Improving Techniques for Proving Undecidability of Checking Cryptographic Protocols

Zhiyao Liang
Computer Science Department
University of Houston
Houston TX 77204-3010, USA
zliang@cs.uh.edu

Rakesh M Verma
Computer Science Department
University of Houston
Houston TX 77204-3010, USA
rmverma@cs.uh.edu

Abstract

Existing undecidability proofs of checking secrecy of cryptographic protocols have the limitations of not considering protocols common in literature, which are in the form of communication sequences, since only protocols as non-matching roles are considered, and not considering an attacker who is an insider since only an outsider attacker is considered. Therefore the complexity of checking the realistic attacks, such as the attack to the public key Needham-Schroeder protocol, is unknown. The limitations have been observed independently and described similarly by Froschle in a recently published paper [9], where two open problems are posted. This paper investigates these limitations, and we present a generally applicable approach by reductions with novel features from the reachability problem of 2-counter machines, and we solve the two open problems. We also prove the undecidability of checking authentication which is the first detailed proof to our best knowledge. A unique feature of the proof is to directly address the secrecy and authentication goals as defined for the public key Needham-Schroeder protocol, whose attack has motivated many researches of formal verification of security protocols.

1. Introduction

Since networks are indispensable to all kinds of communications nowadays, checking and analyzing cryptographic protocols are especially important. A significant research direction is to check secrecy and authentication of protocols against a dominating attacker, first introduced by Dolev and Yao [4], assuming the cryptographic primitives cannot be broken. Since Lowe discovered an attack [16] on the public key Needham-Schroeder protocol (call it PKNS protocol) 17 years after it was published [19], many papers have focused on the topics in this area using formal methods. An essential part of these researches is to investigate the complexity of checking security protocols. Here we focus on the undecidability results.

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

1.1. Limitations of Existing Undecidability proofs

Undecidability of secrecy checking has been mentioned by other researchers in several papers [1] [3] [6] [5] [7] [21] [8], and [6] [5] [21] provide proofs with details. In [5] and [6] the authors used multi-set rewriting to analyze protocols. The proof is by a two-stage reduction from the halting problem of Turing machine with the style of Turing machine tableau to Horn clause theories without function symbols and then from Horn clause theories to protocols specified as a set or roles. In [21] the authors showed that the undecidability result of [6] can be proved more directly by a reduction from the reachability problem of 2-counter machines to the secrecy checking problem of protocols as sets of roles (we call them role-oriented protocols).

The above proofs of undecidability have three limitations. **First**, all of the undecidability proofs, except [3], do not consider a protocol as a sequence of message exchanges, which we call a ***communication sequence***, or ***CS*** for short. The published protocols we have noticed (see the

0-7695-3102-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ARES.2008.198

protocol library [2]) are all in the form of communication sequences. These undecidability proofs directly consider a protocol as a set of roles, we call this kind of protocols ***Role Oriented*** or ***RO*** for short, where each role is a sequence of message sending and receiving executed by a principal (also called an agent). Usually the first step to analyze a protocol as a communication sequence is to translate it into a set of roles, where the action steps executed by the same agent are organized into a role, and the relative order of action steps in the communication sequence is kept. Every step of the communication sequence implies two action steps, one is the message sending from the sender's role, and the other is the corresponding message receiving in the receiver's role.

However, there is a difference between an RO protocol translated from a CS, and a RO protocol directly designed without considering the corresponding CS. The first one is ***matching***, in the sense that every message sending (or receiving) action step in a role can always be matched with (be unified with) a unique message receiving (or sending) action step in another role. The second one could be ***non-matching***. We call a RO protocol non-matching if for some message received (sent) in a role, the other corresponding role in the protocol where this message is sent (received) does not exist. In other words, a set of non-matching roles are impossible to be obtained by parsing any CS. In the proofs mentioned earlier, except [3], the protocols considered are non-matching RO.

In [3] the authors showed a proof of the undecidability of secrecy checking by a reduction from the reachability problem of Petri nets. The protocol constructed in the reduction is called 'real' and has at least one honest run. A 'real' protocol is equivalent to what we call a communication sequence in this paper. However, as noticed by Froschle in [9], the proof of [3] depends on messages with unbounded size in a run, which is a part of the motivation of the first open problem of [9] (quoted later in this paper). Bounding message size in a run (or only consider runs where message size is bounded) is a condition making the undecidability stronger and more interesting as proposed by [6]. The proof of [3] has another limitation of considering an outsider attacker (the third limitation, described later). A unique feature of our proof to handle 'real' protocol is that we directly address secrecy as in the public key Needham-Schroeder protocol, which has been a phenomenal research focus.

Second, improper secrecy declaration. This limitation, described below, is directly related to the first limitation and can show further why the proofs are not quite suitable for practically designed protocols. The proofs (except [3]) have some role where a term, which is declared as the secret one, is sent out in a message, not encrypted or trivially encrypted so that the attacker can know the secret term once he can get the message. Suppose this role belongs to a set of roles which are translated from some practically designed protocol as a CS, we can see that an honest run of the protocol (running the CS) will inevitably send the message containing a secret term and the attacker can always know the secret term. In other words, secrecy checking for protocols in the form of a CS with this 'trivial' secrecy declaration is always decidable, and the undecidability proofs will not work.

Froschle has independently noticed the above two limitations in a recently published paper [9]. She mentioned protocols having an honest run or 'real' protocols (notions first used by [3]), and proposed the first open problem, as quoted below.

> "A protocol has an *honest run* if all of its rules can be played in the given orders: the first rule, then the second, and so on. "
>
> *Problem 1.* "Is Insecurity decidable for protocols with honest runs when the message size is bounded?"

Third, considering the attacker as an outsider.

Every protocol assumes a perfect *initial knowledge establishing stage* at the beginning of a run. Note that if the initial knowledge of agents are not established perfectly and securely, there is no way to guarantee any security of the protocol. In this stage, keys and other terms will be distributed among a group of agents following some initial knowledge policy required by the protocol. We call this group as the ***legitimate agents group***. We call every agent belonging to this group as an ***insider*** to this group, and every other agent as an ***outsider*** to this group.

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

Actually to consider an insider attacker has special importance, since in practice, the majority of security failures may be due to some insider attacker [10]. Gollmann mentioned in [10] that in the attack to the PKNS protocol [16] the attacker should be considered as an insider. The reason seems to be that one of the regular agent must initially know the attacker. We define insider based on the initial knowledge establishing stage of a run so that the intuition of insider attacker is exposed more clearly.

In practice the terminologies of "insider" and "outsider" are used to describe some real security cases. Despite some

difference of the details among cases, essentially the behind meaning is quite similar, So using this two terminologies here is appropriate.

Froschle also independently noticed that in the existing undecidability proofs the attacker is different from any regular agents in terms of initial knowledge, which is not very "natural". She described the notion of natural key policy, and the second open problem in [9] as follows.

> "A protocol has a *natural key policy* if in the specification of the initial key knowledge the Intruder is treated like all other principals."
>
> *Problem 2.* "Is Insecurity decidable for protocols with a natural key policy when the message size is bounded?"

Note that the existing undecidability results cannot cover the two open problems of [9], therefore their solutions are unknown.

Our reduction scheme using 2-counter machine is inspired from [21]. However, there are some key differences. First, the proofs in [21] assume non-matching RO protocols and an outsider attacker and we need new ideas of reduction to avoid the limitations. Furthermore, we have found and fixed two minor errors in the work of [21]: a counter can be negative, and zero can be used as a positive number. The proof of correctness of the reduction in [21] is sketchy and consequently misses the two errors. Details of the errors and our fixes are presented in the Appendix of [13]. We have adapted the approach of using 2-counter machine in [13] and [14] to prove the undecidability of an open problem pinpointed by [6]. Our work in [13] and [14] give us some confidence and motivation in finding a general and powerful proof scheme. However the proof in [13] also has the limitations of assuming non-matching RO protocols and an outsider attacker.

2. Notations and Modeling

In order to make the proof rigorous, the notions related to protocol, attacker, and protocol run must be established. These concepts have been extensively studied and are familiar to researchers. There are different formalization systems to describe a protocol and a protocol run assuming a Dolev-Yao attacker [4], such as (not a comprehensive list) the model defined by Paulson [20], Multi-set Rewriting (MSR) [6], Strand Space [11], and Constraint Solving [18]. While these models emphasize the behavior of the attacker and a protocol run, they may gloss over some details.

Due to limit of space, we only present the well-known key ideas of protocol, protocol run, terms, and the attacker, which should be enough for the proof. More details are presented in [15]. A reader familiar with modeling of Dolev-Yao attacker and protocol run can directly verify the correctness of the proof in Section 3 using their own modeling system, while paying attention to the design features that the attacker is an insider and the roles are matching.

Notations are chosen in a style similar to other papers, e.g., [16]. A ***variable*** is a symbol with at least one uppercase letter, such as N_A, $A2$, A. A ***constant*** is an atomic term with no uppercase letter, such as a, n_a, r_1.

A pair of asymmetric keys is represented as k_X^1 and k_X^0. X is the unique ID (UID) of the asymmetric key pair. If X is an agent name, k_X^1 is X's public key and usually should be initially known to other agents, and k_X^0 is X's private key that should be known only to X. This notation can also describe the asymmetric keys generated during a run.

A ***term*** is an atomic term (a variable or a constant), or a list, or an asymmetric encryption, or a symmetric encryption. A list has the form $[X, Y, \cdots]$, where X and Y are terms and the list contains finite number of terms. A list is a simpler representation of a sequence of nested pairs. For example $[W, X, Y, Z]$ is the same as $[W, [X, [Y, Z]]]$. A ***message*** is a term. When a message is a list, the top level enclosing [ ] is omitted. An asymmetric encryption has the form $\{T\}_{k_A^i}^{\rightarrow}$, $i \in \{0, 1\}$, where T is a term called the encrypted text, and k_A^i is the atomic encryption key. A symmetric encryption has the form $\{T\}_Y^{\leftrightarrows}$, where T is the encrypted text and Y is a term which is the encryption key. Messages in a run are ground (variables are instantiated).

A special constant is the upper case letter I, for intruder, which is reserved as the name of the attacker.

An ***action*** can be an ***internal action*** or an ***external action***. Let P, A, and B be agent names. An internal action of fresh term generation is denoted as $\#_P(t1, t2, \cdots)$, where "$t1, t2, \cdots$" represent the fresh terms generated by agent P before P sends a message that contains these fresh terms. A fresh term (such as a nonce) should be different from all other terms appeared in the run so far. 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 $-(B \Rightarrow A) : Msg$. In this paper the only kind of internal actions explicitly expressed in action code is fresh term generation.

An ***action step*** is a sequence of actions which can be considered as being executed together in a unit. 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 the other three can be executed by both regular agents and I. 4) is the only kind of action step that contains more than one action.

The well-known ***Dolev-Yao model*** [4] commonly refers to two assumptions: 1) a dominating attacker who records every message immediately when the message is sent, and

can prevent a message from being received, and can send any message it can obtain to any agent; 2) perfect encryption, which means no agent can decrypt a term unless the agent knows the decryption key. This paper assumes every agent (including the attacker) has equal power of generating unbounded fresh terms. When a fresh term is generated it is different from all other terms appeared in the run so far. We need unbounded nonces to address undecidability. Note that if only bounded many nonces can be generated while message size is bounded, there are bounded many terms appearing in the run and secrecy checking is decidable [6].

The attacker initially knows a set of terms, denoted as $I.init$, before a run of a protocol. The attacker's knowledge (the set of terms I can obtain) grows during a run. After a sequence E of action steps has been executed, the attacker's knowledge is denoted as $know_I(E)$, which includes: 1) all terms in $I.init$; 2) all messages that has been sent in E; 3) all fresh term X that have been generated by I in E, i.e., an action step of the form $\#_I(X, \cdots)$ is included in E; 4) an infinite set of terms described as the closure of applying a set of well known rules of term synthesis and analysis [20] to terms in $know_I(E)$. More details can be found in [13].

Roles usually contain variables since they are the templates of action steps. A role instance will replace all variables in a role by some ground terms.

A consensus of the different modeling systems, cited at the beginning of this section, is the follows. A protocol Pro can be analyzed as a set of roles. A run of Pro can be described a sequence E of actions steps associated with a set R of role instances executed by regular agents. E is formed by interleaving (prefixes of) role instances in R, provided that before every message Msg can be received by a regular agent after a certain point of the run, say after E' which is a prefix of E, the attacker I must be able to construct Msg, denoted as $Msg \in know_I(E')$. Secrecy checking for a protocol Pro is to check if there is a run E of Pro, such that after E, a secret term Sec can be leaked, or $Sec \in know_I(E)$. 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 that can describe them. It can be proved that for protocol analysis a run represented as bundle of role instances [11] is equivalent to a run represented as a trace.

A run should always be associated with a set AN of the names of *legitimate agents* (explained earlier), and a specific initial knowledge pattern of the attacker, call it D. If $I \in AN$, then I is an insider attacker. And D should be the same as the initial knowledge pattern of other regular agents in AN. Formally a run is a tuple (Pro, D, R, AN, E). Pro is the protocol. R is a set of role instances executed by regular agents. E is the trace of actions steps described earlier. A trace of a run is denoted as $run.E$.

The set of all possible runs of a protocol Pro with a specific initial knowledge pattern D of the attacker is denoted as $runs^{D:Pro}$. A ***secrecy checking*** problem can be described as follows. Given a protocol Pro and an initial knowledge pattern D of the attacker, and a set of secret term SEC, to check the validity of the following statement.

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

In addition to a set of (matching) roles, a protocol should specify the initial knowledge patterns of the agents of AN. The initial knowledge of an agent P is denoted as $P.init$.

Since we address the undecidability of analyzing a protocol, the number of role instances should not be bounded. It has been proved by [22] that if the number of role instances in a run is bounded, secrecy is decidable.

3. Proving Undecidability of Checking Secrecy and Authentication

We want to prove that, an arbitrary deterministic 2-counter machine [12] with no input (well-known concept, defined in Def. 1), call it M, can reach a final configuration, if and only if the corresponding protocol Pro has a secrecy attack. Pro is a matching RO protocol. The attack is a run of Pro where the attacker, who is an insider, finally knows some secret term. The secret term is declared by the same way N_b is declared as the secret term for the well-known attack to the PKNS protocol [16]. Since the 2-directional proof can be carried out in a straightforward style, we focus on presenting the "parameters" of the design of the proof, which are the encoding schemes and protocol design. The remaining technical details are included in [15].

Definition 1. *A* **deterministic 2-counter machine** *[12] 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, V_1, V_2), where q is the current state and V_1 and V_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 $(q, V_1, V_2) \longrightarrow^T (q', V_1', V_2')$, 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 $V_h = 0$. When $i_h = 1$, it means that $V_h > 0$. When $j_h = +1$ ($j_h = 0$, $j_h = -1$), it means that after the transition, $V_h' = V_h + 1$ ($V_h' = V_h$, $V_h' = V_h - 1$). 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 configura-*

tion $(q_{final}, \_, \_)$ 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.

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

Proof. Given a 2-counter machine $M = (Q, \delta)$, let $Q = \{q_0, q_{final}, q_1, q_2, \cdots, q_m\}$ and $\delta = \{T_1, T_2, \cdots, T_n\}$.

Encoding is a correspondence between a term and its designed meaning in a reduction. In the reductions of undecidability [3] [6] [21] and NP-hardness [22] [8] encoding is implemented by symmetric encryptions using some key shared by agents. Using shared symmetric key could make the proofs easier and we could also design the encoding scheme using symmetric keys. However we implement the encoding and design the protocols in the proof using only public keys and private keys, like the PKNS protocol, to make the proof more interesting and convincing.

A ***configuration term*** has the form $\{5, B, e, q, C_1, C_2\}^{\rightarrow}_{k^0_A}$, where A and B are two different regular agent names, which means $\{A, B\} \subset AN$, $A \neq I$, $B \neq I$, $A \neq B$. C_1 and C_2 could be any terms. e is a special constant used in the reduction as an evidence to distinguish a real configuration term from a term of the form $\{5, B, N_A, q, C_1, C_2\}^{\rightarrow}_{k^0_A}$, which can be easily obtained by the attacker, where N_A is a nonce generated by A in the first message of a role S_f (introduced later), $1 \leq f \leq n$. An honestly generated nonce N_A cannot be e, due to the assumption of unbounded fresh nonce generation. q is a constant that could be any state in Q. The regular agent name B included in a configuration term is also designed to prevent the I from getting a (final) configuration term trivially. Note that I can easily get a term $\{5, I, e, q_{final}, C_1, C_2\}^{\rightarrow}_{k^0_A}$, which is not a configuration term (I is inside, not B), when A plays the role S_{final} (introduced later) and A talks with I. Note that I cannot construct a configuration term since I does not know k^0_A. The only configuration term that I can trivially obtain is the starting configuration term $\{5, B, e, q_0, z, z\}^{\rightarrow}_{k^0_A}$, which is generated by a role instance of R_0 and corresponds to $(q_0, 0, 0)$. I has to do a run to simulate a computation of M in order to get the non-trivial configuration terms which include the final configuration terms.

A ***connection term*** has the form of $\{7, B, C_1, C_2\}^{\rightarrow}_{k^0_A}$, where $\{A, B\} \subset AN$, $A \neq B$, $A \neq I$, $B \neq I$. Connection terms are used to build the encoding of a counter value (described later). The encryptions containing constant 5 are distinguished from, and cannot be unified with, the encryptions containing constant 7. A ***secret term*** is a term N_B generated by a role instance of R_{final} (introduced later) which is executed by a regular agent B, $B \neq I$, where B tries to send N_B to agent A, $A \neq I$. The secret term is declared exactly in the same way as the PKNS protocol [16].

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

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

The set of roles of Pro is the follows.

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

Note that there are n transition rules in δ.

For each run of Pro, we consider I is an insider, so $I \in run.AN$ and that D is the same as the above initial knowledge pattern for other agents (instantiate P with I).

We describe a matching pair of roles S_i and R_i, $i \in \{0, 1, \cdots n, final\}$ by the CS between them. The action steps of two matching roles can be straightforwardly parsed from the CS between them. We choose different variable names in different matching pairs of roles so there is no confusion when the CSes of these pairs of roles are concatenated to form the CS of the whole protocol. Actually any interleaving of the CSes of the pairs of roles will be a corresponding CS of the protocol.

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

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

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

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

Call the two messages appearing in the above CS as $Msg1$ and $Msg2$. As an example of roles parsed from a CS, the two roles can be represented as follows.

S_{copy1}	R_{copy1}
1. $\#_{P1}(N1_{P1}, N2_{P1}, N3_{P1})$ + $(P1 \Rightarrow G1) : Msg1$	1. $-(P1 \Rightarrow G1) : Msg1$
2. $-(G1 \Rightarrow P1) : Msg2$	2. $+(G1 \Rightarrow P1) : Msg2$

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

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

The two roles S_{final} and R_{final} carry out an adjusted version of the PKNS protocol, executed by agents A and B respectively. The CS between A and B is the follows.

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

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

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

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

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

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

The counter value 0 must be represented by z. When a counter h should be positive, the term $\{7, B_f, Ch_f^{-1}, Ch_f\}^{\rightarrow}_{k^0_{A_f}}$ is needed in Msg_1, which shows that Ch_f^{-1} encodes a number one less than the number encoded by Ch_f. Verbose explanations of the rewrite rules are in [15].

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

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

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

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

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

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

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

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

The encoding of 0 is the constant z. So $0 = \underline{z}$. If a term $\{7, B, X, Y\}^{\rightarrow}_{k^0_A}$ appears in an encoding sequence, it means that $\underline{X} = \underline{Y} - 1$.

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

We prove this direction by constructing a run simulating $Comp$. Pro is the protocol just described. $run.AN = [I, a, b]$. Only three agents are enough here to instantiate the sender and receiver variables in each role instance. run

starts with a role instance of S_0 executed by a, to generate the initial configuration term $\{5, b, e, q_0, z, z\}_{k_a^0}^{\rightarrow}$. Then for each transition step of $Comp$, assume it applies a transition rule t, a role instance, say r, of the corresponding role R_f which is translated from t, is executed. r receives a configuration term encoding the previous configuration, and produce a configuration term encoding the next configuration of $Comp$. In addition, some role instances of R_{copy1} and R_{copy2} are executed to swap the positions of a and b in the newly generated configuration term and number connection terms, in order to maintain the encoding scheme. A lemma can be proved by induction on each step of $Comp$ to show that the attacker can always obtain the message needed for the next step of the attack. Finally, when the final configuration term is generated, the attacker uses it to carry out an attack between S_{final} and R_{final}, which is the same as the well-known attack to the PKNS protocol. Details are presented in [15].

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

The proof design is the follows. The only way the attacker can know n_b is by executing a PKNS attack. The only way for I to execute such an attack is to know a term $\{5, b, e, q_{final}, C1, C2\}_{k_a^0}^{\rightarrow}$. And the only way to know this term is to carry out a run of Pro which simulates a computation of M reaching a final configuration.

We can observe that in a run every term can encode at most one number, and 0 can only be encoded by z. We can prove a stronger lemma to show that every configuration term generated in a run of Pro encodes a configuration reachable to M. It is obvious that the secret term is leaked if and only if the attacker can obtain a final configuration term. Detailed proof of Direction 2 are presented in [15].

Since in a run we can consider message size (the number of atomic in a message) is bounded by 19, which is the maximum size of a message of a role R_f, $1 \leq f \leq n$, when every nonce variable is instantiated by a true nonce (an atomic term). Theorem 1 is proved. □

3.1. Undecidability of Authentication

Formal definition of authentication goals for security protocols have been discussed by by several papers. The most clarified and widely used definitions of authentication goals are presented by Lowe in [17] as the correspondence between role instances with shared data. The authentication goal of the PKNS protocol is discussed in [17] as follows.

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

Definition 2. *Given a protocol Pro as a communication sequence, and two roles $Role_1$ and $Role_2$ parsed from Pro. Suppose S is a set of parameters shared by $Role_1$ and $Role_2$ according to Pro. Assume S always include the two agent names who execute $Role_1$ and $Role_2$. Let M be the last message that is sent by $Role_2$ and received by $Role_1$ according to Pro. The authentication goal of $Role_1$ to authenticate $Role_2$, denoted as $Role_1 \rightarrow Role_2$, means that for a role instance r of $Role_1$ in a run, there is one and only one role instance r' of $Role_2$ in the run which has been executed up to M, and each variable in S is instantiated by the same value in r as in r'.*

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

Proof. The well-known attack to the PKNS protocol [16] is also an authentication attack, since the goal $Role_B \rightarrow Role_A$ is violated. The attacker can impersonate A when B believes the interlocutor is A. The attacker I can carry out the authentication attack if and only if I can obtain N_B. By the same reason, for the protocol designed in the proof of Theorem 1, the authentication goal $R_{final} \rightarrow S_{final}$ can be violated if and only if I can obtain N_B, and by the proof of Theorem 1, if and only if the 2-counter machine can reach a final configuration. □

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

4. Summary

We directly address the secrecy and authentication goals of the PKNS protocol and prove the undecidability of checking secrecy and authentication. To our best knowledge these undecidability results are the first that can be directly applied to protocols common in literature (matching RO protocols) while considering the attacker as an insider. This research solves the two open problems posted by [9].

Our experiences show that to prove the undecidability of a new problem of checking cryptographic protocols, there are reusable proof patterns. The modeling of protocols and protocol runs could be the same. The two-directional proof could be very similar, including the lemmas and observations, and could be executed as routines. The design of encoding and protocols in the reduction can follow some common high-level ideas, such as using a "chain" of encrypted terms to encode a natural number. The encoding design can be very flexible, which is a source of the power of this approach, as long as the high-level ideas can be implemented. For example, a number could be encoded by a composite term, not necessarily by an atomic nonce. Also in an encoding sequence different encryption keys can be used, not necessarily only one encryption key is allowed.

By the above observations, for the purpose of improving techniques for proving undecidability of checking cryptographic protocols and similar reductions, we consider that the techniques of artificial intelligence and automatic theorem proving which utilize reusable solution patterns and experiences of cases are potentially very helpful.

References

[1] R. M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1):695–740, 2003.

[2] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Technical report, Department of Computer Science, University of York, UK, 1997.

[3] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. Technical Report LSV-01-13, Laboratoire Spécification et Vérification, ENS Cachan, France, `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PS/rr-lsv-2001-13.rr.ps`, December 2001. 98 pages.

[4] D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[5] N. A. Durgin. *Logical Analysis and Complexity of Security Protocols*. PhD thesis, Computer Science Department, Stanford University, March 2003.

[6] N. A. Durgin, P. Lincoln, and J. C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[7] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 34–39, 1983.

[8] 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*, pages 185–211. IOS Press, 2006.

[9] S. Froschle. The insecurity problem: Tackling unbounded data. In *IEEE Computer Security Foundations Symposium 2007*, pages 370–384. IEEE Computer Society, 2007.

[10] D. Gollmann. Insider fraud (position paper). In B. Christianson, B. Crispo, W. S. Harbison, and M. Roe, editors, *Security Protocols Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 213–219. Springer, 1998.

[11] J. D. Guttman, F. J. Thayer, and J. C. Herzog. Strand spaces: Why is a security protocol correct? In *1998 IEEE Symposium on Security and Privacy*, pages 160–171, May 1998.

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

[13] Z. Liang and R. M. Verma. Secrecy Checking of Protocols: Solution of an Open Problem. Technical report, Computer Science Department, University of Houston, Texas, USA, `http://www.cs.uh.edu/preprint`, April 2007. UH-CS-07-04.

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

[15] Z. Liang and R. M. Verma. Improving Techniques for Proving Undecidability of Checking Cryptographic Protocols. Technical report, Computer Science Department, University of Houston, Texas, USA, `http://www.cs.uh.edu/preprint` or `http://www.cs.uh.edu/~zliang`, January 2008.

[16] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *TACAS*, pages 147–166, 1996.

[17] G. Lowe. A hierarchy of authentication specifications. In *CSFW '97: Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, page 31, Washington, DC, USA, 1997. IEEE Computer Society.

[18] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[19] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, 1978.

[20] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[21] R. Ramanujam and S. P. Suresh. Undecidability of secrecy for security protocols. Manuscript, July 2003.

[22] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theor. Comput. Sci.*, 1-3(299):451–475, 2003.