Secrecy Checking of Protocols: Solution of an Open Problem

Zhiyao Liang Rakesh M Verma

Computer Science Department, University of Houston,
Houston TX 77204-3010, USA
Email: zliang@cs.uh.edu, rmverma@cs.uh.edu
Telephone: (713) 743–3338 Fax: (713) 743–3335

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. The proof is by a reduction from 2-counter machines to protocols, and we prove both directions of the reduction in detail. The modeling and proof method are generally applicable and can be conveniently adapted to solve other problems about the complexity analysis of checking properties of protocols.

Key words: Cryptographic protocols, secrecy, undecidability.

1 Introduction

Analyzing the security properties of cryptographic protocols has been one of the most 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 algorithms 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 PCS (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 precisely in theorem 1.

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).

In a recently published paper [15], Froschle showed the NEXPTIME-complete complexity of the secrecy problem of a setting (problem 4 of [15]) where disequality tests are allowed and only bounded number of nonce can be used (can appear) in role instances (called sessions in [15]) executed by regular agents. The motivation to consider disequality tests is that the unbounded set of nonces may not be able to be reduced in a bounded set of terms trivially due to the enforcement of disequality tests. Although inn [15] the author mentioned that problem 4 is pinpointed out by [10], it does not match the description of the open problem of [10] quoted in the appendix of this paper for two reasons. First, the disequality tests in problem 4 can only apply to terms occurring in the same role instance, while the 'disequality test' in the open problem are used by an agent trying to enforce freshnessof a term and should be applied to terms recorded across different role instances. Second and more importantly, the open problem does not assume that "the number of fresh data used in honest sessions is bounded" as in problem 4. The notation of "bounded $\exists$" in [10] means bounded number of nonces are generated, not that bounded number of nonces that can be used. This assumption will make the open problem obviously decidable. Since the messages size are bounded in a run, and assuming agent names and terms other than nonce are bounded, although unbounded number of role instances are allowed, only bounded many distinct role instances (with different messages) will appear in the run. So problem 4 is reduced to a setting with bounded number of role instances (note that we only need to consider role instances executed by regular agents), independent to disequality tests allowed or not, a decidable situation. Note that the authors of [10] conjectured that the open problem is undecidable (page 71 of [11]).

The concept of freshness check of [14] is an assumption that in any where a nonce is received by a regular agent when it should be freshly generated according to the protocol it must be indeed fresh, that is, different from any terms (subterms) appeared so far in a run. One difference between the freshness check and consideration of the open problem is that freshness check is at the assumption level where how to implement it is unclear, where the open problem considers the internal operations of agents trying to ensure freshness. The second difference is that freshness check is global to all terms appeared so far in a run, where the uniqueness checks (which should be proper for the open problem) in this paper only ensure freshness local to individual agents. Recording terms in the memory of individual agents seems to be the only way to implement freshness check. The third difference is that freshness check of [14] ensures a nonce different from any subterms appeared so far in a run, where the open problem only require the uniqueness of a nonce among all nonces. Our proof, which considers uniqueness check described later, can easily be adapted to show the undecidability in the corresponding setting where 'freshness check' of [14]is assumed. To prove this, the adjustment to our proof is to design a protocol such that every message must go through a special server s, who records every subterm appeared so far in the run. This setting may have been mentioned in [16] but has been pointed out having an error in the undecidability proof by Froschle in [15], if [15] is right on the issue.

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. Our reduction scheme using 2-counter machine is inspired from [13]. However there are key differences. The paper [13] dealt with different problems, not the open problem. Because of the constraints of the open problem, we cannot use their scheme directly and need new ideas such as stamping nonces with agent id's. Moreover, we have found and fixed two errors in the reduction 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 D. The proof of correctness of the reduction in [13] is sketchy and consequently misses the two errors.

2 Notations and Modeling

We introduce our notations and modeling here. A more detailed description of them can be found at [17]. Notations are chosen in a style that is commonly used in the literature, e.g., [2]. The notations for asymmetric keys are new.

A *term* is either an atomic term or a composite term. An *atomic term* is a *variable* (represented by a symbol with at least one upper case letter), and a constant (a symbol without any upper case letter). A special constant is I, the name of the attacker. Asymmetric keys are atomic terms. 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. 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. This notation can also be adapted to describe the asymmetric keys generated during a run. A *composite* *term* is a list, or an asymmetric encryption, or a symmetric encryption. A *list* has the form of $[X, Y, \cdots]$, where X and Y are terms and the list contains finite number of member terms. A list is a simpler representation of a sequence of nested pairs. For example $[W, X, Y, Z]$ is the same as $[W, [X, [Y, Z]]]$. When a message is a list, the top level enclosing [ ] is omitted. 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, and it can be decrypted using the key k^{1-i}_A. 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 (Y could be a composite term). 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. A *message* is a term. Every message appearing in a run of a protocol is a ground term.

The attacker model is, as usual, the Dolev-Yao model [1]. There are different equivalent formalizations for the Dolev-Yao model, such as (not a comprehensive list) Paulson's [18], Multiset Rewriting (MSR) [10], Constraint Solving [19], and Strand Space [20]. Our model is somewhat similar to Paulson's [18] where a run is represented as a trace which is convenient for proofs based on induction. We will clarify the unique features of our model, which are needed for the open problem.

A clear consensus of modeling can be described as follows. A protocol can be described as a set of roles, each role is a sequence of actions steps of message sending or receiving executed by an agent. A run E is a sequence of actions steps formed by interleaving (prefixes of) role instances (called strands in [20]) executed by regular agents, where before every message Msg can be received by a regular agent after at a certain point of the run, say after E' which is a prefix of E, the attacker I must be able to construct Msg, or $Msg \in know_I(E')$. Here $know_I(E')$ represents the knowledge of the attacker built from a set well-known analysis and synthesis rules [18] on the messages appeared in E' and the attacker's initial knowledge $init_I$. The secrecy checking protocol is to check if a secret term Sec can be leaked, or $Sec \in know_I(E)$ after a run E of the protocol. The formal proof of our reduction is based on the formal definitions of a protocol run and $know_I(E)$, and should be independent to different but equivalent choices to define them. A reader familiar with the formal concepts of protocol run and attacker's knowledge can directly verify the correctness of the proof assuming their own definitions of *run* and $know_I(E)$.

In the reduction proofs of undecidability of published papers such as [9] [10] [13] [14] and NP-hardness [5] [6], a constructed protocol is presented directly as a set of roles, we call them *role-oriented* (RO) protocols. However we call these protocols non-matching, since they do not correspond to protocols in the form of a sequence of message exchanges, such as those in [21]. For compatibility with other papers and especially with [11] and [10], which describe the open problem, in this paper a protocol presented is RO and non-matching.

An ***agent*** is a tuple $[name, init, mem]$, where $name$ is its unique name, $init$ is its initial knowledge (a set of terms), and mem is the set of terms that it has remembered so far in the current protocol run. The *mem* field is essential to explain the open problem. The different patterns of the initial knowledge of agents will be defined in the protocol. *Regular agents* means honest agents.

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 like nonces (they should be different from all other terms appeared in the run so far) 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 $-(B \Rightarrow A) : Msg$. Some internal action can be implicitly described by the protocol code, such as equivalence checking for the values of the same term. 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.

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) $\#_A(t1, t2, \cdots)$ $+(A \Rightarrow B) : Msg$; 1) is executed by I, while other three can be executed by both a regualar agent or I. 4) is the only kind of action step that is a sequence of more than one action.

A ***role*** or *role template* or a *role type*, is a tuple $[RID,\ agent, vars,\ acts,\ conds]$, where RID is the UID of the role, which is a constant, *agent* is the the agent who will execute the role template, *vars* is the set of variables that appear in *acts* or *conds* (the other atomic terms appearing in the role are constants), *acts* is the sequence of action steps numbered sequentially starting from 1, and *conds* is the internal actions that are not implicitly expressible by the *acts*. 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 are enforced to be satisfied after the n^{th} action step is executed to update the properties of agents. Especially, when a condition $X \in Q$ is included in $n.pre$, it means to check term X is in set Q before the n^th action step. Q should be defined in the context of the protocol or the role. If $X \in Q$ appears in $n.post$ it means to insert term X into set Q after the n^{th} action step.

A ***role instance*** is a tuple $[agent, role, vmap, acts]$, where *agent* is the agent who executes the role instance, *role* is the role template for this role instance, *vmap* is a ground substitution, (note that*vmap*(*role.conds*) should be satisfied), and *acts* is the sequence of (ground) action steps, and $acts = vmap(role.acts)$.

A ***protocol*** Pro is a tuple $[PID,\ roles,\ AN, rsts]$, where PID is the UID of the protocol (a constant), *roles* is a set of role templates, AN is the set of agent names (insiders) which are to be instantiated in the setting of a run, $rsts$ is the restrictions describing the initial knowledge and initial memory (indicated as $mem^{initial}$) of the agents, and definitions of some related sets if needed, such as the a set of terms shared by a certain group of agents.

A ***Dolev-Yao attacker*** [1], or an ***attacker*** for short, is a tuple $[name, init_I, know_I]$, where by convention $name = I$, $init_I$ is the initial knowledge of the attacker, and $know_I$ is 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. $know_I(E)$ is calculated as the closure of a applying a set of well-known rules of term synthesis and analysis as showed in [18]. Details of these rules are presented in [17].

A ***run*** is a tuple $[Pro, D, R, AN, E, conds]$, where Pro is the protocol, D is the initial knowledge pattern of the specific Dolev-Yao attacker, R is a set of role instances that are executed honestly by regular agents, AN is the names of the agents who can legally participate in a run (AN instantiates $Pro.AN$), E is a sequence of actions steps, which is called a trace in Paulson's model [18], and $conds$ is the set conditions required for a run of the protocol. $conds$ includes the following conditions
1) $Pro.rsts$ should be satisfied. That is, the initial knowledge of every agent in AN are assigned with a set of ground terms according to $Pro.rsts$
2) For each role instance prefix r in R, $r.agent.name \in AN$, and $r.agent.name \neq I$, The 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) Each agent with its name included in AN is called an ***insider***. Especially, if $I \in AN$ then I is an ***insider attacker***, then I's initial knowledge patter should be the same as (some) other regular agents as specified by $Pro.rsts$. Otherwise if $I \notin AN$, I is an ***outsider***, and then we assume the attacker's initial knowledge pattern D will instantiate $I.init$ by a set of ground terms that is a subset of the initial knowledge of every regular agent (insiders), usually only the agent names and public keys of the regular agents and some constants that is known to every agent.
5) Let $W \diamond X$ represent a sequence formed by appending an element X to a sequence W. For a prefix of E, call it E' and it is a sequence of action steps, suppose $E' = W \diamond X$, $X \in r.acts$ for some $r \in R$. If $X = -(A \Rightarrow B)msg$, where B is the name of a regular agent , then $msg \in know_I(W)$.
6) For a fresh term X that is generated by I, the action $\#_I(M)$ where M is a list of terms including X, is explicitly included in $run.E$ before X can appear in any message receiving action step by a regular agent.

We present some explanations of the above conditions to define a run. For 2), only the behavior of regular agents are organized into role instances. Although the attacker can execute a role instance normally as a regular agents, its behavior are covered by the Dolev-Yao model, and we only need to care about condition 5), that is, I can obtain every message before it can be received by a regular agent. For 4) we assume an outsider's initial knowledge should be less than any insider. Condition 6) is included for the convenience of describing $known_I(E)$, since the term generated by I should be used to build the knowledge of I. Note that we do not need to explicitly record the message sending actions of the attacker.

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

The set of all possible runs of a protocol Pro, with a specific initial knowledge pattern D of the attacker, is indicated as $runs^{D:Pro}$. Given a protocol Pro, and a set of secret terms SEC, A ***secrecy problem*** is to check the validity of the following statement.

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

3 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, and the theme is not relevant to the special cases that secrecy problems are decidable.

The open problem is described in [10] and [11], and is precisely stated in Theorem 1. Appendix C discusses it in more detail.

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.

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. Note that in the proof we allow C_h and C_h^{-1} to be instantiated by a pair, where $h \in \{1, 2\}$. However, if we make a stronger requirement so that the open problem only considers runs of a protocol where no type flaw can occur, which means every variable can only be instantiated by an atomic term in a run considered, it is still undecidable. To prove this, we only need to adjust the protocol code in the proof and replace C_h and C_h^{-1} with pairs like $[A, C_h]$ and $[B, C_h^{-1}]$, and then encode 0 with a pair $[A, z]$ instead z, and then adjust the messages of the protocol accordingly. The rest of the proof is the same.

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 (constructed from the 2-counter machine) in which the secret term is leaked.

Definition 1. *A* **deterministic 2-counter machine** *[22] 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 is a tuple (q, C_1, C_2), where q is the current state and C_1 and C_2 are two non-negative integers representing the two counters. The 2-counter machine can detect whether a counter is 0 or not. A transition rule,* *(call the rule $T \in \delta$) is of the form $[q, i_1, i_2] \rightarrow [q', j_1, j_2]$, where $q, q' \in Q$; $i_1, i_2 \in \{0, 1\}$; $j_1, j_2 \in \{-1, 0, +1\}$. An application of T can be described as $(q, C_1, C_2) \longrightarrow^T (q', C_1', C_2')$, where LHS and RHS are the configuration before and after the transition respectively. For $h \in 1, 2$, when $i_h = 0$, it means that $C_h = 0$. When $i_h = 1$, it means that $C_h > 0$. When $j_h = +1$ ($j_h = 0$, $j_h = -1$), it means that after the transition, $C_h' = C_h + 1$ ($C_h' = C_h$, $C_h' = C_h - 1$). Especially, when $j_h = -1$, i_h must be 1, since decrementing 0 is not allowed. The reachability problem of such a 2-counter machine is to decide that, starting from the initial configuration $(q_0, 0, 0)$, after applying some applicable transition rules, whether some final configuration $(q_{final}, \_, \_)$ can be reached, where $\_$ represents an arbitrary possible value. We assume (for convenience) that $q_0 \neq q_{final}$ and, for nontriviality, that δ is not empty.*

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

Theorem 1. *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, (vii) considering only the runs where the number of agents is bounded, and (viii) 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. 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 constructed according to M.

The messages received in a role is in the format of: $sender$, $receiver$, $receiver's$ $role$, $\cdots$, so the receiver of the message has clear hints to understand the message and know what he should do. The variables B, A_{final}, A_0, A_f, for $1 \leq f \leq n$ are agent names. We differentiate the namess of the variables representing the executors of different roles, including A_{final}, A_0, A_f, $1 \leq f \leq n$, for the clarity of the presentation, although a single variable can be used in different roles. B represents some agent talking with the executor of every role.

The set of secret term SEC is defined in $Pro.rsts$, which is initially known by every regular agent but the attacker. We specify the secret term Sec as a variable, instead of constant in the messages of a role, for a practical concern, so even though the attacker knows the protocol code, the attacker does not know the instance of Sec unless a role instance of R_{final} can be executed in a run, where Sec will be instantiated by a member of SEC.

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.

$Pro = [PID,\ roles,\ AN,\ pk,\ gk,\ rsts]$. 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. $+(A_0 \Rightarrow B):\quad A_0, B, \{q_0, z, z\}_{\overrightarrow{k^0_{g1}}}$
  - $conds = \{$ 1.$pre : (q_0,\ A_0,\ r_0,\ B,\ k^0_{g1}\} \subseteq init,\ agent.name = A_0,\ \{A_0, B\} \subset AN, A_0 \neq B)\ \}$
- $R_{final} = [RID,\ agent,\ vars,\ acts,\ conds]$
  - $RID = r_{final}$; $agent = [name,\ init,\ mem]$; $vars = \{A_{final},\ X,\ Y,\ B,\ Sec\}$.
  - acts= 1. $-(B \Rightarrow A_{final}):\quad B, A_{final}, r_{final}, \{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$
    2. $+(A_{final} \Rightarrow B):\quad A_{final}, B, Sec$
  - $conds =$ 1.$pre : (\{q_{final},\ A_{final},\ r_{final},\ B,\ k^1_{g1}\} \subseteq init,$
    $agent.name = A_{final}, \{A_{final}, B\} \subset AN,\ A_{final} \neq B)\ )$;
    2.$pre : (\ Sec \in init,\ Sec \in SEC\ )\ \}$
- For each $T_f \in \delta$, for some f, $1 \leq f \leq n$, suppose $T_f = [q, i_1, i_2] \rightarrow [q', j_1, j_2]$. $R_f \in roles$. 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. $-(B \Rightarrow A_f) : B, A_f, r_f, \{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}, \{C_1^{-1}, C_1\}_{\overrightarrow{k^0_{g2}}},$
       $\{C_2^{-1}, C_2\}_{\overrightarrow{k^0_{g2}}},\ C_1^{+1}, C_2^{+1}$
    2. $+(A_f \Rightarrow B) : A_f, B, \{q', C_1', C_2'\}_{\overrightarrow{k^0_{g1}}}, \{C_1, [A_f, C_1^{+1}]\}_{\overrightarrow{k^0_{g2}}},$
       $\{C_2, [A_f, C_2^{+1}]\}_{\overrightarrow{k^0_{g2}}}$

For $h \in \{1,2\}$, the following rewrite rules are applied to adjust the above role template to make each individual transition role R_f according to the conditions satisfied by the corresponding transition rule T_f of the 2-counter machine. Each rewrite rule is described as "condition $\mapsto$ effects".

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

$W \rightarrowtail V$ 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". $h \in \{1,2\}$. 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, we emphasize that there must be evidence that it has a preceding nonnegative value. This rule is redundant since a default rule is that any term that is not removed from the template will still be there.
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 new nonce received by A_f. Note that in a run C_h^{+1} is provided by the intruder who impersonates B_f. 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 D shows some examples.

The conditions of R_f ($R_f.cond$), $1 \leq f \leq n$, is the follows.

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

We continue to finish describing remaining fields of *Pro*.
* *AN* 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* = { $pk = Q \cup AN \cup \{r_0, r_{final}, r_1, \cdots, r_n\} \cup \{z, k_{g1}^1, k_{g2}^1\}$; Let *SEC* be a set of terms. $SEC \cap pk = \{\}$; $gk = \{k_{g1}^0, k_{g2}^0\} \cup SEC$; $\forall P(P \in CA)$: $P.init = pk \cup gk$, $P.mem^{initial} = P.init$ }

The initial knowledge pattern D of I's initial knowledge (as an ousider) is: $init.I = pk$, where pk is defined in *Pro.conds*.

We show that the constructed protocol and the proof satisfies the bounds imposed by the open problem, before we show further details of the reduction. Note that no regular agent will generate any fresh nonce, so the nonces generated from regular agents are trivially bounded. All of the nonces, which instantiate C_h^{+1}, $h \in \{1, 2\}$, in every role instance of R_f, are 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, $1 \leq f \leq n$. And every regular agent is required to do the uniqueness check of each term received that is supposed to be fresh nonce. The number of agents can be bounded by three, since in the proof (direction 1) we only assume two regular agents a and b in a run, while the intruder I is the third agent in a run.

As explained in the introduction section, the protocol is a non-matching role oriented one. The attacker is an outsider as described by the following paragraph.

A symmetric key is used as the encryption key in [13] [10] [6], which is known to all the regular agents (who are insiders) but unknown to the attacker (who is an outsider). So the attacker can neither construct an encryption, nor understand it, which could make it not practical for attacker to deploy an attack. In the proof of this paper we also consider the attacker is an outsider and we choose asymmetric keys k_{g1}^0 and k_{g2}^0 as the encryption keys, which are unknown to the attacker, but known to all of the regular agents. $g1$ and $g2$ are the UID of the key pairs, not agent names. The attacker I knows the decryption key k_{g1}^1 and k_{g2}^1. So I cannot construct the encryptions, but can decrypt them and understand them, and easily deploy the attack. Every role can only be executed by some regular agent since attacker cannot construct the encryptions in the messages.

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 action steps E, we say a term X is the ***encoding*** of a positive integer N, if and only if there is a sequence of terms:
$\{z, X_1\}_{\overrightarrow{k^0_{g2}}}, \{X_1, X_2\}_{\overrightarrow{k^0_{g2}}}, \{X_2, X_3\}_{\overrightarrow{k^0_{g2}}}, \cdots, \{X_{N-2}, X_{N-1}\}_{\overrightarrow{k^0_{g2}}}, \{X_{N-1}, X\}_{\overrightarrow{k^0_{g2}}}$
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), or X is the ***encoding*** of N, denoted as $N = \underline{X}$. We say X ***encodes*** N. The above term sequence is called the ***encoding sequence*** of X. The encoding sequence of z is z.

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 the E of a run, let $E = W \diamond X$. We will show that if the last action setp X of E is to receive a message Msg, we will show that $Msg \in know_I(W)$.

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, such that $Sec \in know_I(run.E)$ for some term $Sec \in SEC$. We prove this direction by constructing run. Pro is the protocol just described.

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

- $Pro = [PID, roles, agents, AN, pk, gk, conds]$. Especially, the instantiation of pk will be clear once AN is specified.
- D is the pattern which requires that $init_I = Pro.pk$.
- 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.
- $AN = \{a, b\}$. 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. we will justify that is every message received by a regular agent can be constructed by the attacker.

Now we focus on describing $run.E$, which can be divided into three parts: the starting, the transition, and the finishing. 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, we only have to show two things. 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 steps***. 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 B is instantiated by b. The first action of $run.E$ is: $+(A_0 \Rightarrow s): \quad A_0, B, \{q_0, z, z\}_{\vec{k^0_{g1}}}$.

The ***transition action steps***. 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 $run.E$: $\#_I(c_1^w, c_2^w)$, where c_1^w and c_2^w 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.

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

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

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 $\{q, C_1, C_2\}_{\vec{k^0_{g1}}}$ 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 $\{q, C_1, C_2\}_{\vec{k^0_{g1}}}$, and $\{C_h^{-1}, C_h\}_{\vec{k^0_{g2}}}$ (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 steps of M 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 action steps 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* $\{q', C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^{w'})$*.*

This lemma can be proven by induction on the length of the computation of M. The details are included in Appendix A.

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

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

The following observations are easy to verify. Due to limit of space, detailed explanation are included in [17].

Observation 1: First, every encrypted term is constructed by a regular agent. Second, two encrypted terms appearing in *run* with different format cannot be unified and cannot be used interchangeably due to different encryption keys. .

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, z must be a freshly generated nonce, impossilbe.

Observation 3: Given any term X, X can appear at most once in a term of the form $\{Y, X\}_{\overrightarrow{k^0_{g2}}}$. Assuming the contrary, we can see that X must have the form of $[A, T]$, where T has been accepted by the same agent A as a fresh nonce twice, impossible.

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 of Pro with the attacker (an outsider, as described earlier) 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* $h \in \{1, 2\}$*.*

This lemma is proven by induction on the sequence of configuration terms generated in the *run*. The detailed proof is included in Appendix A.

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. □

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 1 in Appendix C.

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.*

The proof of Theorem 2 is based on the proof of Theorem 1. The detailed proof is included in Appendix B. The idea is to create fresh copies of nonces which can encode counter values already reached in the computation, and then generate fresh copies of produced configuration terms where the terms encoding counter values are replaced by fresh and equivalent (in terms of number encoding) nonces. By organizing a bounded number of agents to execute role instances alternatively, an unbounded number of fresh copies of terms can be made.

4 Summary

We solve the open problem of Durgin, Lincoln and Mitchell [10] [11] using a direct reduction scheme from the reachability 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. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–207
2. Lowe, G.: An Attack on the Needham-Schroeder Public-Key Authentication Protocol. Inf. Process. Lett. **56**(3) (1995) 131–133
3. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In: TACAS. (1996) 147–166
4. Needham, R.M., Schroeder, M.D.: Using Encryption for Authentication in Large Networks of Computers. Commun. ACM **21**(12) (1978) 993–999
5. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is NP-complete. In: CSFW. (2001) 174–
6. Rusinowitch, M., Turuani, M.: Protocol insecurity with a finite number of sessions, composed keys is NP-complete. Theor. Comput. Sci. **1-3**(299) (2003) 451–475
7. Amadio, R.M., Lugiez, D., Vanackère, V.: On the symbolic reduction of processes with cryptographic functions. Theor. Comput. Sci. **290**(1) (2003) 695–740
8. Comon, H., Cortier, V.: Tree automata with one memory set constraints and cryptographic protocols. Theor. Comput. Sci. **331**(1) (2005) 143–214
9. Durgin, N.A., Lincoln, P., Mitchell, J.C., Scedrov, A.: Undecidability of bounded security protocols. In Heintze, N., Clarke, E., eds.: Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy. (july 1999)
10. Durgin, N.A., Lincoln, P., Mitchell, J.C.: Multiset rewriting and the complexity of bounded security protocols. Journal of Computer Security **12**(2) (2004) 247–311
11. Durgin, N.A.: Logical Analysis and Complexity of Security Protocols. PhD thesis, Computer Science Department, Stanford University (March 2003)
12. Even, S., Goldreich, O.: On the security of multi-party ping-pong protocols. In: IEEE Symposium on Foundations of Computer Science. (1983) 34–39
13. Ramanujam, R., Suresh, S.P.: Undecidability of secrecy for security protocols. Manuscript (July 2003)
14. Ferucio L. Tiplea and C. Enea and C. V. Birjoveanu: Decidability and complexity results for security protocols. In: Verification of Infinite-State Systems with Applications to Security, IOS Press (2006) 185–211
15. Froschle, S.: The insecurity problem: Tackling unbounded data, `http://homepages.inf.ed.ac.uk/sib/publ.html` To be published in 20^{th} IEEE Computer Security Foundations Symposium.
16. Ferucio L. Tiplea and C. Enea and C. V. Birjoveanu: Secrecy for bounded security protocols with freshness check is nexptime-complete. (2007) To be published in Journal of Computer Security.
17. Liang, Z., Verma, R.M.: Secrecy Checking of Protocols: Solution of an Open Problem. Technical report, `http://www.cs.uh.edu/preprint` (April 2007) Technical Report UH-CS-07-04.
18. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6**(1-2) (1998) 85–128
19. Millen, J.K., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: ACM Conference on Computer and Communications Security. (2001) 166–175
20. Thayer, F.J., Herzog, J.C., Guttman, J.D.: Strand spaces: Proving security protocols correct. Journal of Computer Security **7**(1) (1999)
21. Clark, J., Jacob, J.: A survey of authentication protocol literature: Version 1.0. Technical report (1997)

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

A Proofs of Lemmas

Lemma 1: For the role instance r^w and the transition steps 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 $\{X, C_h\}_{\overrightarrow{k_{g2}^0}}$, and $\underline{X} = V_h - 1$. The instance of $\{X, C_h\}_{\overrightarrow{k_{g2}^0}}$ must be included in $know_I(E^w)$, and is used to instantiate $\{C_h^{-1}, C_h\}_{\overrightarrow{k_{g2}^0}}$. The second fact is proven.

Fact 3. We only need to show that $V'_h = \underline{C'_h}$, and the term $\{q', C'_1, C'_2\}_{\overrightarrow{k_{g1}^0}}$ 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 $\{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k_{g2}^0}}$ 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, $V'_h = V_h + 1 = \underline{[A_f, C_h^{+1}]} = \underline{C'_h}$.

□

Lemma 2 : For an arbitrary *run* of *Pro* with the attacker (an outsider, as described earlier) for every configuration term of the form $\{q, C_1, C_2\}_{\overrightarrow{k_{g1}^0}}$ 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 $\{q_0, z, z\}_{\overrightarrow{k_{g1}^0}}$ generated by a role instance of R_0. Obviously $\{q_0, z, z\}_{\overrightarrow{k_{g1}^0}}$ 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 $X = \{q_0, z, z\}_{\overrightarrow{k_{g1}^0}}$ 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 $\{q', C_1', C_2'\}_{\overrightarrow{k_{g1}^0}}$ generated in the second action step of $r.acts$. By the construction of the protocol, there must be a term $\{q, C_1, C_2\}_{\overrightarrow{k_{g1}^0}}$ 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, $\{q, C_1, C_2\}_{\overrightarrow{k_{g1}^0}}$ 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 $\{q', C_1', C_2'\}_{\overrightarrow{k_{g1}^0}}$ encodes it.

First, we show that t is applicable to (q, V_1, V_2). t must have the form of $[q, i_1, i_2] \to [q', j_1, j_2]$. 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 $\{C_h^{-1}, C_h\}_{\overrightarrow{k_{g2}^0}}$ 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 $\{q', C_1', C_2'\}_{\overrightarrow{k_{g1}^0}}$. 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 $\{C_h, [A_f, C_h^{+1}]\}_{\overrightarrow{k_{g2}^0}}$ 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 $\{C_h^{-1}, C_h\}_{\overrightarrow{k_{g2}^0}}$ 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 $\{Y, C_h\}_{\overrightarrow{k_{g2}^0}}$ 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 $\{Y, C_h\}_{\overrightarrow{k_{g2}^0}}$ is unique, Then, the attacker can only use $\{Y, C_h\}_{\overrightarrow{k_{g2}^0}}$ as the term $\{C_h^{-1}, C_h\}_{\overrightarrow{k_{g2}^0}}$. So $V_h' = V_h - 1 = \underline{C_h^{-1}} = \underline{C_h'}$.

So the induction step is proven. □

B Detailed Proof of Theorem 2

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. 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 requires 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* : $(A_{s1}, r_{s1}, B, k_{g2}^1, k_{g3}^1, z\} \subseteq init,\ name = A_{s1},$
$\{A_{s1}, B\} \subset AN, A_{s1} \neq B, \{X, Y, V_1, V_2\} \cap mem = \emptyset,$
$X \neq Y,\ X \neq V_1,\ X \neq V_2,\ Y \neq V_1,\ Y \neq V_2,$
$V_1 = z\ or\ V_1 \notin mem,\ \ V_2 = z\ or\ V_2 \notin mem,$
$V_1 = V_2 = z\ or\ V_1 \neq V_2)$
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 unbounded 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. $-(B \Rightarrow A_{s2}) : B, A_{s2}, r_{s2}, \{G, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}, \{C_1, X\}_{\overrightarrow{k^0_{g3}}}, \{C_2, Y\}_{\overrightarrow{k^0_{g3}}}$
2. $+(A_{s2} \Rightarrow B) : A_{s2}, B, \{G, X, Y\}_{\overrightarrow{k^0_{g1}}}$
– *conds* = { 1.*pre* :($\{A_{s2}, r_{s2}, B, k^1_{g1}, k^1_{g3}, z\} \subseteq init$, $name = A_{s2}$,
$\{A_{s2}, B\} \subset AN, A_{s2} \neq B, \{X, Y\} \cap mem = \emptyset, G \in Q$,
$C_1 = z$ *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* : ($k^0_{g1} \in init$) }

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 $\{z, z\}_{\overrightarrow{k^0_{g3}}}$. 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 $\{z, X\}_{\overrightarrow{k^0_{g3}}}$ 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 $\{z, z\}_{\overrightarrow{k^0_{g3}}}$.

Observation 7: Whenever an equivalence term of the form $\{U, V\}_{\overrightarrow{k^0_{g3}}}$ 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 $\{z, z\}_{\overrightarrow{k^0_{g3}}}$ 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 $\{X, [A_{s1}, Y]\}_{\overrightarrow{k^0_{g3}}}$ generated in the second message (Msg_2) of a role instance of R_{s1}. Since $\{V_1, V_2\}_{\overrightarrow{k^0_{g3}}}$ 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 $\{V_1, X\}_{\overrightarrow{k^0_{g2}}}$ appears in Msg_1, X must encode the number $\underline{V_1} + 1$. Since in Msg_2 $\{V_2, [A_{s1}, Y]\}_{\overrightarrow{k^0_{g2}}}$ 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 $\{U, V\}_{\overrightarrow{k^0_{g2}}}$ 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 $\{V_2, [A_{s1}, Y]\}_{\overrightarrow{k^0_{g2}}}$ is generated by a role instance of R_{s1}, V_2 must encode some number since $\{V_1, V_2\}_{\overrightarrow{k^0_{g3}}}$ 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 $\{q, z, z\}_{\overrightarrow{k^0_{g1}}}$ 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 $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$, 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 counter 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_1^i, C_2^i, 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_1^{i+1}, C_2^{i+1}, X^{i+1}, Y^{i+1}, ${C_1^{-1}}^{i+1}$, and ${C_2^{-1}}^{i+1}$, where C_1^{i+1} could be ${C_1^{-1}}^i$ or X^i, similarly for C_2^{i+1}. 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_1^{i+2}, C_2^{i+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, $1 \le f \le n$, 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.

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

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

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 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. □

Table 1. 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.

C Understanding the open problem

Table 1 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 C.6) , 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 1 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, 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.

C.1 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 1 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.2 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.

C.3 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.

C.4 Understanding "Role" and the Bounds on Nonces

We add more explanations in Table 1 than the original table appearing in Page 282 of [10]. What we call "role instance" in Table 1 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 ∃" and "Unbounded ∃" in the original table are replaced by "Bounded total $\exists$ from regular agents" and " Unbounded total $\exists$ from regular agents" in Table 1, 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 C.

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. 1.

C.5 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.

C.6 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.

C.7 Assuming Bounded Number of Agents

In the description of the open problem in theorem 1 we assumed that (condition vii) only runs with bounded number of agents are considered. The reason is the follows. First, in the undecidability proof of [10], only a small number of agents need to participate in a run. Second, allowing unbounded many agents in a run the proof of Theorem 1 still works and the open problem is also undecidable. However, assuming bounded number of agents makes the proof more challenging, since assuming more bounds in general can make a undecidability result stronger which automatically implies the undecidability result of a corresponding setting with less bounds. The proof of Theorem 2 would be considerably easier if unbounded number of agents are allowed in a run. Third, if unbounded number of agents are allowed to participate in a run, the attacker could always choose a new agent to execute the next role instance. Then since every agent can only execute one role instance, the behavior of uniqueness check of an agent A to terms received by A in a run so far, as required by the open problem, becomes irrelevant, and there is no need to specify $A.mem$. We can observe that if we require only bounded number of nonces can be generated, while unbounded number of agents are allowed, the proof of Theorem 1 can still work. The reason is that the unbounded number of agent names can be used to simulate the unbounded counter values and carry out the reduction. This observation also confirms that the general reason of undecidability for checking security protocols is that unbounded number of different terms can appear in a run of a protocol.

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

However, the attacker can send the message: $\{z,z\}_K^{\leftrightarrow}, \{q_0, z, z\}_K^{\leftrightarrow}, \{z,z\}_K^{\leftrightarrow}$ as the first message to A. The terms $\{z,z\}_K^{\leftrightarrow}$ and $\{q_0, z, z\}_K^{\leftrightarrow}$ 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. $-(B \Rightarrow A) : B, A, r_1, \{q_0, z, C_2\}^{\rightarrow}_{k^0_{g1}}, \{C_2^{-1}, C_2\}^{\rightarrow}_{k^0_{g2}}, C_1^{+1}$
2. $+(A \Rightarrow B) : A, B, \{q_{final}, C_1^{+1}, C_2^{-1}\}^{\rightarrow}_{k^0_{g1}}, \{z, [A, C_1^{+1}]\}^{\rightarrow}_{k^0_{g2}}$

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: $[q_0, 0, 1] \rightarrow [q_{final}, +1, +1]$. 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] is :

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

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 $\{Y, X\}^{\rightarrow}_{g^0_2}$ 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. $-(B \Rightarrow A) : B, A, r_1, \{q_0, z, C_2\}^{\rightarrow}_{k^0_{g1}}, \{C_2^{-1}, C_2\}^{\rightarrow}_{k^0_{g2}}, C_1^{+1}, C_2^{+1}$
2. $+(A \Rightarrow B) : A, B, \{q_{final}, C_1^{+1}, C_2^{+1}\}^{\rightarrow}_{k^0_{g1}}, \{z, [A, C_1^{+1}]\}^{\rightarrow}_{k^0_{g2}}, \{C_2, [A, C_2^{+1}]\}^{\rightarrow}_{k^0_{g2}}$

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.