Process Calculus and Horn Clauses-Based Deduction in the Verification of Security Protocols

Nguyen Truong Thang†, Khuat Thanh Son†
ntthang@ioit.ac.vn ktson@ioit.ac.vn
† Institution of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam

Summary:
In security protocols, there is a well-known theorem about the undecidability of security properties such as secrecy and authentication. That is, it is not possible to develop an automatic procedure that in finite time and space always decides correctly whether or not a security property holds for any given protocol model. With respect to the importance of security protocols in the Internet nowadays, to deal with this limitation, several approaches imposing assumptions in the model are presented in order to make protocol design more reliable. These approaches are classified into relaxing some restrictions on the theorem requirements. This paper presents in details a notable technique, namely the combination of process calculus and Horn clauses-based reasoning in an automatic decision procedure. In particular, the characteristics of process calculus for modeling and Horn clauses for automatic verification are presented. Some strengths and weaknesses of this combination compared with other approaches are also given. This combination technique is implemented in ProVerif an automatic symbolic verifier of security protocols.

Key words:
Undecidability, security protocols, process calculus, Horn clauses, formal verification.

1. Introduction

Security protocols are programs that aim at securing communications on insecure networks, such as Internet, by relying on cryptographic primitives. For instance, in practice, security protocols are usually used to establish shared key between parties. This shared key is sent over hostile networks under the asymmetric encryption scheme among involving parties (or principals). Once the shared key has been exchanged, a session of symmetrically encrypted data transmission can be executed between principals. When the data session is completed, the shared key is also discarded. However, public and private keys associated with a principal remain for future run of the security protocol.

With the evolution of Internet, security protocols are becoming more important than ever as our lives are more dependent on Internet and its reliability. However, the design of security protocols is errorprone, e.g. the notable flaw in the very famous Needham-Schroeder public-key protocol was found 17 years after the protocol publication [1]. An automatic decision procedure with regards to a security protocol is very much needed. Unfortunately, there is a well-known theorem about the undecidability of security properties such as secrecy and authentication. That is, it is not possible to develop an automatic procedure that in finite time and space always decides correctly whether or not a security property holds for any given protocol model. This undecidability problem is fundamental in protocol design and verification. To overcome this limitation, some further restrictions to the model are imposed. These can be grouped into 3 model restriction types. They are: (1) bounding number of finite sessions to make a finite-state model; (2) accepting semi-automatic decision procedure via interactive verification process; and (3) allowing semi-decision procedure which is automatic but may not terminate or may terminate with uncertain result.

This paper presents a notable technique in the 3rd approach, i.e. overestimating the possibilities of attacks. This is done most of the time by computing an over-approximation of the intruder knowledge. The Horn clause approach is one such method and Horn clauses are at the basis of the automatic protocol verifier ProVerif [2]. The paper discusses the rationale of the selected technique, characteristics deciding strengths and weaknesses of the Horn clauses-based deduction compared with other techniques. The paper is structured as follows. Section II presents briefly about formal modeling and security properties of protocols, the theorem of undecidability issue. Section III is about some approaches to deal with the undecidability problem by imposing some restrictions in protocol model and the choice of the Horn clauses-based method. The method is selected in this section due to its inherent strength in an automatic decision procedure about a given process calculus-based protocol model. Compared with other techniques, the most significant contribution of the method is on the decidability with respect to protocol models while keeping unbounded number of sessions. However, this method has to pay the price, i.e. the overapproximation can ensure the soundness of the procedure but not its completeness. It is sound, in the sense that if the procedure does not find a flaw in the protocol, then there is no flaw. The procedure therefore provides real security guarantees. In contrast, it may give a false attack against the protocol. Section IV discusses key points in protocol modeling in the method via applied

Manuscript received May 5, 2020
Manuscript revised May 20, 2020

π-calculus [3]. Later, Section V goes to the theoretical aspects of Horn clauses-based deduction. An example of protocol verification in the method is shown in Section VI. Some discussion about the inherent strength and weakness of the method is also presented.

2. Technology Overview

A. Formal Model of Security Protocol

In order to obtain proofs that security protocols are correct, the first job is to model them mathematically. Along this research line, two model types of protocols have been considered [1].

First, in the symbolic model, aka. Dolev-Yao model, the cryptographic primitives are represented by function symbols considered as black boxes, the messages are terms on these primitives, and the intruder is restricted to compute only using these primitives. This model assumes perfect cryptography. Perfect encryption means that, one can decrypt a message only when he has the corresponding key. Second, in the computational model, the messages are bitstrings, the cryptographic primitives are functions from bitstrings to bitstrings. Encryption is not perfect as the intruder's decrypting capability is represented by a probabilistic function. A security property is considered to hold when the probability that it does not hold is negligible [1]. This paper considers symbolic model only.

Next is about security properties ensured by protocols. Among them, secrecy and authentication are the most important. Secrecy means that the intruder cannot obtain some information on data manipulated by the protocol. In the symbolic model, secrecy can be more often formalized as the fact that the intruder cannot compute exactly the considered piece of data. Authentication means that, if an agent A runs the protocol apparently with an agent B, then B runs the protocol apparently with A, and conversely. In general, one also requires that A and B share the same values of the parameters of the protocol.

B. Undecidability Theorem

It is proved that there exists no algorithm to decide on a property with regards to an input security protocol. That is, the algorithm can not confirm whether the protocol is secure or not. And if not, it shows an attack scenario. This leads to the fundamental undecidability theorem in security protocols. Both secrecy and authentication among all the interesting properties are undecidable in the symbolic model [4], [5], [6]. More details on the rationale of undecidability are given below.

Despite this limitation, there exist semi-decision techniques which can be automated in various ways. First, observe that it is possible to design an algorithm which will always find an attack by an intruder in finite time if an attack exists and may not terminate if the protocol is correct. This can be done by simply enumerating all traces of the protocol's state transition system. Then, in each state, it can be decided if secrecy has been violated. Other semidecision techniques, e.g. the inductive method [7], correspond to the fact that the algorithm may not provide a definite answer.

There are several sources of undecidability. First, the protocol itself can simulate one step of computation for a universal computation model: each state of the machine is an agent who, upon reception of a configuration, sends the next configuration to the appropriate state. The intruder only has to bridge two successive sessions forwarding the last message of one session to the appropriate principal as the first message of the next session. The overall effect is a non-terminating protocol session as many single sessions are concatenated together due to the intruder's attack. With regards to this non-terminating sessions, the decision procedure is not decidable. That is why decision methods have to either impose a bound on the number of instances [5], or restrict manipulation of the messages (e.g., impose a "single reference to previous messages" restriction [6]). The second source of undecidability is the ability to generate nonces, which may be used, roughly, to simulate arbitrarily many memory locations and therefore encode machines with unbounded memory [4]. Again, if the number of protocol instances is bounded in advance, this cannot occur. In fact, it is sufficient to bound the total number of nonces which are generated in any trace. Even if it is assumed that there is a bounded number of instances, it is not yet easy to design a decision algorithm since, according to the Dolev-Yao model, the intruder still has an unbounded number of possible choices at any point. In particular, the number of messages that can be created by the intruder is unbounded. An additional restriction bounding the intruder's memory allows development of finite-state decision techniques.

3. Over-Approximation Of The Intruder For The Decidability Of Security Protocols

As mentioned in Section II-B, there is no decision procedure giving definite answer on whether a property holds in a general security protocol due to the infinite space constructed from the general protocol parameters. Typical protocol parameters are:

1) The number of involving principals;
2) The number protocol sessions which could be executed concurrently;
3) Protocol depth or number of steps within the input protocol;
4) The maximum data size within a message transmitted in the protocol;

5) The number of times each cryptographic operation can be applied by the intruder (or attacker);

Certainly, if these parameters are unbounded, the search space should be infinite and that is the rationale for the undecidability theorem. Some techniques are required to impose restriction on the general models at some aspects. Even the protocol depth (3rd parameter), data size (4th parameter) and set of possible principals (1st parameter) are finite, the secrecy and authentication properties are still undecidable [4]. Indeed, 2nd and 5th parameters are the most important elements in security protocol analysis. Many research lines only focus on handling these two parameters. Hence, the decidable classes of protocols are further restricted. Note that the 2nd parameter is about concurrent protocol sessions, while the 5th parameter represents the intruder in terms of traffic analysis and message forging synthesis.

A first group of techniques tries to bound the number of sessions (2nd parameter) and the capability of the intruder (5th parameter). The model becomes finite and a finite-state analysis on the model can yield a definite response, i.e. decidable. The finite-state analysis in this group is often done by model checking [8].

Within these parameters, the number of protocol sessions (2nd parameter) is always the most challenging for protocol analysis. When the number of executions of the protocol is not bounded, the problem is undecidable for a reasonable model of protocols. Hence, there exists no automatic tool that always terminates and solves this problem. However, there are several approaches that can tackle an undecidable problem:

1) Relying on interactive help from the user. This is done for example using the interactive theorem prover Isabelle as inductive method [7] which just requires the user to give a few lemmas to help the tool. Here, we lose the automatic mechanism of the procedure.
2) Allowing non-terminating or incomplete tools, which sometimes answer "I do not know" but succeed on many practical examples. For instance, one can use abstractions based on tree-automata to represent the knowledge of the intruder. Horn clauses approach and its tool - ProVerif belong to this approach [2].

ProVerif uses an abstract representation of protocols by Horn clauses which is more precise than tree-automata because it keeps relational information on messages. However, using this approach, termination is not guaranteed in general.

4. Process Calculus-Based Protocol Modeling

A. Applied π-Calculus

The applied π-calculus is a language for describing concurrent processes and their interactions. It is based on the π-calculus but is specifically targeted at modeling security protocols.

The calculus assumes an infinite set of names, an infinite set of variables, and a signature Σ consisting of a finite set of function symbols each with an associated arity. A function symbol with arity 0 is a constant. Terms are built by applying function symbols to names, variables and other terms. A term is ground when it does not contain variables.

L,M,N ::=	terms
a,b,c	name
x,y,z	variable
g(M1,...,Ml)	function application

where g ranges over the functions of signature Σ and l is the arity of g.

The grammar for processes is shown below:

$P,Q,R ::=$	[plain] processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu\, n.P$	name restriction
$\nu\, x.P$	variable restriction
$\{M/x\}$	active substitution
if $M = N$ then P else Q	conditional
$u(x) : P$	message input
$\bar{u}\langle M\rangle : P$	message output

The null process 0 does nothing; P |Q is the parallel composition of processes P and Q, used to represent principals of a protocol running in parallel; and replication !P is the infinite composition P |P |..., which is often used to capture an unbounded number of sessions. Name restriction ν n.P binds n inside P, the introduction of restricted names (or private names) is useful to capture both fresh random numbers (modeling nonces and keys, for example) and private channels. The conditional if M = N then P else Q is standard. Notice that M = N represents equality (modulo an equational theory) rather than strict syntactic identity. Finally, communication is captured by message input and message output. The process u(x) : P waits for a message from channel u and then behaves as P with the received message bound to the variable x; that is, every free occurrence of x in P refers to the message received. The process $\bar{u}$(M) : P is ready to send M on channel u and then run P. P{M/x} stands for P with all

free occurrences of x replaced by M and it is similar to the notation let x = M in P.

B. Protocol Modeling

From components defined in Section IV-A, security protocols are defined via a sequence of message exchanges. A message X is sent from principal A to principal B with the standard notation A → B: X. Asymmetric data encryption of message m via public key of S is shown by {m}pkS, while signature of S for the message m is {m}pkS, On the other hand, symmetric data exchange of m via shared key k is [m]k. As an illustrated example, a simple handshake protocol is used. This protocol is to generate a fresh shared key k used for a data session between C - a client and S - a server under the public key scheme. It is assumed that each of them has a public/private key pair, and that the client knows the server's public key pkS. The aim of the protocol is to establish a secret symmetric key k, enabling the client to communicate a secret s to the server. Once the data session is over, k is discarded while the public/private keys associated with C and S remain for future protocol run. The simplified protocol is shown below:

- Step 1. $S \rightarrow C : \{\{k\}_{sk_S}\}_{pk_C}$
- Step 2. $C \rightarrow S : [s]_k$

Informally, the protocol proceeds as follows. On request from a client C, server S generates a fresh session key k, signs it with her private key skS and encrypts it using her client's public key pkC. When C receives this message he decrypts it using his private key skC, verifies the signature made by S using her public key pkS, and extracts the session key k. C uses this key to symmetrically encrypt the secret s and sends the encrypted message to S. The rationale behind the protocol is that C receives the signature asymmetrically encrypted with his public key and hence he should be the only one able to decrypt its content. Moreover, the signature should ensure that S is the originator of the message.

Informally, the three basic security properties this protocol should provide are:

1) Secrecy: The value s is known only to C and S.
2) Authentication of S: if C reaches the end of the protocol with session key k, then S proposed k for use by C.
3) Authentication of C: if S reaches the end of the protocol and believes session key k has been shared with C, then C was indeed her counterpart and has k.

Careful analysis reveals that the protocol does not satisfy all three of the intended properties. Verification by ProVerif is shown in Section VI. This part only explains informally the way protocol could be attacked. If an intruder I starts a session with S, then I is able to impersonate S in a subsequent session he starts with C. At the end of the protocol C believes that he shares the secret s with S, while he actually shares s with I.

- Step 1. $S \rightarrow I : \{\{k\}_{sk_S}\}_{pk_I}$
- Step 1'. $I \rightarrow C : \{\{k\}_{sk_S}\}_{pk_C}$
- Step 2. $C \rightarrow I : [s]_k$

The protocol can easily be corrected by adding the identities of the intended agents to the data that is signed in the first message:

- Step 1. $S \rightarrow C : \{\{pk_S, pk_C, k\}_{sk_S}\}_{pk_C}$
- Step 2. $C \rightarrow S : [s]_k$

With this correction, I is not able to re-use the signed key from S in his session with C.

The simple handshake protocol is defined with respect to the signature Σ, which is used to capture primitives modeling cryptographic operators, data structures and constants: Σ = {true,fst,snd,hash,pk,getmsg,pair,sdec,senc,adec,aenc,sign,chksign,mac} where true is a constant; fst,snd,hash,pk, getmsg are unary functions; and pair,sdec,senc,adec,aenc,sign,chksign,mac are binary functions. The behavior of these functions is captured by a simple equational theory E satisfying the following equations over variables x,y:

fst(pair(x,y))	= *x* - projection
snd(pair(x,y))	= *y* - projection
sdec(x,senc(x,y))	= *y* - symmetric
adec(x,aenc(pk(x),y))	= *y* - asymmetric
getmsg(sign(x,y))	= *y* - signature
chksign(pk(x),sign(x,y))	= true

For example, the application of the symmetric decryption function sdec to the term modeling a symmetric encryption senc(k,m) should return the plaintext m if the correct key k is supplied, i.e. the equation sdec(x,senc(x,y)) = y. The handshake protocol can now be captured in our calculus as the process P, defined as follows.

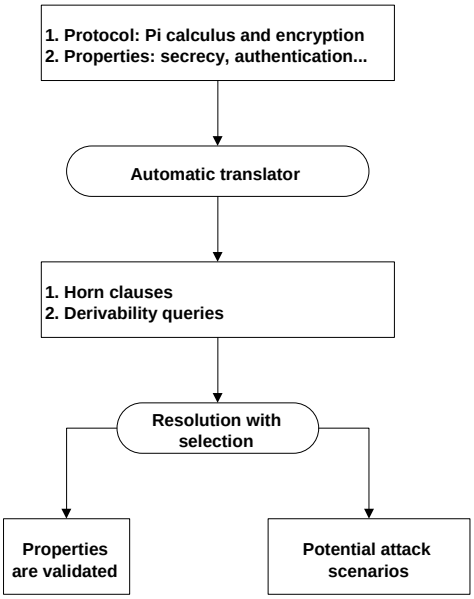
$$
\begin{aligned}
P \triangleq\ & \nu\, sk_S.\nu\, sk_C.\nu\, s \\
& \text{let } pk_S = \mathtt{pk}(sk_S) \text{ in let } pk_C = \mathtt{pk}(sk_C \\
& \quad c(pk_S) \mid \bar{c}(pk_C) \mid !P_S \mid !P_C) \\
P_S \triangleq\ & c(x_pk).\nu\, k.\bar{c}\langle \mathtt{aenc}(x\_pk, \mathtt{sign}(sk_S, k))\rangle \\
& c(z).\text{if } \mathtt{fst}(\mathtt{sdec}(k, z)) = tag \text{ then } Q \\
P_C \triangleq\ & c(y).\text{let } y' = \mathtt{adec}(sk_C, y) \text{ in} \\
& \text{let } y\_k = \mathtt{getmsg}(y') \text{ in} \\
& \text{if } \mathtt{chksign}(pk_S, y') = \mathtt{true} \text{ then} \\
& \bar{c}\langle \mathtt{senc}(y\_k, \mathtt{pair}(tag, s))\rangle
\end{aligned}
$$

The process begins by constructing the private keys skC, skS for principals C, S respectively. The public key parts pkC, pkS are then output on the public communication

channel c, ensuring they are available to the intruder. The protocol then launches multiple copies of processes PC, PS representing multiple sessions of the roles of C and S. Indeed, the sessions are unbounded. Note that syntactic scope does not represent the knowledge of a protocol's participants. For example, the server's private key skS is assumed not to be known by the client C (hence it does not occur in PC), even though skS is in the scope of PC.
We assume that S is willing to run the protocol with any other principal; the choice of her counterpart will be made by the environment. This is captured by modeling the first input c(x_pk) to PS as the counterpart's public key. C on the other hand only wishes to share his secret s with S, and C is assumed to know S's public key; accordingly, S's public key is hard-coded into the process PC. We additionally assume that each principal is willing to engage in an unbounded number of sessions and hence PC, PS are under replication.
On request from her counterpart (or interlocutor), server S starts the protocol by selecting key k and outputting aenc(x_pk,sign(skS,k)), that is, her signature on the key k encrypted with her interlocutor's public key x_pk. Meanwhile C waits for the input of his counterpart's signature on the key k encrypted using his public key. C decrypts the message and verifies the signature. Next, if C believes he is indeed talking to S, he outputs his secret s encrypted with the symmetric key k. Note that he inserts a tag (modeled as a free name), so that the decryptor can verify that the decryption has worked correctly. Principal S inputs z and confirms the presence of the tag. Finally, principal S executes the process Q. The description of Q is independent of the protocol, but one would expect the recovery of the counterpart's secret; that is, Q is defined by the process let z_s = snd(sdec(k,z)) in Q′ for some Q′.
The purpose of the protocol is to establish a session key to transport the secret represented by . We abstract away from the details of what s is, and how many such secrets there are, by modeling it simply as a restricted name.

5. Theoretical Foundation In Protocol Verification

Horn clauses are used the basis for deduction system of ProVerif (shown in Figure 1). All primitives, intruder's initial knowledge as well as its reasoning via the protocol model are all translated into Horn clauses for later automatic deduction.



Hình 1: The structure of ProVerif

In Figure 1, the protocol is formally modeled in process calculus, i.e. applied π-calculus as presented in Section IV-A. The properties are also specified informally. Next, the model and properties are automatically translated into equivalent Horn clauses and derivability queries respectively. These clauses are used to validate queries about properties. In this Horn clauses-based deduction system, if the properties holds, they are validated. Otherwise, potential attacks could be shown as evidences of protocol vulnerability. This paper does not go into details of resolution algorithms of the approach, particularly those implemented in ProVerif. Rather, we focus on the overall approach via process calculus modeling and Horn clauses-based deduction.

A. Protocol Primitives

Cryptographic primitives are represented by functions. For instance, we represent the publickey encryption by a function aenc(m,pk), which takes two arguments: the message m to encrypt and the public key pk. There is a function pk that builds the public key from the secret key. The secret key is represented by a name that has no arguments skS[ ] for S and skC[ ] for C. Then pkS = pk(skS[ ]) and pkC = pk(skC[ ]).
There are two kinds of functions: constructors and destructors. The constructors are the functions that explicitly appear in the terms that represent messages. For instance, aenc and pk are constructors. Destructors manipulate terms. For instance, the decryption adec is a destructor, defined by adec(aenc(m,pk(sk)),sk) = m. This rewrite rule models that, by decrypting a ciphertext with the corresponding secret key, one obtains the cleartext. Other functions are defined similarly.

B. Protocol Primitives

We assume that the protocol is executed in the presence of an intruder that can intercept all messages, compute new messages from the messages it has received, and send any message it can build. During its computations, the intruder can apply all constructors and destructors. If f is a constructor of arity n, this leads to the general clause form: intruder(x1) ∧ ... ∧ intruder(xn) = ⇒ intruder(f(x1,...,xn)). On the contrary, if g is a destructor, for each rewrite rule g(M1,...,Mn) −→ M in def(g), the general clause is: intruder(M1) ∧ ... ∧ intruder(Mn) = ⇒ intruder(M).

The destructors never appear in the clauses, they are coded by pattern-matching on their parameters in the hypothesis of the clause and generating their result in the conclusion. For example, in the case of public-key encryption, this yields:

- $intruder(m) \wedge intruder(\text{pk}) = \Rightarrow intruder(\text{aenc}(m,\text{pk}))$
- $intruder(sk) = \Rightarrow intruder(\text{pk}(sk))$
- $intruder(\text{aenc}(m,\text{pk}(sk))) \wedge intruder(sk) = \Rightarrow intruder(m)$

Where the first two clauses correspond to the constructors aenc and pk, and the last clause corresponds to the destructor adec showing that m is known to the intruder.

C. Protocol Primitives

This section describes how the protocol itself is represented by Horn clauses. This is the theoretical foundation for the translation from applied πcalculus into equivalent Horn clauses.

We consider that S and C can talk not only to any honest principal, but also malicious principals that are represented by the intruder. Therefore, the first message sent by S can be aenc(pk(x),sign(skS[ ],k)) for any x. We leave to the intruder the task of starting the protocol with the principal it wants, that is, the intruder will send a preliminary message to S, mentioning the public key of the principal with which S should talk.

This principal can be C, or another principal represented by the intruder. Hence, if the intruder has some key pk(x), it can send pk(x) to S. S replies with his first message, which the intruder can intercept, so the intruder obtains aenc(pk(x),sign(skS[ ],k)). Therefore, we have a clause of the form: intruder(pk(x)) = ⇒ intruder(aenc(pk(x),sign(skS[ ],k))).

Moreover, a new key k is created each time the protocol is run. Hence, if two different keys pk(x) are received by S, the generated keys k are certainly different: k depends on pk(x). The clause becomes: intruder(pk(x)) = ⇒ intruder(aenc(pk(x),sign(k[pk(x)],skA[ ]))). When C receives a message, he decrypts it with his secret key skC, so C expects a message of the form aenc(pk(skB[ ]),x′). Next, C tests whether S has signed x′, that is, C evaluates chksign(pkS,x′), and this succeeds only when x′ = sign(skS[ ],y). If so, he assumes that the key y is only known by S, and sends a secret s (a constant that the intruder does not have a priori) encrypted under y. We assume that the intruder relays the message coming from S, and intercepts the message sent by C. Hence the clause: intruder(aenc(pk(skC[ ]),sign(skS[ ],y))) = ⇒ intruder(senc(y,s)).

D. Protocol Primitives

A protocol can be represented by three sets of Horn clauses, as detailed below in the context of handshake protocol.

- Clauses representing the computation abilities of the intruder: constructors, destructors, and name generation (Section V-A).
- Facts corresponding to the initial knowledge of the intruder. In general, there are facts giving the public keys of the participants and/or their names to the intruder (Section V-B).
- Clauses representing the messages of the protocol itself. There is one set of clauses for each message in the protocol. In the set corresponding to the *i*-th message, sent by principal *X*, the clauses are of the form $intruder(M_{j1} \wedge \ldots \wedge intruder(M_{jn}) = \Rightarrow intruder(M_i)$ where $M_{j1},\ldots,M_{jn}$ are the patterns of the messages received by *X* before sending the *i*-th message, and M_i is the pattern of the *i*-th message (Section V-C).

This Horn clause-based representation of protocols is approximate. Specifically, the number of repetitions of each action (i.e. a message exchange/action) is ignored, since Horn clauses can be applied any number of times. So a step of the protocol (i.e. a message transmission) can be completed several times (instead of once as in the protocol model), as long as the previous steps have been completed at least once between the same principals. This is called over-estimation of intruder (or environment) capability.

However, the important point is that the approximations are sound: if an attack exists in a more precise model, such as the applied π-calculus, then it also exists in this representation. Performing approximations allows a much more efficient verifier, which will be able to handle larger and more complex protocols. Another advantage is that

the verifier does not have to limit the number of runs of the protocol - unbounded number of sessions.

7. Conclusion

On the hostile communication environment like Internet, this paper presents a technique based on strong theoretical foundation to deal with the decidability of security protocols. Due to the undecidability theorem, there are some approaches to overcome the limitation by dealing with fundamental point in the theorem. This paper attempts to make definite verification result with regards to any given security protocol via semi-decision procedure (sometimes the results are not certain, i.e. false attacks). The procedure is based on the applied π-calculus modeling and Horn clauses-based deduction. By accepting sound abstraction, the technique can handle unbounded protocol sessions, which means to analyze infinite-state space. On the other hand, the method ignores repetitions of computation from the intruder's viewpoint. This is the cause for the method's incompleteness. The model and property verification in the proposed method are well defined and supported by ProVerif. A simplified handshake protocol is taken as an illustrated example of the method.

Acknowledgement

We would like to thank CS'20.15 topic: " Research collaborative filtering models to suggest users to apply the emotional scoring method", Institute of Information Technology, Vietnam Academy of Science and Technology that has provided funding for the study.

References

[1] B. Blanchet, "Security protocol verification: Symbolic and computational models," in Principles of Security and Trust - First International Conference, POST 2012, Volume 7215 of Lecture Notes in Computer Science. Springer, 2012, pp. 3–29.
[2] B. Blanchet, B. Smyth, and V. Cheval, ProVerif 1.91: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial, 2015.
[3] M. D. Ryan and B. Smyth, "Applied pi calculus," 2011.
[4] D. L. Mitchell, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov, "Undecidability of bounded security protocols," 1999.
[5] R. M. Amadio, R. M. Amadio, D. Lugiez, and D. Lugiez, "On the reachability problem in cryptographic protocols." Springer-Verlag, 2000, pp. 380– 394.
[6] H. Comon, V. Cortier, and J. Mitchell, Tree Automata with One Memory, Set Constraints, and Ping-Pong Protocols. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 682–693.
[7] L. C. Paulson, "The inductive approach to verifying cryptographic protocols," Journal of Computer Security, 1998.
[8] H. Comon and V. Shmatikov, "Is it possible to decide whether a cryptographic protocol is secure or not?" 2001.
[9] B. Blanchet, "Using Horn clauses for analyzing security protocols," in Formal Models and Techniques for Analyzing Security Protocols, ser. Cryptology and Information Security Series, V. Cortier and S. Kremer, Eds. IOS Press, 2011, vol. 5, pp. 86–111.

Truong- Thang Nguyen received a Ph.D. in 2005 at the Japan Advanced Institute of Science and Technology (JAIST), Japan. Currently working at the Institute of Information Technology, Vietnam Academy of Science and Technology. Research fields: software quality assurance, software verification, program analysis.

Thanh- Son Khuat received the B.S. degrees in Information Technology from University of Engeneering and Technology, Vietnam National University in 2016, respectively. During 2016-2017, he stayed in Samsung Vietnam Mobile R&D Centre, Samsung Electronic Vietnam, to study mobile and application for samsung mobile. Currently working at the Institute of Information and Technology, VietNam Academy of Science and Technology. Research fields: software quality assurance, software verification, program analysis.