when $i_h = 0$ and $j_h = 0$, rule 4 is applied to change C'_h to C_h, and then rule 1 is applied to change C_h to z. In the rules 4 and 5, the label "In Msg_1" is to make sure that after $C_h^{+1} \rightarrowtail \varepsilon$ is applied, $\{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k_{g2}^0}} \rightarrowtail \varepsilon$ is still applicable to Msg_2. So the order of rule application is not relevant. Appendix M shows some examples.

This is the conditions of R_f, $1 \le f \le n$

- $conds = \{ \quad 1.pre: \quad (\ C_1^{+1} \ne C_2^{+1},\ C_1^{+1} \notin mem,\ C_2^{+1} \notin mem,\ agent.name = A_f,$
  $\{B, A_f, r_f, q, k_{g1}^1, k_{g2}^1\} \subset init,\ \{A_f, B\} \subset CN,\ A_f \ne B\ );$
  $1.post: \quad (\ \{C_1^{+1},\ C_2^{+1}\} \subseteq mem\ ); \quad 2.pre: (\ \{k_{g1}^0, k_{g2}^0\} \subset init\ );\ \}$

* $AN = [SN, CN]$. SN and CN are to be instantiated in a run of Pro.
* pk and gk will be instantiated in a run of Pro according to $Pro.rsts$.
* $rsts = \{ \quad pk = Q \cup CN \cup \{r_0,\ r_{final},\ r_1,\ \cdots,\ r_n\} \cup \{z, k_{g1}^1, k_{g2}^1\};$ Let SEC be a set of terms. $SEC \cap pk = \{\}; \quad gk = \{k_{g1}^0, k_{g2}^0\} \cup SEC; \quad \forall P(P \in CA): P.init = pk \cup gk,\ P.mem^{initial} = P.init \quad \}$

We show that the resulting protocol satisfies the bounds imposed by Durgin et al. Note that no regular agent will generate any fresh nonce, so the nonces generated from regular agents are trivially bounded. All of the nonces, unbounded many, can only be generated from the attacker I. Every role has at most two action steps, so the role length is bounded by two. The message size in a run is bounded by any number equal to or greater than 15, the size of the first message of R_f, for some f, $1 \le f \le n$. Every role can only be executed by some regular agent. And every regular agent is required to do the uniqueness check of each term received that is supposed to be fresh nonce.

Before we prove the correctness of the reduction, we explain the intuition. If M can reach a final configuration $(q_{final}, \_, \_)$ starting from $(q_0, 0, 0)$, then there is a finite sequence of configurations connected by applicable rules in δ. Call this computation of M, $Comp$, which can be written as

$$(q_0, 0, 0) \longrightarrow^{t_1} (Q^1, V_1^1, V_2^1) \cdots (Q^w, V_1^w, V_2^w) \longrightarrow^{t_{w+1}} (Q^{w+1}, V_1^{w+1}, V_2^{w+1}) \ \cdots \ \longrightarrow^{t_u} (q_{final}, V_1^u, V_2^u)$$

where $w, u > 0$, $t_0, t_w, t_u \in \delta$, and u is the number of transitions in $Comp$.

After running a sequence of actions E, we say a term X is the ***encoding*** of a positive integer N, if and only if there is a sequence of terms:

$$\{z, X_1\}_{\overrightarrow{k_{g2}^0}}, \{X_1, X_2\}_{\overrightarrow{k_{g2}^0}}, \{X_2, X_3\}_{\overrightarrow{k_{g2}^0}}, \cdots, \{X_{N-2}, X_{N-1}\}_{\overrightarrow{k_{g2}^0}}, \{X_{N-1}, X\}_{\overrightarrow{k_{g2}^0}}$$

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

A symmetric key is used as the encryption key in [13] [10] [6], which is unknown to the attacker. So the attacker can neither construct an encryption, nor understand it, which could make it not practical for attacker to deploy an attack. We choose asymmetric keys k^0_{g1} and k^0_{g2} as the encryption keys, while the attacker D knows the decryption key k^1_{g1} and k^1_{g2}. So D cannot construct the encryptions, but can understand them, and can easily deploy the attack.

The encoding of 0 is the special constant z. So $0 = \underline{z}$. A positive integer is encoded by a pair $[A, X]$, where A is an agent name and X is a nonce. The encodings of numbers are connected in an encryption to show the consecutive order between numbers. $\{X, Y\}_{\overrightarrow{k^0_{g2}}}$ means that $\underline{X} = \underline{Y} - 1$.

For a sequence E, E^{-1} means a sequence that is the same as E except that the last action of E is removed. If expressed using our notations, $E^{-1} = E \uparrow^{n-1}$, where $n = len(E)$, and $n-1 = len(E^{-1})$. We will show that if the last action of an action sequence E is to receive a message msg, we will show that $Msg \in know_I(E^{-1})$.

Direction 1: Suppose M can reach a final configuration $(q_{final}, \_, \_)$ from the initial configuration, we prove that there is a run, call it run, $run \in Runs^{Pro:D}$ and $sec \in know_I(run.E)$. We prove this direction by constructing run. Pro is the protocol just described, and we will specify D and AN.

$run = [Pro,\ D,\ R,\ agents,\ AN,\ E,\ conds]$

- $Pro = [PID, roles, agents, AN, pk, gk, conds]$. The fields of Pro have been described in Def. 4. Especially, the instantiation of pk will be clear once AN is specified.
- $D = [name, init_I, know_I]$; $name = I$; $init_I = Pro.pk$ (D is an outsider); $know_I$ defined in Def. 6.
- R: A role instance r will obviously be included in R when some actions of r will be included in E when we show the proof.
- $agents = [SA, CA]$. $SA = \{\ \}$. No special agent is needed. The name of each common agent will be clear after AN is specified. The $init$ and mem fields of each common agent have been specified by Pro.
- $AN = [SN, CN]$. $SN = \{\ \}$. $CN = [a, b]$. Only two agents are enough here to instantiate the sender and receiver variables in each role.
- E: The action sequence is described below.
- $conds$: SEC is instantiated by $\{sec\}$. So sec is the only secret ground term. When we describe the run in the proof, we will justify that condition (7) of Def. 7 is satisfied, that is every message received by a regular agent can be constructed by the attacker. It is obvious that other restrictions in $conds$ defined in Def. 7 are satisfied.

Now we focus on describing $run.E$, which can be divided into three parts: the starting actions, the transition actions, and the finishing actions. We build $run.E$ by appending actions to $run.E$, starting from an empty sequence.

We need to prove that the constructed run is a run, or $run \in Runs^{Pro:D}$. We only have to show two things. The other aspects of Def. 7 are obviously satisfied.

First, given a role instance r (executed by a regular agent) of the run, the internal actions and conditions described by $r.role.conds$ should be satisfied. Particularly, all the instantiation of nonce variables should pass the uniqueness checking by the agent who executes the role instance. This is obvious since in the run all nonce variables are instantiated by nonces freshly honestly generated by the attacker, who can generate unbounded many fresh nonces and has no problem to do it.

Second, we need to show that if a message msg is received by in a regular role instance r, say at the end of an action sequence E, then $msg \in know_I(E^{-1})$. We only need to explain this aspect. A regular agent will receive a message either in a role instance of R_{server} or of a R_f, $1 \leq f \leq n$, or of R_{final}. We will show that this condition is satisfied when we add an action of message receiving to $run.E$.

The ***starting action***. At the beginning of $run.E$, we choose a role instance of R_0, call it r^0, which means that $r^0 \in run.R$, $r^0.agent.name = A_0$. A_0 is instantiated by a in $run.agents.CA$. B is instantiated by b. The first action of $run.E$ is: $\quad +(A_0 \Rightarrow s): \quad A_0, B, \{q_0, z, z\}_{\overrightarrow{k^0_{g1}}}$.

The ***transition actions***. Suppose w^{th} step in $Comp$ is $(q, V_1, V_2) \longrightarrow^t (q', V_1', V_2')$, where $0 \leq w \leq u$, and $t \in \delta$.

If according to t, $j_1 = +1$ or $j_2 = +1$, which means that $V_1 + 1 = V_1'$ or $V_2 + 1 = V_2'$, the following action of nonces generation by I is appended to $run.E$: $\#_I(c^w_1, c^w_2)$, where c^w_1 and c^w_2 are two fresh nonces. Otherwise, this action is not appended to $run.E$.

The transition rule t corresponds to a transition role in the protocol, say R_f, where $1 \leq f \leq n$, and $R_f \in Pro.roles$. A role instance of R_f is included in the run for the w^{th} transition of the 2-counter machine, call it r^w. Then $r^w \in run.R$, $r^w.role = R_f$. The 2 actions of r^w are appended to $run.E$. According to pro, the two actions have the following general form.

$-(B \Rightarrow A_f):\quad B, A_f, r_f, \{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}, \{C_1^{-1}, C_1\}_{\overrightarrow{k^0_{g2}}}, \{C_2^{-1}, C_2\}_{\overrightarrow{k^0_{g2}}}, C_1^{+1}, C_2^{+1}$

$+(A_f \Rightarrow B):\quad A_f, B, \{q', C_1', C_2'\}_{\overrightarrow{k^0_{g1}}}, \{C_1, [A_f, C_1^{+1}]\}_{\overrightarrow{k^0_{g2}}}, \{C_2, [A_f, C_2^{+1}]\}_{\overrightarrow{k^0_{g2}}}$

We have to specify for each variable in R_f its ground instantiation term. A_f and B are instantiated by a and b respectively. I impersonates B to send the first message to A_f. C_h^{+1} is instantiated by c_h^w, which is just freshly generated by I, for $h \in \{1, 2\}$. Now the variables in the above message template remaining to be instantiated in r_w are C_h, and C_h^{-1}, with $h \in \{1, 2\}$. We do not need to specify C_h', since it will be replaced by one of C_h^{-1}, C_h, or C_h^{+1} depending on the specific role R_f.

Let E^w be the prefix of $run.E$ which ends immediately before the first action of r^w. We require that the instantiation of C_h must encode V_h, denoted as $V_h = \underline{C_h}$, for $h \in \{1, 2\}$, after running E^w. q and q' are the same as the state names appearing in t. Intuitively speaking, we require $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ to encode the configuration (q, V_1, V_2). If C_h^{-1} will appear in r^w, we require that C_h^{-1} encodes $V_h - 1$.

Let Msg^w be the message received by the first action of r^w. Now we need to show that $Msg^w \in know_I(E^w)$. If $[A_f, C_h^{+1}]$ appears in Msg^w, then by the design of the protocol, it must be true that in the transition t of the 2-counter machine, $j_h = +1$. Then by the construction of the run, c_h^w is just freshly generated by I. So C_h^{+1}, which is instantiated by c_h^w is in $know_I(E^w)$. A_f is initially known by I. So $[A_f, C_h^{+1}] \in know_I(E^w)$, for $h \in \{1, 2\}$. we only need to justify that the required terms $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$, and $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}}$, if it appears in Msg^w, are included in $know_I(E^w)$.

We prove this by showing a stronger result below. It is obvious that if Lemma 1 is proven, then $Msg^w \in know_I(E^w)$ is justified.

Lemma 1: For the role instance r^w and the transition step just described, the following three facts are true.

1) There exists $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^w)$, such that $V_1 = \underline{C_1}$ and $V_2 = \underline{C_2}$.
2) For $h \in \{1, 2\}$, if $V_h > 0$, then $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}} \in know_I(E^w)$, such that after running E^w, $V_h - 1 = \underline{C_h^{-1}}$.
3) After running the two actions of r^w, we call the executed action sequence so far $E^{w'}$. For $E^{w'}$, $V_h' = \underline{C_h'}$, for $h \in \{1, 2\}$. And $\{q', C_1', C_2'\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^{w'})$.

Proof: This lemma can be proven by induction on the length of the computation of M.

Base case: The first transition step must have the form of $(q_0, 0, 0) \longrightarrow^t (q', V_1', V_2')$, for some $q' \in Q$, and $V_1', V_2' \in \{0, 1\}$. Suppose t is translated into a role R_f of the protocol, for some f, $1 \leq f \leq n$. r^1 is a role instance of R_f. Obviously $\{q_0,\ z,\ z\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^1)$ since $\{q_0, z, z\}_{\overrightarrow{k^0_{g1}}}$ is just produced by the starting actions. Since $0 = \underline{z}$, the first fact is proved. According to Pro, the terms $\{C_1^{-1}, C_1\}_{\overrightarrow{k^0_{g2}}}$ and $\{C_2^{-1}, C_2\}_{\overrightarrow{k^0_{g2}}}$ will not appear in the first message of R_f. So the second fact is trivially true. In the first transition step, either $V_h' = V_h + 1 = 1$ or $V_h' = V_h = 0$. If $V_h' = 1$, then according to the design of Pro, the second message of r^0 must include a term $\{z, [A_f, C_h^{+1}]\}_{\overrightarrow{k^0_{g2}}}$. Since $0 = \underline{z}$, after running r^1, $V_h' = 1 = \underline{C_h'}$, where $C_h' = [A_f, C_h^{+1}]$. If $V_h' = 0$, then $V_h' = 0 = \underline{C_h'}$, where $C_h' = C_h = z$. Also by the design of Pro, the q' and q in R_f are always the same as the q' and q in t. Finally, $\{q', C_1', C_2'\}_{\overrightarrow{k^0_{g1}}}$ represents (q', V_1', V_2'). The third item is proven. The base case is proven.

Induction step: Suppose for the $w - 1^{th}$ step, the lemma is true. We need to prove that the lemma is also true for the w^{th} transition step.

Fact 1. The w^{th} step is of the form $(q, V_1, V_2) \longrightarrow^t (q', V_1', V_2')$, where the configuration (q, V_1, V_2) must be just generated by the $w - 1^{th}$ step. By the induction hypothesis, for the $w - 1^{th}$ step, which just occurred in the run, the lemma is satisfied, so there must be a term $\{q, X_1, X_2\}_{\overrightarrow{k^0_{g1}}}$ generated in the second action step of r^{w-1}, where $V_1 = \underline{X_1}$ and $V_2 = \underline{X_2}$. In other words, the same instance of the term $\{q, X_1, X_2\}_{\overrightarrow{k^0_{g1}}}$ generated by the second action step of r^{w-1} is used to instantiate the term $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ in r^w. Note that $E^{w-1'} = E^w$. Obviously the needed term instance of $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ for r^w is in $know_I(E^w)$. The first fact is proven.

Fact 2. If $V_h > 0$, then according to the design of Pro, the instance of $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}}$ will appear in the first message of r^w. By Fact 1 above, $V_h = \underline{C_h}$ after running E^w. By the definition of encoding, there must be a encoding sequence of C_h, where the last term of this sequence has the form $\{X, C_h\}_{\overrightarrow{k^0_{g2}}}$, and $\underline{X} = V_h - 1$. The instance of $\{X, C_h\}_{\overrightarrow{k^0_{g2}}}$ must be included in $know_I(E^w)$, and is used to instantiate $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}}$. The second fact is proven.

Fact 3. We only need to show that $V'_h = \underline{C'_h}$, and the term $\{q', C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$ will obviously be generated in the second message of r^w and $\in know_I(E^{w'})$. 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 Pro, $C'_h = C_h^{-1}$. 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, $C'_h = C_h$. By the proven fact 1, $V'_h = V_h = \underline{C_h} = \underline{C'_h}$.
- $j_h = +1$ and $V'_h = V_h + 1$. Then according to the design of Pro, $C'_h = [A_f, C_h^{+1}]$. According to the design of the protocol and the run, in the second message of r^w, there must be a term $\{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k^0_{g2}}}$ included in the second message of r^w, where C_h^{+1} is a fresh nonce generated by the attacker. By the proven fact 1, $V_h = \underline{C_h}$. Then by the definition of encoding, after running r^w, $V'_h = V_h + 1 = \underline{[A_f, C_h^{+1}]} = \underline{C'_h}$.

■

The ***finishing actions***: A role instance of R_{final}, call it r^{final}, is included in $run.R$. The following two actions of r^{final} are appended to $run.E$.

$-(B \Rightarrow A_{final}): \quad B, A_{final}, r_{final}, \{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}} \quad ; \quad +(A_{final} \Rightarrow B): \quad A_{final}, B, Sec$

A_{final} and B are instantiated by a and b respectively. X and Y can be instantiated by any terms. Sec is instantiated by sec. Let E^{final} be the prefix of $run.E$ that ends immediately before the first action of r^{final}. In order to show that the first message of $r^{final} \in know_I(E^{final})$, we only need to show that $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^{final})$, the other terms are included in $init_I$. Since we assume the 2-counter machine can reach a final configuration $(q_{final}, \_, \_)$, the last transition step must have the form $(q, V_1, V_2) \longrightarrow^t (q_{final}, V'_1, V'_2)$. It is proven by Lemma 1 that the last transition action (the u^{th}) will produce a term $\{q_{final}, C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$, where $V'_h = \underline{C'_h}$, for $h \in \{1, 2\}$.

It is obvious that at end of run, $sec \in know_I(E)$. Direction 1 is proved.

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

Observation 1: First, every encrypted term is constructed be a regular agent. Second, two encrypted terms appearing in run with different format cannot be unified and cannot be used interchangeably. The reason for the first part is that the attacker does not know the keys k^0_{g1}, k^0_{g2} initially (although the attacker knows their inverse keys) and will never know them after any run of the protocol, since these keys will not appear encrypted or unencrypted in any message. The reason for the second part is that a configuration term $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$, and a number connection term $\{N_1, N_2\}_{\overrightarrow{k^0_{g2}}}$ are encrypted using different keys. If the same encryption key is used for both kinds of encryptions, the second part is not true. See Appendix K for more discussion.

Observation 2: A term of the form $\{X, z\}_{\overrightarrow{k^0_{g2}}}$ will never be generated in the run. If it can be generated, it must be an instance of the term $\{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k^0_{g2}}}$, for $h \in \{1, 2\}$, in a message sent by a transition role, where z instantiates the term $[A_f, C_h^{+1}]$, which is impossible. If Observation 2 is not true, the term X is effectively encoding a negative number -1. By this observation, no term can encode a negative integer.

Observation 3: Given any term X, X can appear at most once in a term of the form $\{Y, X\}_{\overrightarrow{k^0_{g2}}}$. By the construction of the protocol, $\{Y, X\}_{\overrightarrow{k^0_{g2}}}$ can only be generated as a number connection term in the second message of some transition role instance, and $X = [A_f, T]$, where A_f is the name of the agent who executes this transition role instance, and T is the term that has passed the uniqueness checking by A_f. Suppose to the contrary, there is another term $\{U, X\}_{\overrightarrow{k^0_{g2}}} = \{U, [A_f, T]\}_{\overrightarrow{k^0_{g2}}}$ generated in the run, where $U \neq Y$, then $\{U, [A_f, T]\}_{\overrightarrow{k^0_{g2}}}$ must be generated by the same agent A_f. If the two terms are generated in the same role instance executed by A_f, then T must instantiate both C_1^{+1} and C_2^{+1}. But it is impossible since according to the construction of the protocol, A_f will check that $C_1^{+1} \neq C_2^{+1}$. Suppose these two terms are generated by two different role instances executed by A_f, then T must have passed the uniqueness checking of A_f twice, but it impossible, since by the conditions of the transition roles, A_f will record T in her memory after T passed the uniqueness checking by her the first time.

Observation 4: For every term X, there can be at most one encoding sequence of X, and therefore X can only encode at most one number, especially z can only encode 0. We can see this 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 $\{Y, X\}_{\overrightarrow{k^0_{g2}}}$ appearing at the end of a encoding sequence of X. By Observation 3, the term $\{Y, X\}_{\overrightarrow{k^0_{g2}}}$ is unique, so the term Y preceding to X is fixed. By the same reasoning, the term preceding to Y in the same encoding sequence is also fixed. The same reasoning can be applied recursively backwards, until the term z, which is the starting point of the encoding sequence, and it is impossible for z to appear in the middle of the encoding sequence, by Observation 2. So the encoding sequence of X is unique.

On the other hand, it is possible that there exist two different terms of the form $\{X, Y_1\}_{\overrightarrow{k^0_{g2}}}$, and $\{X, Y_2\}_{\overrightarrow{k^0_{g2}}}$, 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, corresponding to the run of the protocol, each time a different pair $[A_f, nonce]$ 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 $\{X, Y\}_{\overrightarrow{k^0_{g2}}}$ 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 $\{X, Y\}_{\overrightarrow{k^0_{g2}}}$ 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 arbitrary run, $run \in Runs^{Pro:D}$, for every configuration term of the form $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ generated in run (q is any state), it 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 proven by induction on the sequence of configuration terms generated in the run. The proof does not distinguish whether a nonce variable is instantiated by a true nonce or a composite term. So it does not matter whether type-flaw is allowed or not.

Base case: Consider the first configuration term generated. By Observation 1, every configuration term must be generated by a regular agent. A configuration can be generated either by a role instance of R_0 or a role instance of a transition role R_f, where $1 \leq f \leq n$. The first configuration term cannot be generated by a role instance of R_f, since R_f needs to receive a configuration term in its first message, which must have been generated even earlier, impossible. So the first configuration term must be $\{q_0, z, z\}_{\overrightarrow{k^0_{g1}}}$ generated by a role instance of R_0. Obviously $\{q_0, z, z\}_{\overrightarrow{k^0_{g1}}}$ encodes the reachable configuration $(q_0, 0, 0)$ of (Q, δ) when it is generated.

Induction step: Suppose for a configuration term X generated in run, all of the earlier generated configuration terms satisfy the lemma, we need to prove that X also encodes a reachable configuration of M.

The configuration term must be generated by a role instance, say r, executed by a regular agent. There are two possibilities.

1) $r.role = R_0$. Then $X = \{q_0, z, z\}_{\overrightarrow{k^0_{g1}}}$ and it is the same as the base case. 2) $r.role = R_f$, $0 < f \leq n$. We only need to prove this case. Then X must be the term $\{q', C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$ generated in the second action step of $r.acts$. By the construction of the protocol, there must be a term $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ include in the first message received in r. By observation 1, this term cannot be instantiated by an encrypted term other than a ground configuration term which is generated earlier. By the induction hypothesis, $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ encodes a configuration (q, V_1, V_2), which is reachable from the starting configuration of M. R_f must correspond to a transition rule of the 2-counter machine, say t. Now we show that t is applicable to (q, V_1, V_2), and after applying t to (q, V_1, V_2), a configuration (q', V'_1, V'_2) can be reached, such that $\{q', C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$ encodes it.

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 $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}}$ included in the first message of R_f. This term must be instantiated by a ground number connection term, as showed by Observation 1. By Observation 2, $C_h \neq z$. Since C_h must encode either a positive number or 0, while 0 can only be encoded by z as showed by Observation 5, so $\underline{C_h} > 0$. By the induction hypothesis, $V_h = \underline{C_h}$. So $V_h > 0$.
- If $i_h = 0$, then we need to show that $V_h = 0$. By the construction of R_f, $C_h = z$. Obviously $0 = V_h = \underline{C_h}$.

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 $\{q', C_1', C_2'\}_{\overrightarrow{k^0_{g1}}}$. By the construction of R_f, the state terms 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 $C_h' = C_h$. Since $V_h = \underline{C_h}$ by the induction assumption, $V_h' = \underline{C_h'}$.
- If $j_h = +1$, then $V_h' = V_h + 1$. By the construction of R_f, there must be a term C_h^{+1} included in the first message of r. In the second message of r, a number connection term $\{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k^0_{g2}}}$ is generated, and $C_h' = [A_f, C_h^{+1}]$. Then by the definition of encoding, $\underline{C_h} + 1 = \underline{[A_f, C_h^{+1}]} = \underline{C_h'}$. Since $V_h = \underline{C_h}$, and C_h can only encode a unique number by Observation 4, $V_h' = V_h + 1 = \underline{C_h} + 1 = \underline{C_h'}$.
- If $j_h = -1$, then $V_h' = V_h - 1$. By the construction of R_f, there must be a term $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}}$ included in the first message of r. $C_h' = C_h^{-1}$. By the induction hypothesis, $V_h = \underline{C_h}$, and $V_h > 0$. By Observation 3, $C_h \neq z$, since z can only encode 0. Then by the definition of encoding sequence, there exists a term $\{Y, C_h\}_{\overrightarrow{k^0_{g2}}}$ in the encoding sequence of C_h, which has appeared in the run and known by I, where $V_h - 1 = \underline{C_h} - 1 = \underline{Y}$.
  By Observation 3, the term $\{Y, C_h\}_{\overrightarrow{k^0_{g2}}}$ is unique, Then, the attacker can only use $\{Y, C_h\}_{\overrightarrow{k^0_{g2}}}$ as the term $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}}$. So $V_h' = V_h - 1 = \underline{C_h^{-1}} = \underline{C_h'}$.

So the induction step is proven. ■

Now we finish the proof of direction 2. We assume that $sec \in know_I(run.E)$. Then sec must have been sent by a regular agent, since $sec \notin init_I$. A regular agent will generate sec only in the second message of a role instance, call if r^{final}, of R_{final}. r^{final} needs to receive a term of the form $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$ in its first message, where X and Y are some arbitrary terms. By Observation 1, and by free term algebra assumption, $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$ must be a configuration term. By Lemma 2, $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$ must encode a configuration (q_{final}, V_1, V_2), which is a reachable configuration to the 2-counter machine. Direction 2 is proved.

To translate a description of a 2-counter machine to the corresponding protocol Pro can always be done in finite amount of time, since Pro is always constructed by finitely many symbols. Theorem 1 is proved.

■

Remark. The proof of Direction 1 depends on the capability of the attacker to generate unbounded number of distinct nonces. The attack constructed in direction 1 of the above proof the attack only provide fresh atomic terms to instantiate the terms of C_h^{+1}, $h \in \{1, 2\}$. Direction 1 can also be proved by constructing a run while the attacker uses composite terms as nonces. However, that capability (assumption (v)) is still indispensable to prove direction 1. The reason is that the size of term instances are assumed bounded in a run. Suppose there are only bounded number of nonces generated from the attacker, since the regular agents can only generate bounded number of nonces, there are bounded number of atomic terms, and then the attacker cannot get unbounded number of distinct composite terms (or "fake" nonces) with bounded size, therefore the computation of an arbitrary M cannot be simulated, which can have unbounded number of computation steps. In fact, without assumption (v), the problem is proven decidable by [10] (by the reason of bounded number of ground facts in the proof system).

The proof can be enhanced to cover a stronger consideration, which could be a more restricted interpretation of the open problem.

In the proof of theorem 1, when an agent, say A, receives the first message in a transition role, A does not check the uniqueness of the variables C_h and C_h^{-1}, for $h \in \{1, 2\}$, neither does A check whether C_h and C_h^{-1} belong to the initial knowledge of A. Theorem 1 deals with a general consideration such that for the variables received by A which are not created by A and may not be known by A initially, A will check the uniqueness of some of them, but will not care about the others. In other words, uniqueness check by A is allowed but not required. This situation should be consistent to the description of table I.

However we have noticed that in the protocols appearing in the proofs of [10] [11] where the open problem is mentioned, there are only two types of variables appearing a protocol run: agent names, which must belong to the initial knowledge of agents, or the nonces (created by regular agents). A stronger consideration is that an agent A will treat all of the variables, which are not names, received from other agents uniformly as fresh nonces, i.e., A will always check their uniqueness, and the variables that A does not care about such as C_h and C_h^{-1} as in the general consideration of theorem 1 are not allowed. Theorem 2 solves the open problem with the this stronger consideration.

Theorem 2: Suppose for a variable, say X, appearing in a role executed by a regular agent A, and the value of X is not determined by A (X first appears in the role in a message received by A), A must do one of the two kinds internal actions to X upon receiving it as follows. 1) A will make sure that $X \in A.init$, e.g., X is an agent name.; Or 2) A will check that $X \notin A.mem$, e.g., X should be treated a nonce freshly generated by some agent other than A. With this consideration the open problem described in theorem 1 is still undecidable.

Proof: The idea is to create fresh copies of nonces which can encode some counter value, and then generate fresh copies of configuration terms from the processed configuration terms where the nonce variables encoding counter values are replaced by fresh and equivalent nonces. During the computation, the attacker always provides fresh copies of terms to the agents who execute the transition roles. So the run of the protocol still simulates the computation of the 2-counter machine while no agent will see the same nonce twice.

Now each transition role, say R_f, $1 \leq f \leq n$, executed by A_f, will require in $1.pre$ (the precondition of receiving message 1) that $\{C_h^{-1}, C_h\} \cap A.mem = \emptyset$, $C_h^{-1} \neq C_h$, and in $1.post$ A will requies that $\{C_h^{-1}, C_h\} \subseteq A.mem$, for $h = 1$ or/and $h = 2$ (A remembers them).

The following two roles R_{s1}, R_{s2} (s stands for stronger) are used to generate fresh and equivalent copies of terms. They are added to $Pro.roles$.

$R_{s1} = [RID,\ agent,\ vars,\ acts,\ conds]$

- $RID = r_{s1}$; $agent = [name,\ init,\ mem]$; $vars = \{A_{s1}, V_1, V_2, X, Y, B\}$.
- $acts =$ $1. - (B \Rightarrow A_{s1}) : B, A_{s1}, r_{s1}, \{V_1, X\}_{\overrightarrow{k_{g2}^0}}, \{V_1, V_2\}_{\overrightarrow{k_{g3}^0}}, Y$
  $2. + (A_{s1} \Rightarrow B) : A_{s1}, B, \{V_2, [A_{s1}, Y]\}_{\overrightarrow{k_{g2}^0}}, \{X, [A_{s1}, Y]\}_{\overrightarrow{k_{g3}^0}}$
- $conds = \{$ $1.pre :$ $(A_{s1}, r_{s1}, B, k_{g2}^1, k_{g3}^1, z\} \subseteq init,$ $name = A_{s1},$
  $\{A_{s1}, B\} \subset CN, A_{s1} \neq B, \{X, Y, V_1, V_2\} \cap mem = \emptyset,$
  $X \neq Y,\ X \neq V_1,\ X \neq V_2,\ Y \neq V_1,\ Y \neq V_2,$
  $V_1 = z\ or\ V_1 \notin mem,\ \ V_2 = z\ or\ V_2 \notin mem,\ \ V_1 = V_2 = z\ or\ V_1 \neq V_2)$
  $1.pos :$ $(\{V_1, V_2, X, Y\} \subseteq mem);$
  $2.pre :$ $(\ k_{g2}^0 \in init,\ \ k_{g3}^0 \in init)\ \}$

The term $\{X, [A, Y]\}_{\overrightarrow{k_{g3}^0}}$ is to show the equivalence between the term X and $[A, Y]$. We call the a term of the form $\{U, V\}_{\overrightarrow{k_{g3}^0}}$ as ***equivalence term*** where U and V are two arbitrary terms. Every regular agent knows the key pair k_{g3}^0 and k_{g3}^1, while the attacker only knows the key k_{g3}^1. V_h, $h \in \{1, 2\}$, is required to either be z or some term A has never seen before. In the literature we do not see many cases of internal action of logical or, although it is trivial to implement it. If we do not allow an agent to do logical or, we can design four different roles depending on V_h is z or not, $h \in \{1, 2\}$, and the proof still works. Y is provided by the attacker. That attacker has to generate unbounded number of nonces to instantiate Y in oubounded number of role instances of R_{s1} in order to generate unbounded number of copies (with bounded size) of terms.

The role R_{s2} creates a copy of a configuration term by using the equivalent terms of C_h, $h \in \{1, 2\}$.

$R_{s2} = [RID,\ agent,\ vars,\ acts,\ conds]$

- $RID = r_{s2}$; $agent = [name,\ init,\ mem]$; $vars = \{A_{s2}, G, C_1, C_2, X, Y, B\}$.
- $acts =$ $1. - (B \Rightarrow A_{s2}) : B, A_{s2}, r_{s2}, \{G, C_1, C_2\}_{\overrightarrow{k_{g1}^0}}, \{C_1, X\}_{\overrightarrow{k_{g3}^0}}, \{C_2, Y\}_{\overrightarrow{k_{g3}^0}}$
  $2. + (A_{s2} \Rightarrow B) : A_{s2}, B, \{G, X, Y\}_{\overrightarrow{k_{g1}^0}}$
- $conds =$ $\{$ $1.pre :($ $\{A_{s2}, r_{s2}, B, k_{g1}^1, k_{g3}^1, z\} \subseteq init, name = A_{s2},$
  $\{A_{s2}, B\} \subset CN, A_{s2} \neq B, \{X, Y\} \cap mem = \emptyset, G \in Q,$
  $C_1 = z\ or\ (C_1 \notin mem, C_1 \neq C_2, C_1 \neq X, C_1 \neq Y, X \neq Y),$
  $C_2 = z\ or\ (C_2 \notin mem, C_2 \neq C_1, C_2 \neq X, C_2 \neq Y, X \neq Y)\ )$;
  $1.pos : (\{X, Y, C_1, C_2\} \subset mem);$
  $2.pre : (\ k_{g1}^0 \in init)\ \}$

Again, we allow the logical or in the internal action to treat C_1 and C_2. Otherwise we could design four different roles depending on the choices of values of C_1 and C_2.

In addition, we adjust R_0 so that the first message of R_0 will include one more term of the form $\{z, z\}_{\overrightarrow{k_{g3}^0}}$. This is to show that z is equivalent to itself. This term is the first equivalence term generated in a run and is needed to start the process of copying encoding sequences by role instances of R_{s1}.

In addition to the five observations presented in the proof of theorem 1, we can have the following observations.

Observation 6: The equivalence term of the form $\{z, X\}_{\overrightarrow{k_{g3}^{0}}}$ where $X \neq z$ will never be generated in the run. In other words, the only term of this form that can appear in the run is $\{z, z\}_{\overrightarrow{k_{g3}^{0}}}$.

Observation 7: Whenever an equivalence term of the form $\{U, V\}_{\overrightarrow{k_{g3}^{0}}}$ is produced in the run, both U and V encodes the same number, i.e., they have some encoding sequences, and $\underline{U} = \underline{V}$. This can be proven by induction. The first equivalence term generated in the run must be $\{z, z\}_{\overrightarrow{k_{g3}^{0}}}$ by a role instance of R_0 (R_0 of theorem 1 is adjusted here for theorem 2). Obviously z encodes the number 0. For the induction case, suppose for all equivalence term produced in the run so far the observation is true. The next generated equivalence term must be of the form $\{X, [A_{s1}, Y]\}_{\overrightarrow{k_{g3}^{0}}}$ generated in the second message (Msg_2) of a role instance of R_{s1}. Since $\{V_1, V_2\}_{\overrightarrow{k_{g3}^{0}}}$ must appear in the first message (Msg_1) of R_{s1}, V_1 and V_2 must encode the same number by the induction hypothesis. Since $\{V_1, X\}_{\overrightarrow{k_{g2}^{0}}}$ appears in Msg_1, X must encode the number $\underline{V_1}+1$. Since in Msg_2 $\{V_2, [A_{s1}, Y]\}_{\overrightarrow{k_{g2}^{0}}}$ appears, $[A_{s1}, Y]$ must encode $\underline{V_2} + 1$. So $\underline{X} = \underline{[A_{s1}, Y]}$, where $U = X$ and $V = [A_{s1}, Y]$.

Observation 8: Whenever a number connection term of the form $\{U, V\}_{\overrightarrow{k_{g2}^{0}}}$ is produced in the run, both U and V encode some number, i.e., each of U and V has its own encoding sequence, and $\underline{V} = \underline{U}+1$. This observation can be proven by induction, similar to the proof of Observation 7. Note that when the term $\{V_2, [A_{s1}, Y]\}_{\overrightarrow{k_{g2}^{0}}}$ is generated by a role instance of R_{s1}, V_2 must encode some number since $\{V_1, V_2\}_{\overrightarrow{k_{g3}^{0}}}$ appears in Msg_1 and Observation 7 guarantees that V_2 has an encoding sequence. So $\underline{[A_{s1}, Y]} = \underline{V_2} + 1$.

Observation 7 and 8 can show that a term cannot be copied unless all of its precedents in its encoding sequence are copied. So the attacker cannot use a term t' as a new equivalent copy of a term t while t' does not encode the same number that t encodes.

Observation 9: An execution of a role instance of R_{s1} could produce a new term $[A_{s1}, Y]$ which encodes some number, say u, and $u = \underline{[A_{s1}, Y]}$. However u will never be larger than the largest encoded number (the largest counter value) reached so far in the run. The reason is that the new number-encoding term $[A_{s1}, Y]$ must encode the same number as V_2 does, by Observation 7, and the encoding sequence of V_2 has been produced earlier, and $\underline{V_2}$ represents a counter value that has already been reached in the run of M.

By Observation 9, and by the fact that R_{s2} does not create new number connection terms, it follows that the counter value is increased or decreased solely by the transition roles (R_f, $1 \leq f \leq n$). In other words the two additional roles R_{s1} and R_{s2} do not interfere with the computation, they are only used to make copies of the computation results. It is obvious that the five observations (1 to 5) described in direction 2 of the proof of theorem 1 are still true. Then direction 2 can be proven exactly the same as in theorem 1.

To prove direction 1 is to show that when M can reach a final configuration, there is an attack to Pro such that no agent will accept a nonce which is supposed to be freshly generated by others and the agent has seen it before. If we assume infinite many different agents participating in the run, in order to construct the attack, we can always choose a new agent, who has not participated in the run yet, to execute the next role instance, and then every agent will not see the same term twice since he only participates in one role instance in the run, and then the strong condition is trivially satisfied. The more interesting question is whether there is such an attack where only bounded many agents are allowed.

We can organize the agents in the attack in three groups by different taskes as follows. 1) Let a bounded number of agents perform the unbounded number of role instances of R_{s1} to produce unbounded number of new copies of number-encoding terms, i.e., to copy the whole encoding sequence. We will show how this can be done soon. 2) Let two agents b_1 and b_2 execute the unbounded number of role instances of R_{s2} to generate unbounded many new copies of configuration terms. 3) Let a single agent a execute all of the role instances of R_0, R_{final} and R_f, $1 \leq f \leq n$.

The attack can be described as follows. Same as the attack described in the proof of direction 1 of theorem 1, every reachable configuration in the computation of M corresponds to a configuration term generated in the run of the protocol. In theorem 1, the configuration term just generated is used as an input to a transition role instance to produce the next configuration term. Here the difference is that a new equivalent copy of the configuration term just generated is used as the input instead.

The first configuration term is $\{q, z, z\}_{\overrightarrow{k_{g1}^{0}}}$ generated by a role instance of R_0 executed by a. This configuration term is used as an input to start producing a sequence of "next" configuration terms produced by the role instances of R_f, $1 \leq f \leq n$, executed by a. Whenever in the next configuration term $\{q, C_1, C_2\}_{\overrightarrow{k_{g1}^{0}}}$, for some $q \in Q$, C_1

or C_2 is not z (corresponding to a positive counter value of M), then the following process is used to produce an equivalent new copy of $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$.

Suppose u is the largest positive number one counter has reached so far in the computation of M, which must corresponds to some term, call it C^u, produced in the run of Pro and $\underline{C^u} = u$. C^u has an encoding sequence consisting a chain of number encoding terms like $z, X_1, X_2, \cdots, C^u$. A sequence of role instances of R_{s1}, executed by agents of group 1), are used to generate a new copy the encoding sequence, where the sequence of number encoding terms are $z, X'_1, X'_2, \cdots, C^{u'}$, and the terms along the two sequences are equivalent one-to-one. Which means that there are equivalence terms of $\{z, z\}_{\overrightarrow{k^0_{g3}}}$, $\{X_1, X'_1\}_{\overrightarrow{k^0_{g3}}}$, $\{X_2, X'_2\}_{\overrightarrow{k^0_{g3}}}$, $\cdots$, $\{C^u, C^{u'}\}_{\overrightarrow{k^0_{g3}}}$ generated in the run. We can view the new copy of the encoding sequence in a chart as an upper layer built from the current sequence, the lower layer, with equivalent length. Then choose the term from the new copy of the encoding sequence, say X, which is equivalent to C_h, i.e., $\{C_h, X\}_{\overrightarrow{k^0_{g3}}}$ is produced. Do the same for both counters, if both counters are positive.

A role instance of R_{s2} is executed by agents b_1 or b_2 of group 2), where Msg_1 contains the input terms of $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$, $\{C_1, X\}_{\overrightarrow{k^0_{g3}}}$, and $\{C_2, Y\}_{\overrightarrow{k^0_{g3}}}$. Note that if C_1 is z, then $\{z, z\}_{\overrightarrow{k^0_{g3}}}$ will replace $\{C_1, X\}_{\overrightarrow{k^0_{g3}}}$, and the same for C_2. Then a new configuration term $\{q, X, Y\}_{\overrightarrow{k^0_{g1}}}$ is produced. It is guaranteed that b_1 or b_2 have not seen X and Y yet, since X and Y are produced by the agents of group 1). But if b_1 produced the copy of the configuration term previous to $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$, which is used as an input to of an role instance of R_f to produce $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$, b_1 may have seen C_1 or C_2 already ($C_1 \in b_1.mem$), in case some counter does not change. The solution is to let b_1 and b_2 to execute the role instances of R_{s1} in turn to make copies of configuration terms.

More details of idea are here. If we name the sequences of role instances of R_{s2} in the attack as w_1, $w_2 \cdots$, and let b_1 and b_2 to execute them alternatively. Suppose b_1 executes w_i, $i \leq 1$, then b_1 sees and checks (in the most complex situation, when both counters are positive) the uniqueness of the terms C^i_1, C^i_2, X^i, Y^i, $C_1^{-1^i}$, and $C_2^{-1^i}$. b_2 executes w_{i+1}, and b_2 will see and check the uniqueness of C^{i+1}_1, C^{i+1}_2, X^{i+1}, Y^{i+1}, $C_1^{-1^{i+1}}$, and $C_2^{-1^{i+1}}$, where C^{i+1}_1 could be $C_1^{-1^i}$ or X^i, similarly for C^{i+1}_2. X^i and Y^i are the fresh terms produced by agents of group 1) who execute role instances of R_{s1} and b_2 has never seen them before. Since w_{i+1} is not executed by b_1, b_1 does not see the same fresh term twice. Then w_{i+2} is executed by b_1, and b_1 will see and check the uniqueness of C^{i+2}_1, C^{i+2}_2, X^{i+2}, Y^{i+2}, $C_1^{-1^{i+2}}$, and $C_2^{-1^{i+2}}$, which are different from the terms b_1 has seen in w_1. The reasoning continues and is guaranteed that b_1 and b_2 will not see the same fresh term twice while copies of configuration terms can be generated unbounded times. The idea of executing role instances in turn is extended to organize the behavior of the agents in group 1).

After a has produced the configuration term $\{q, C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$ by executing some role instance of R_f, $1 \leq f \leq n$, the encoding sequence of the term, which encodes the largest number of counter 1 and is generated for the previous configuration term, is made, same for counter 2. A new copy of $\{q, C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$ is made, say it is $\{q, U, V\}_{\overrightarrow{k^0_{g1}}}$. a executes the next role instance of some transition role, where the first message include the new copy $\{q, U, V\}_{\overrightarrow{k^0_{g1}}}$, and the number connection terms $\{U^{-1}, U\}_{\overrightarrow{k^0_{g2}}}$, and/or $\{V^{-1}, V\}_{\overrightarrow{k^0_{g2}}}$ chosen from the encoding sequence of U and V, if both counters are, or one of them is, positive. It is guaranteed that a has not seen U, X, V, U^{-1} and V^{-1} yet since they are produced by the agents of group 1). The process continues until the final configuration term is produced and the secret term is leaked.

Figure 1 represents the copying process for one counter, say counter 1 (for counter 2 it is the same), in a beginning section of a run. Roles in figure 1 are labeled from 1 to 7, representing the first 7 steps of computation of M corresponding to 7 times of executing some transition role instances (call them ri_1 to ri_7) by a. Every column is marked with a number from 0 to 4 representing the number which the terms in the column encode. For example, the terms in column 3, from $C_{3.3}$ to $C_{7.3}$ all encode 3. The term marked with $*$ represents C_1 in the configuration term $t = \{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ which is used as an input term for a transition role instance of each step from 1 to 7. We call the output configuration term by a transition role instance t' which has the form of $\{q', C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$. At step 1, counter 1 is increased from 0 to 1, $C_{1.1}$ appears as C'_1 in t' produced by ri_1. now the largest value of counter 1 is 1. One role instances of R_{s1} is executed to copy the encoding sequence of $C_{1.1}$. $C_{2.1}$ is equivalent to $C_{1.1}$, and is used as an input term to a role instance of R_{s_2} to make a new copy t' of step 1, which is t for step 2, where C_1 in t is $C_{2.1}$. At step 2, counter 1 is increased and $C_{2.2}$ is the C'_1 in t' of step 2. Then two role instances of R_{s1} are executed to make a copy of the encoding sequence of $C_{2.2}$. And $C_{3.2}$ is the one equivalent to $C_{2.2}$, and is used

$$\begin{array}{llllll}
\cdots & & & & & \\
\uparrow & & & & & \\
7 & z & C_{7.1} & C_{7.2} & C_{7.3}* & C_{7.4} \\
6 & z & C_{6.1} & C_{6.2} & C_{6.3} & C_{6.4}* \\
5 & z & C_{5.1} & C_{5.2} & C_{5.3} & C_{5.4}* \\
4 & z & C_{4.1} & C_{4.2} & C_{4.3}* & C_{4.4} \\
3 & z & C_{3.1} & C_{3.2}* & C_{3.3} & \\
2 & z & C_{2.1}* & C_{2.2} & & \\
1 & z* & C_{1.1} & & & \\
 & 0 & 1 & 2 & 3 & 4 \qquad \rightarrow \cdots
\end{array}$$

Fig. 1. The copying process for one counter in the proof of direction 1 of Theorem 2

as an input term to a role instance of R_{s2} to make a new copy of t' of step 2, which is the t for step 3 where C_1 is $C_{3.2}$. In step 3 and step 4 counter 1 keeps incrementing. $C_{3.3}$ is the C'_1 of t' of step 3, and its equivalent new copy is $C_{4.3}$, which is used as the C_1 of t for step 4. Then $C_{4.4}$ is the C'_1 of t' of step 4. $C_{4.4}$ is copied to $C_{5.4}$ by four role instances of R_{s1}. $C_{5.4}$ is used as an input term to generate the copy of t' of step 4, and the copy is used as t for step 5. At step 5 counter 1 stays the same, so $C_{5.4}$ appears as C'_1 of t' of step 5. Now the largest counter value is 4. Four role instances of R_{s1} are executed to make a copy of the encoding sequence of $C_{5.4}$ where $C_{6.4}$ is the one equivalent to $C_{5.4}$. $C_{6.4}$ is used as an input of a role instance of R_{s2} which makes a copy of the t' of step 5, which is t of step 6, where C_1 is $C_{6.4}$. At step 6 counter 1 decrements and $C_{6.3}$ becomes the C'_1 of t' of step 6. Before step 7, the current largest counter value is 4. Although the current value is 3, the encoding sequence of $C_{6.4}$ is made with length 4 where $C_{7.3}$ is the one equivalent to $C_{6.3}$. Then $C_{7.3}$ is used in a role instance of R_{s2} to generate a copy of t' of step 6, which is t of step 7 where C_1 is $C_{7.3}$. The process continues for the remaining of the run of pro corresponding to the computation of M until the final configuration is reached and the secret term is leaked.

The remaining question is whether the number agents of group 1) can be bounded for generating the unbounded copies of encoding sequences, considering that every agent should not see the same nonce twice. We illustrate the idea by figure 1.

Group 1) is divided to two subgroups, 1.1) works for counter 1, and 1.2) works for counter 2. 1.1) is further divided to two subgroups 1.1.1) and 1.1.2). The agents of group 1.1.1) do the copying tasks of the odd rows and the group 1.1.2) do the copying jobs of the even rows (see figure 1. Then it is guaranteed that the same agents who do the copying jobs of row 1 can do the copying jobs of row 3, since they must have not seen the terms of row 2 yet. So group 1.1.1) can do the copying tasks for all odd rows. Group 1.1.1) may only have two agents, say d_{1111} and d_{1112}, to do the copying tasks of row 1 alternatively. For example when d_{1112} executes a role instance of R_{s1}, in Msg_1, V_1 and X are produced in the lower row by agents of 1.1), so d_{1111} has not seen them yet. V_2 is produced in the same row but by the other agent d_{1112}, so d_{1111} has not seen it yet. Y is a new term provided by the attacker which has not seen by d_{1111}. Similarly group 1.1.2) can have another two different agents to do the copying tasks of the even rows. Group 1.1) totally has four agents. Similarly in group 1.2) there are four agents to do the copying tasks for counter 2. So totally there are eight agents in group 1).

Totally in the three groups there are eleven agents, a small number. With more roles introduced to implement more properties of the equivalence relationship between terms, the number of agents in group 1 can be reduced, e.g., the same four agents could cover both counters, and then group 1 may only have four agents. The attack is found while the strong condition is satisfied, which finishes the proof of direction 1, and the proof of theorem 2. ■

IV. Summary

We solve the open problem of Durgin et al. [10] [11] using a direct reduction scheme from the halting problem of 2-counter machines. We give a rigorous proof of correctness and carefully consider the assumptions and scenarios of the problem. This proof method is applicable beyond the above result. For example, with extended modeling and adaptation of the above reduction, we have proved other new and important undecidability results, including undecidability of checking secrecy for matching RO protocols with an attacker who is an insider.

References

[1] 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.

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

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

[4] 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.

[5] M. Rusinowitch and M. Turuani, "Protocol insecurity with finite number of sessions is NP-complete." in *CSFW*, 2001, pp. 174–.

[6] ——, "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.

[7] 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.

[8] H. Comon and V. Cortier, "Tree automata with one memory set constraints and cryptographic protocols," *Theor. Comput. Sci.*, vol. 331, no. 1, pp. 143–214, 2005.

[9] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov, "Undecidability of bounded security protocols," in *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy*, N. Heintze and E. Clarke, Eds., july 1999.

[10] 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.

[11] N. 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

[12] 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.

[13] 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

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

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

[16] 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.

[17] 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.

[18] J. Clark and J. Jacob, "A survey of authentication protocol literature: Version 1.0," Tech. Rep., 1997. [Online]. Available: http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz

[19] 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

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

[21] 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.

[22] D. J. Otway and O. Rees, "Efficient and timely mutual authentication," *Operating Systems Review*, vol. 21, no. 1, pp. 8–10, 1987.

Appendix

A. *Understanding the open problem*

Table I shows the complexity results provided by [10] and [11] (Page 282 of [10] and Page 47 of [11]), with more explanations added by us. In [10] the authors focus on bounded security protocols, which means the scenario are bounded in two aspects. First role length is bounded (Pages 250 and 261, [10]) which means that the number of messages appearing in any role template of the protocol is bounded. Second, the size of any message instance (the number of ground atomic terms appearing in the message instance, which is a ground term) in a run is bounded (Detailed discussion in Appendix G) , which implies that the size of the every message template in the protocol is also bounded. We think the bound on the role length is not essential, since without it all the complexity results of [10] [11] are still true.

Table I shows the following results. Assume the protocols has bounded role length, and consider only the runs of a protocol that has bounded size of message instances. Left column: when the number of role instances appearing in a run of the protocol is bounded, checking secrecy is NP-complete. Right column: when the number of role instances in a run is unbounded, and the total number of nonces generated from regular agents in a run is unbounded, checking secrecy is undecidable. Middle column: Suppose the number of role instances in a run is unbounded,

TABLE I
THE COMPLEXITY TABLE PROVIDED BY [10] OF CHECKING SECRECY, WITH ADDED EXPLANATION BY US. IN THIS TABLE, ROLE LENGTH AND SIZE OF MESSAGE INSTANCES IN A RUN ARE ASSUMED BOUNDED. $\exists$ MEANS NONCE INSTANCE, $\neq$ MEANS REGULAR AGENTS REQUIRE UNIQUENESS CHECK ON NONCE INSTANCES, AND DISEQUALITY TEST IS ALLOWED, WHILE $=$ MEANS THAT UNIQUENESS CHECK AND DISEQUALITY TEST ARE NOT ALLOWED.

		Bounded role instance num.	Unbounded role instance num.	
			Bounded total $\exists$ from regular agents	Unbounded total $\exists$ from regular agents
I with unbounded $\exists$	$\neq$	NPC	???	Undec.
	$=$	NPC	DEXPC	Undec.
I with bounded $\exists$	$\neq$	NPC	DEXPC	Undec.
	$=$	NPC	DEXPC	Undec.

checking secrecy is DEXP-complete if I can only generate bounded number of nonces, or I can generate unbounded number of nonces while agents do not have nonce uniqueness check.

The following sentence describing the open problem appears on Page 48 of [11]:

"The series of ??? in the box at the top of column two indicates an unresolved question for the upper bound in the case of unbounded roles, bounded protocol existentials, and unbounded intruder existentials, when disequality tests are allowed."

To understand the open problem (and the above sentence) is to understand exactly the various assumed conditions and bounds and notions for the table.

B. Understanding $\exists$ and $\neq$

$\exists$ is used in [10] as a notation to show that a term is a nonce freshly generated, and its value is different from any other term which has already appeared in the run of the protocol so far. In our paper we represent the internal action of nonce generation by the notation $\#()$ to avoid possible confusion.

The equality symbol $=$ means the internal action of an agent in a run to check the equality of two terms with the same name in the protocol. $=$ is not an explicit notation used in MSR, as explained in [10] Page 259:

"We do not need to add a condition to test for equality, because it is expressible by matching the names of the variables in the terms."

The disequality symbol $\neq$ can represent that some term (nonce) is forced to be unique and be different from any other term which has appeared in the run or the protocol so far. In Table I and the original one in [10], the $\neq$ (checking uniqueness) is only applied to nonces. Here is the explanation of the original table in [10], Page 282:

"These rows are further subdivided into the cases where the roles can perform disequality tests which would allow them to determine whether two fresh values are different from each other. The $\neq$ row allows both equality and disequality tests, while the = row allows only equality tests. In a protocol, a test for disequality on a nonce would mean the protocol compares a supposedly fresh nonce it receives against all the other nonces it has received, to make sure it is actually fresh. If disequality is not allowed, then this test is not performed."

Here is the original explanation of $\exists$ in the multiset rewriting system with disequality ($MSR_{\neq}$) in [10], Page 290:

"Computationally, the meaning of $\exists$ in $MSR_{\neq}$ is clear - each value generated by an $\exists$ is unequal to all others. We have not investigated the correspondence between logic and $MSR_{\neq}$."

We think the uniqueness of a term X is not easy to be expressed by just using $\neq$, which is a binary predicate explicitly used in MSR as showed in some examples in Page 259 of [10]. What is meant by Durgin et al. is that X has to be compared with all other terms appeared so far in the run. To express the uniqueness of a term X in MSR, some quantifier like $\forall$ may be needed, in addition to describing the set which contains all the terms recorded by an agent. We think a better way is that the uniqueness check could be expressed by other means, while $\neq$ can be kept as a binary predicate. In other words, for the open problem "disequality test" really means allowing an agent to do disequality test on two terms and requiring uniqueness check on nonces.

C. Understanding Uniqueness Check

The only practical way to implement the uniqueness check of nonces is that every agent maintains a memory recording all nonces which the agent has seen so far in the run, in addition to her initial knowledge.. Then whenever a term which is supposed to be a fresh nonce comes, the agent checks that it is different from all of the recorded terms in her memory. There are three interpretations of the sentence in Page 282 of [10], just quoted in the above section: "$\cdots$ a test for disequality on a nonce would mean the protocol compares $\cdots$".

X in a run of a protocol is that an agent maintains a memory recording or of the atomic terms it has seen so far in the run, and then when X is received by the agent, she checks that X is different from all terms in the memory.

The first interpretation. For every possible run of the protocol, after a term is accepted by a regular agent once as a freshly generated term, the term cannot be accepted again by any regular agent as a new fresh term. In other words, every ground term can only be accepted as a fresh one at most once in the whole run (the ground term accepted as fresh by the same agents twice, or by two different agents each once, counts 2). By saying "an agent accepts a term as a fresh one" we mean that the agent will do the uniqueness check of the term upon receiving it. Since we assume different agents do not share memory (a practical assumption for the distributed nature of the agents), it should be very hard, even unlikely, for two agents to not accept the same fresh nonce, although both agents can only accept the same nonce once. It seems that in order to implement the first interpretation, only one fixed agent will do all of the uniqueness check of the supposed fresh terms, and the other agents do not care about the uniqueness of fresh terms.

The second interpretation. Due to the rareness of the protocol that can satisfy the first interpretation, we can say that the analysis only consider the runs of a protocol such that a nonce can be accepted as fresh at most once.

The third interpretation. The word "protocol" should be replaced by "principal". A protocol does not receive nonces, only a principal (equivalently an agent) receives nonces. In this case the open problem means that when a term, which is supposed to be freshly generated nonce, is received and accepted by a regular agent, it can be instantiated by a recycled term (in the sense that the term has already been accepted by another agent as a fresh term once in the run so far), provided that it is not received by or known by this agent before. It means that each agent can only accept a nonce as a fresh one at most once, but the same nonce could be accepted by different agents.

We believe that the third interpretation is more general and practical. In the proof of Theorem 1, we construct a protocol such that the third interpretation is guaranteed in any run of the protocol. However, by fixing the name of the agent of every role template with the same constant agent name, the proof will also work to show the undecidability with the first and second interpretations.

D. Uniqueness Check is Necessary for the Undecidability of the Open Problem

The first necessary condition for the undecidability of the open problem is the capability of the attacker to generate unbounded many nonces. The second necessary condition is the uniqueness check by a regular agent. Without the second condition, it is proved decidable by [10] [11]. It is interesting to see how the second condition is reflected in our proof. Here we give an example to show that without the uniqueness check, the proof will not go through, by showing that the protocol will generate a final configuration term while it is impossible for the corresponding 2-counter machine to reach q_{final}.

Let a 2-counter machine be $M = (Q, \delta)$. Let $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_{final}\}$. Let $\delta = \{\quad [q_0, 0, 0] \rightarrow [q_1, +1, 0]$; $[q_1, 1, 0] \rightarrow [q_2, +1, 0]$; $[q_2, 1, 0] \rightarrow [q_3, +1, 0]$; $[q_3, 1, 0] \rightarrow [q_4, -1, 0]$; $[q_4, 1, 0] \rightarrow [q_5, -1, 0]$; $[q_5, 1, 0] \rightarrow [q_6, -1, 0]$; $[q_6, 1, 0] \rightarrow [q_{final}, -1, 0] \quad \}$.

The second counter will always be 0. It is obvious that the 2-counter machine cannot reach q_{final}, since the only way to reach q_{final} is to increment the first counter 3 times, starting from 0, and then decrement it 4 times. q_{final} can only be reached from q_6, by applying the last transition rule. Whenever M reaches q_6, the first counter is 0, and the last transition rule cannot be applied, since it requires the first counter to be positive.

If an agent does not do the uniqueness check on nonces, even though the agent may require the nonces appearing in the same role instance to be different, there is a run of the corresponding constructed protocol, where the final configuration term can be generated. The attacker can generate two different fresh nonces x and y. Let a single agent a execute all of the transition role instances in the run. The first time the counter needs to be incremented, the attacker tells a that x is the new nonce. The second time, the new nonce is y. And the third time, it is x again. So there is no problem to generate the configuration term containing q_3. And the number connection terms generated in the run are $\{z, [a,x]\}_{\overrightarrow{k^0_{g2}}}$, $\{[a,x],[a,y]\}_{\overrightarrow{k^0_{g2}}}$, $\{[a,y],[a,x]\}_{\overrightarrow{k^0_{g2}}}$. This can happen since a does not remember nonces received in previous role instances, although a may require that $x \neq y$ in every role instance. The Observation 3 is violated, and so is Observation 4. Whenever the first counter needs to be decremented from $[a,x]$, the attacker will use the number connection term $\{[a,y],[a,x]\}_{\overrightarrow{k^0_{g2}}}$, to show that $\underline{[a,y]} = \underline{[a,x]} - 1$, and discard $\{z,[a,x]\}_{\overrightarrow{k^0_{g2}}}$. Then starting from q_3, in the run of the protocol, the first counter can be decremented 4 times. from $[a,x]$ to $[a,y]$, to $[a,x]$, to $[a,y]$, and then to $[a,x]$. So a final configuration term $\{q_1,[a,x],z\}_{\overrightarrow{k^0_{g1}}}$ can be generated.

E. Understanding "Role" and the Bounds on Nonces

We add more explanations in Table I than the original table appearing in Page 282 of [10]. What we call "role instance" in Table I is called "role" in the table of [10]. The original table considers the nonces generated by all regular agents or the attacker I. The terms "bounded $\exists$" and "Unbounded $\exists$" in the original table are replaced by "Bounded total $\exists$ from regular agents" and " Unbounded total $\exists$ from regular agents" in Table I, respectively. We use the word "total" since in [10] it is the total number of nonces generated from all regular agents that is considered. Since the consideration of nonces generated from I is orthogonal to the consideration of nonces generated from regular agents, the words "I with $\exists$" and "I no $\exists$" appearing in the original table are replaced by "I with unbounded $\exists$" and "I with bounded $\exists$", respectively. Note that assuming I with bounded $\exists$ will not make any different results in the table from assuming I with no $\exists$, since the same results can be proved by the same reasoning assuming I with no $\exists$. We quote some sentences from Durgin's thesis [11], where more explanations of some aspects are included than [10].

In Page 49 of [11], section 2.5.3.2, in the proof of "Bounded existentials without disequality is in DEXP", the key reason to show DEXP is the following:

> "Therefore there is an exponential bound on the number of distinct ground facts that may appear in any possible protocol execution."

There is no need to differentiate the bounded number of distinct ground facts generated from regular agents or from the attacker, in order to show the DEXP results.

Later in the same proof, in Page 50 of [11]:

> "The above argument assumes that the number of intruder existentials is bounded."

Another place where the unbounded number of nonces generated from the attacker is the sentence in [11] Page 48 explaining the open problem, quote at the beginning of Appendix A.

¿From the quotations above, it is clear that "I no $\exists$" and "I with $\exists$"‘ in the original table in [10] are better explained as "I with bounded $\exists$" and "I with unbounded $\exists$", as shown in Table. I.

F. Outsider Attacker and Non-Matching RO protocol

Note that for these complexity results, the attacker is considered as an outsider, since according to the proofs in [10] and [11] of these results, the attacker does not know a key that is known by all insiders (with the definitions of this paper, we may call the key K and $K \in Pro.gk$), and the attacker cannot participate as an insider in a protocol run.

The protocols constructed in the proofs of [10] and [11] are a set of role templates. We call this kind of protocol Role-Oriented (RO). Especially, in these protocols, for some message received in a role, the corresponding role that sends the message is missing. We call this kind of RO protocol non-matching.

G. Understanding Bounded Size of Message Instances and Type Flaw

To understand the assumption of bounded message size, say bounded by K, in [10] and [11], there can be two interpretations.

1) Consider only those protocols such that every possible run of the protocol cannot have a message instance with size larger than K.

The protocol constructed in the proof of the undecidability result in [10] and [11], by translating a Turing machine tabular to a set of Horn clauses without function symbols, and then to the protocol, is of this kind. The reason is the following: (a) Every nonce appearing in the run is generated by a regular agent. (b) The attacker cannot construct any encryption due to his ignorance of the encryption key. (c) The attacker cannot put any term generated by him into any encryption term in the run. (d) Different kinds of encryption terms include different constants, so different encryption terms cannot be used interchangeably. So there is no type flaw in any run of the protocol, and every nonce variable is instantiated by an atomic term. Therefore the size of the largest message instance in a run is the same as the symbolic size of the largest message template in the protocol, which is chosen as the bound.

2) Although it is possible that in a run of the protocol a message can have unbounded size, the analysis only considers the runs with message sizes bounded, and neglects those runs having messages with sizes larger than K.

Assuming the interpretation 2), all of the results in the complexity tables in [10] and [11] are still right. Actually there is a caption of that table, saying that there is a limit on term size. In fact interpretation 2) is a stronger case covering the interpretation 1), which makes the decidable results of the table more interesting.

We have to justify that 2) is the right interpretation. The reason is that considering nonces from the attacker will induce a conflict with the interpretation 1), as explained below.

If the nonces generated by the attacker are not used in any messages accepted by regular agents, or only bounded many attacker's nonces appear in these accepted messages, the problem is trivially the same as assuming bounded number of nonces generated from the attacker. Then the open problem is not open, since its complexity is already obtained by [10] and [11], which is DEXP-complete, when regular agents can generate bounded number of nonces.

It will not be meaningful to discuss the open problem unless the unbounded attacker nonces effectively participate in a run of the protocol. That is, the attacker nonces can appear in messages accepted by regular agents. Then it is non-stoppable for the attacker to use composite term as a nonce, and introduce the type flaw. And then the size of a message in a run will never be guaranteed to be bounded.

There are sentences in [10] express the similar meaning as interpretation 2), such as the one in Page 279:

> "So we will consider derivations that limit both the length of messages and the depth of encryption, by bounding the size of the ground facts that can appear in a derivation."

So 2) is the only correct interpretation for the bounded message size assumption, which is applied to the open problem.

H. The Errors of [13] and Our Fix

In [13] a symmetric key, say K, is used as the encryption key for all encryptions, which is not known to the attacker. We use different asymmetric keys for different kinds of encryptions, for the reasons explained in Observation 1 of direction 2 in the proof of Theorem 1. We describe the errors of the scheme in [13] to reduce a 2-counter machine transition rule to a protocol role.

Error 1: 0 can be decremented, and effectively a counter of -1 can be reached. The dummy term $\{z,z\}^{\leftrightarrow}_{K}$ is always available since it is produced by the starting role. Then this term can be effectively used as the number connection term to show that $\underline{z} = \underline{z} - 1$. Or in fact $0 - 1 = 0$ is allowed.

Example: Suppose the only rule of the 2-counter machine is: $[q_0, 0, 1] \rightarrow [q_{final}, +1, -1]$. Starting from $(q_0, 0, 0)$, it is obvious that no rule is applicable and q_{final} cannot be reached. The corresponding transition role according to [13], using our notations of terms and nonce generation, and message sending and receiving. is :

1. $-(B \Rightarrow A): \quad \{z,z\}^{\leftrightarrow}_{K}, \{q_0, z, W_2\}^{\leftrightarrow}_{K}, \{W_2', w_2\}^{\leftrightarrow}_{K}$
2. $\#_A(W_1') \quad +(A \Rightarrow B): \quad \{z, W_1'\}^{\leftrightarrow}_{K}, \{q_{final}, W_1', W_2'\}^{\leftrightarrow}_{K}, \{z,z\}^{\leftrightarrow}_{K}$

However, the attacker can send the message: $\{z,z\}^{\leftrightarrow}_{K}, \{q_0, z, z\}^{\leftrightarrow}_{K}, \{z,z\}^{\leftrightarrow}_{K}$ as the first message to A. The terms $\{z,z\}^{\leftrightarrow}_{K}$ and $\{q_0, z, z\}^{\leftrightarrow}_{K}$ are known by the attacker since they are produced by a role instance of the starting role. Then A will respond and send the message $\{z, W_1'\}^{\leftrightarrow}_{K}, \{q_{final}, W_1', z\}^{\leftrightarrow}_{K}, \{z,z\}^{\leftrightarrow}_{K}$. It is wrong since a configuration term encoding a final configuration should not be generated.

The fix of error 1: In our reduction scheme, we make sure that no term of the form $\{X, z\}^{\rightarrow}_{g2}$ can be generated, as showed in Observation 2. So 0 can never be decremented, and the first error is avoided. After applying our rewrite rules, the actions of the corresponding role in the protocol for solving the open problem will be follows.

1. $-(B \Rightarrow A): \quad B, A, r_1, \{q_0, z, C_2\}^{\rightarrow}_{k^0_{g1}}, \{C_2^{-1}, C_2\}^{\rightarrow}_{k^0_{g2}}, C_1^{+1}$
2. $+(A \Rightarrow B): \quad A, B, \{q_{final}, C_1^{+1}, C_2^{-1}\}^{\rightarrow}_{k^0_{g1}}, \{z, [A, C_1^{+1}]\}^{\rightarrow}_{k^0_{g2}}$

The final configuration term will not be generated by this role, since there is no number connection term available to instantiate the term $\{C_2^{-1}, C_2\}^{\rightarrow}_{k^0_{g2}}$ in the first message.

Error 2: No guarantee that a counter is positive. When $i_h = 1$, for $h \in \{1, 2\}$, there is no way to make sure that the counter C_h is positive, and 0 can be used as a positive number.

Example: Suppose the only rule of the 2-counter machine is: $[q_0, 0, 1] \to [q_{final}, +1, +1]$. Starting from $(q_0, 0, 0)$, it is obvious that no rule is applicable and q_{final} cannot be reached.

The corresponding transition role according to [13] is :

1. $\quad -(B \Rightarrow A): \quad \{z, z\}^{\leftrightarrow}_{K}, \{q_0, z, W_2\}^{\leftrightarrow}_{K}, \{z, z\}^{\leftrightarrow}_{K}$
2. $\#_A(W'_1, W'_2) \quad +(A \Rightarrow B): \quad \{z, W'_1\}^{\leftrightarrow}_{K}, \{q_{final}, W'_1, W'_2\}^{\leftrightarrow}_{K}, \{W_2, W'_2\}^{\leftrightarrow}_{K}$

However the attacker can use z as the positive counter value W_2, and send the following message to A: $\{z, z\}^{\leftrightarrow}_{K}, \{q_0, z, z\}^{\leftrightarrow}_{K}, \{z, z\}^{\leftrightarrow}_{K}$. And then A will respond and send $\{z, W'_1\}^{\leftrightarrow}_{K}, \{q_{final}, W'_1, W'_2\}^{\leftrightarrow}_{K}, \{z, W'_2\}^{\leftrightarrow}_{K}$. It is wrong since a configuration term encoding a final configuration should not be generated.

The fix of error 2: When a counter value, say X, is supposed to be positive, there must be a term of the form $\{Y, X\}^{\rightarrow}_{g_2^0}$ received in the first message of the transition role. So the counter is guaranteed be positive, and the second error is avoided.

The actions of the corresponding role in the protocol for solving the open problem will be follows.

1. $\quad -(B \Rightarrow A): \quad B, A, r_1, \{q_0, z, C_2\}^{\rightarrow}_{k^0_{g1}}, \{C_2^{-1}, C_2\}^{\rightarrow}_{k^0_{g2}}, C_1^{+1}, C_2^{+1}$
2. $\quad +(A \Rightarrow B): \quad A, B, \{q_{final}, C_1^{+1}, C_2^{+1}\}^{\rightarrow}_{k^0_{g1}}, \{z, [A, C_1^{+1}]\}^{\rightarrow}_{k^0_{g2}}, \{C_2, [A, C_2^{+1}]\}^{\rightarrow}_{k^0_{g2}}$

The final configuration term will not be generated by this role, since there is no number connection term available to instantiate the term $\{C_2^{-1}, C_2\}^{\rightarrow}_{k^0_{g2}}$ in the first message. These fixes are crucial to maintain Observations 2, 4, and 5 in direction 2.

I. Matching and Non-matching Role-Oriented Protocols

Commonly a protocol is presented as a sequence of message exchanges, such as the first protocol showed in Appendix N. We call such a sequence as ***communication sequence***. The published protocols we have noticed (see the protocol library [18]) are all in the form of communication sequences.

Usually the first step to analyze a protocol as a communication sequence is to translate it into a set of roles. In other words, the corresponding set of roles can be used as the formal representation of the communication sequence. We call a protocol represented as a set of roles ***Role-Oriented***, or ***RO*** for short. Note that when we prove Theorem 1, the protocol presented is also RO. We can consider Def. 4 as the formal definition of a RO protocol.

In the published papers of the complexity of analyzing security protocols, see [9] [10] [11] [13] [14] [5] [6], a protocol is directly designed as a set of roles.

There is a difference between a RO protocol translated from a communication sequence, and a RO protocol directly designed without considering the corresponding communication sequence. The reason is that for the first one every message sending (or receiving) action in role can always be matched with a unique message receiving (or sending) action in another role. In Appendix N an example is presented to show a matching RO protocol translated from a communication sequence (the Otway-Rees protocol). This "matching" feature is formally defined in Def. 10. The second one could be non-matching, in the sense that for a message received in a role, this message may not be generated by any other role of the protocol.

All of the protocols constructed in the reductions of [9] [10] [13] [14] [5] [6], are non-matching.

When we need to consider a protocol as a communication sequence, we can adjust Definition 4 to include a CS filed, which is the communication sequence considered, and the $roles$ field should be derived from CS.

Definition 10: We call a RO protocol, say Pro, a ***matching*** one, if there is a one to one map, say map, between message sending actions and message receiving actions, and a substitution (assuming different roles of Pro does not share variables, i.e., every role assigns an implicit scope to its variables), say γ, such that:

$\forall R,\ R \in Pro.roles:$
$\quad if X \in R.acts,\ X = -(\_ \Rightarrow \_)msg,\ then$
$\quad\quad \exists R',\ R' \in Pro.roles,\ R' \neq R:$
$\quad\quad \exists Y,\ Y \in R'.acts,\ +(\_ \Rightarrow \_)msg'$ appears in $Y,\ map(X) = Y,\ map(Y) = X,\ msg\gamma = msg'\gamma$

and symmetrically

$\quad if X \in R.acts,\ +(\_ \Rightarrow \_)msg$ appears in X, then
$\quad\quad \exists R',\ R' \in Pro.roles,\ R' \neq R,$
$\quad\quad \exists Y:\ Y \in R'.acts,\ Y = -(\_ \Rightarrow \_)msg',\ map(X) = Y,\ map(Y) = X,\ msg\gamma = msg'\gamma$

A matching RO protocol can easily be translated back to the corresponding communication sequence. In other words, the communication sequence describes a run of the matching RO protocol. However for a non-matching RO protocol, the corresponding communication sequence may not exist.

The difference between matching and non-matching RO protocols will make the analysis a little different. The undecidability proofs of [9] [10] [11] [13] [14], which works for non-matching RO protocols, will not work straightforwardly for matching RO protocols. The reason is that in these proofs, the secret term is not encrypted in a message of a role. But if this happens in a matching RO protocol, it means that the secret term is guaranteed to be leaked, and thus secrecy is decidable, explained by the following fact.

Fact 1: For a matching RO protocol Pro, if the secret term Sec is not encrypted (or equivalently, it is encrypted but the attacker always knows the decryption key), then the attacker will always know Sec, and then the secrecy problem is decidable.

Proof: For a matching RO protocol, we can always interleave the messages between different roles to form a sequence of message exchanges, the communication sequence. This communication sequence describes a run of the protocol. Since Sec is leaked in this run, the problem of checking secrecy is decidable. ■

We also consider the non-matching RO protocols when we solve the open problem in Theorem 1, in order to be consistent with [10] and [11].

Bounding the Number of Agents:

J. Bounding the Number of Agents

In [20] the author proved that for a large family of protocols, when an agent is not allowed to talk with herself, it is sufficient to consider only $K+1$ agents who can participate in a run of the protocol, plus the constant agents described in the protocol, where K is the number of roles of the protocol, to check secrecy and authentication goals. So once we choose such a set of agent names AN, with some other conditions satisfied, the assertion of the secrecy goal is equivalent to the following:

$$\forall run \ (run \in Runs^{Pro:D:AN}) : \forall X \ (X \in SEC) : X \notin D.know_I(run.E)$$

Bounding the number of agents considered in a run could benefit devising decision algorithms to check security protocols. However for our purpose of undecidability reduction, the proof does not need, and is independent of, assuming a bounded number of agents. When we talk about an arbitrary run, as in the direction 2 of the proof of Theorem 1, we only need to know that there is a finite number of agents involved.

K. List as Nested Pairs

If we consider a List as a nested sequence of pairs, the list $[a, b, c]$, which is the same as $[a, [b, c]]$, is unifiable with $[X, Y]$, where X and Y are instantiated by a and $[b, c]$ respectively. This can produce some tricky behavior in a run. Suppose we use a single encryption key, say a symmetric key G, for all encryptions, the same as the in proofs of other papers such as [13] [10] [6], the proof to solve the open problem may not work. The reason is that if there is a configuration term $\{q, X, Y\}_{\overleftrightarrow{G}}$, then the attacker may use the pair $[X, Y]$ as a nonce, and then $\{q, X, Y\}_{\overleftrightarrow{G}}$, which is the same as $\{q, [X, Y]\}_{\overleftrightarrow{G}}$, can be used as a number connection term, where $\underline{q} = \underline{[X, Y]} - 1$. We avoid this behavior of the attacker by using different encryption keys for different encryptions.

L. Proving Theorem 1 Considering Runs Without Type Flaw

We can adjust the design of the message template of the roles to prove Theorem 1 while we consider only the runs without type flaw. The idea is to let $[A, z]$ to encode 0, and to let $[A, V]$ to encode a positive number where A is an arbitrary agent, and V is a term which is not z. When a counter is incremented, every agent will check that $C_h^{+1} \notin mem$, so C_h^{+1} is guaranteed to be different from z. Therefore $[A, z]$ can still only encode the number 0. Now a configuration term is in the form of $\{q, [E_1, C_1], [E_2, C_2]\}_{\overrightarrow{k_{g1}^0}}$, and a number connection term is in the form of $\{[E_1, C_1], [E_2, C_1']\}_{\overrightarrow{k_{g2}^0}}$, for some agent names E_1 and E_2. The action steps of the transition roles as follows.

1. $-(B \Rightarrow A_f):$ $B, A_f, r_f, \{q, [E_1, C_1], [E_2, C_2]\}_{\vec{k^0_{g1}}}, \{[F_1, C_1^{-1}], [E_1, C_1]\}_{\vec{k^0_{g2}}}, \{[F_2, C_2^{-1}], [E_2, C_2]\}_{\vec{k^0_{g2}}},$ C_1^{+1}, C_2^{+1}
2. $+(A_f \Rightarrow B):$ $A_f, B, \{q', C_1', C_2'\}_{\vec{k^0_{g1}}}, \{[E_1, C_1], [A_f, C_1^{+1}]\}_{\vec{k^0_{g2}}}, \{[E_2, C_2], [A_f, C_2^{+1}]\}_{\vec{k^0_{g2}}}$

There are two types of terms, agent names which include A_f, B, E_h and F_h for $h \in \{1, 2\}$, and nonces which include C_h, C_h^{-1} and C_h^{+1} for $h \in \{1, 2\}$. We can consider z as a special fixed nonce. Agent A_f will check that E_h, $F_h \in AN$ upon receiving message 1 of R_f. When the attacker honestly generate the fresh nonces (atomic terms) C_1^{+1} and C_2^{+1}, the run has no type flaw.

The rewrite rules are adjusted to the follows.

1. $i_h = 0 \quad \Mapsto \quad C_h \rightarrowtail z;\ \{[F_h, C_h^{-1}], [E_h, C_h]\}_{\vec{k^0_{g2}}} \rightarrowtail \varepsilon$
2. $i_h = 1 \quad \Mapsto \quad C_h \rightarrowtail [W, V_1];\ \{[F_h, C_h^{-1}], [E_h, C_h]\}_{\vec{k^0_{g2}}} \in Msg_1$
3. $j_h = +1 \quad \Mapsto \quad C_h' \rightarrowtail [A_f, C_h^{+1}];\ C_h^{+1} \in Msg_1;\ \{[E_h, C_h], [A_f, C_h^{+1}]\}_{k^0_{g2}} \in Msg_2$
4. $j_h = 0 \quad \Mapsto \quad C_h' \rightarrowtail [E_h, C_h];\ \{[E_h, C_h], [A_f, C_h^{+1}]\}_{\vec{k^0_{g2}}} \rightarrowtail \varepsilon;$
5. $j_h = -1 \quad \Mapsto \quad C_h' \rightarrowtail [F_h, C_h^{-1}];\ \{[E_h, C_h], [A_f, C_h^{+1}]\}_{\vec{k^0_{g2}}} \rightarrowtail \varepsilon;$

The roles R_0 and R_{final} are adjusted accordingly in the obvious way. The rest of the proof is the same as the one for Theorem 1.

M. Examples of the actions of transition roles

Here we present examples showing the actions of some transition roles constructed according to the rewrite rules presented in the proof of Theorem 1.

Example 1: If $T_f = [q_0, 0, 1] \rightarrow [q_{final}, +1, -1]$, then the $R_f.acts =$:

1. $-(B \Rightarrow A_f):$ $B, A_f, r_f, \{q_0, z, C_2\}_{\vec{k^0_{g1}}}, \{C_2^{-1}, C_2\}_{\vec{k^0_{g2}}}, C_1^{+1}$
2. $+(A_f \Rightarrow B):$ $A_f, B, \{q_{final}, C_1^{+1}, C_2^{-1}\}_{\vec{k^0_{g1}}}, \{C_1, [A_f, C_1^{+1}]\}_{\vec{k^0_{g2}}}$

Example 2: If $T_f = [q_3, 0, 1] \rightarrow [q_5, +1, +1]$, then $R_f.acts =$:

1. $-(B \Rightarrow A_f):$ $B, A_f, r_f, \{q_3, z, C_2\}_{\vec{k^0_{g1}}}, \{C_2^{-1}, C_2\}_{\vec{k^0_{g2}}}, C_1^{+1}, C_2^{+1}$
2. $+(A_f \Rightarrow B):$ $A_f, B, \{q_5, C_1^{+1}, C_2^{+1}\}_{\vec{k^0_{g1}}}, \{C_1, [A_f, C_1^{+1}]\}_{\vec{k^0_{g2}}},$ $\{C_2, [A_f, C_2^{+1}]\}_{\vec{k^0_{g2}}}$

Example 3: If $T_f = [q_1, 1, 0] \rightarrow [q_9, 0, +1]$, then $R_f.acts =$:

1. $-(B \Rightarrow A_f):$ $B, A_f, r_f, \{q_1, C_1, z\}_{\vec{k^0_{g1}}}, \{C_1^{-1}, C_1\}_{\vec{k^0_{g2}}}, C_2^{+1}$
2. $+(A_f \Rightarrow B):$ $A_f, B, \{q_9, C_1, C_2^{+1}\}_{\vec{k^0_{g1}}}, \{C_2, [A_f, C_2^{+1}]\}_{\vec{k^0_{g2}}}$

N. The Otway-Rees protocol as roles

The Otway-Rees protocol is presented in two forms here, a sequence of messages, and a set of roles. According to the semantics of Otway-Rees [22], s is the name of the fixed server, and M is a nonce that should uniquely identify a session of the protocol run.

Note that in $RoleB$, the two encrypted terms that are unintelligible to B (B does not have the key $k_{A:s}$ to do the decryption) are represented two variables Y and X. The mem field of each agent is not required.

The Otway-Rees protocol as a sequence of messages exchanges:

1. $A \Rightarrow B$: $M, A, B, \{N_A, M, A, B\}_{\overleftrightarrow{k_{A:s}}}$
2. $B \Rightarrow s$: $M, A, B, \{N_A, M, A, B\}_{\overleftrightarrow{k_{A:s}}}, \{N_B, M, A, B\}_{\overleftrightarrow{k_{B:s}}}$
3. $s \Rightarrow B$: $\{N_A, KAB\}_{\overleftrightarrow{k_{A:s}}}, \{N_B, KAB\}_{\overleftrightarrow{k_{B:s}}}$
4. $B \Rightarrow A$: $M, \{N_A, KAB\}_{\overleftrightarrow{k_{A:s}}}$

The Otway-Rees protocol described as Role templates:

$Pro = [PID, roles, AN, pk, gk, rsts]$

$PID = OtwayRee$
$roles = \{RoleA, RoleB, RoleS\}$
* $RoleA = [RID, agent, vars, acts, conds]$

- $RID = role\_a$
- $agent = [name, init, mem]$
- $vars = \{A, B, N_A, KAB, M\}$
- $acts =$ 1. $\#_A(N_A, M)\ +(A \Rightarrow B)\ :\quad M, A, B, \{N_A, M, A, B\}^{\leftrightarrow}_{k_{A:s}}$
  2. $-(B \Rightarrow A)\ :\quad M, \{N_A, KAB\}^{\leftrightarrow}_{k_{A:s}}$
- $conds = \{1.pre : (\{A, B, s, k_{A:s}\} \subseteq agent.init,\ agent.name = A, A \in CN)\}$

* $RoleB = [RID, agent, vars, acts, conds]$

- $RID = role\_b$
- $agent = [name, init, mem]$
- $vars = \{A, B, N_A, N_B, M, KAB, X\}$
- $acts =$ 1. $+(A \Rightarrow B)\ :\quad M, A, B, X$
  2. $\#_B(N_B)\ -(B \Rightarrow s)\ :\quad M, A, B, X, \{N_B, M, A, B\}^{\leftrightarrow}_{k_{B:s}}$
  3. $-(s \Rightarrow B)\ :\quad Y, \{N_B, KAB\}^{\leftrightarrow}_{k_{B:s}}$
  4. $+(B \Rightarrow A)\ :\quad Y$
- $conds = \{$ $1.pre : (\{A, B\} \subseteq agent.init, agent.name = B, \{A, B\} \subseteq CN);$
  $2.pre : (\{s, k_{B:s}\} \subseteq agent.init)\ \}$

* $RoleS = [RID, agent, vars, acts, conds]$

- $RID = role\_s$
- $agent = [name, init, mem]$
- $vars = \{A, B, N_A, N_B, KAB, M\}$
- $acts =$
  1. $-(B \Rightarrow s) : M, A, B, \{N_A, M, A, B\}^{\leftrightarrow}_{k_{A:s}}, \{N_B, M, A, B\}^{\leftrightarrow}_{k_{B:s}}$
  2. $\#_s(KAB)+(s \Rightarrow B) : \{N_A, KAB\}^{\leftrightarrow}_{k_{A:s}}, \{N_B, KAB\}^{\leftrightarrow}_{k_{A:s}}$
- $conds = \{\ 1.pre : (\{A, B, s, k_{A:s}, k_{B:s}\} \subseteq agent.init,\ agent.name = s,\ s \in SN)\ \}$

$AN = [SN, CN]$. $SN = \{s\}$. CN will be instantiated in a run of Pro.
pk and gk are described in $rsts$. They will be instantiated in a run of Pro.
$rsts = \{\ pk = CN \cup SN;\ gk = \{\};$
$\forall\ agent\ P,\ P.name \in\ CN :\ P.init\ =\ pk \cup gk\ \cup \{k_{P.name:s}\};$
/* The mem field is not needed */
Let $skeys = \{k_{R:s} | R \in CN\};$
$\forall\ agent\ P,\ P.name \in\ SN :\ P.init = pk \cup gk \cup skeys\quad \}$