On-the-fly Trace Generation and Textual Trace Analysis and their applications to the analysis of Cryptographic Protocols

Yongyuth Permpoontanalarp

Logic and Security Laboratory
Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi, Bangkok, Thailand
yongyuth.per@kmutt.ac.th

Abstract. Many model checking methods have been developed and applied to analyze cryptographic protocols. Most of them can analyze only one attack trace of a found attack. In this paper, we propose a very simple but practical model checking methodology for the analysis of cryptographic protocols. Our methodology offers an efficient analysis of all attack traces for each found attack, and is independent to model checking tools. It contains two novel techniques which are on-the-fly trace generation and textual trace analysis. In addition, we apply our method to two case studies which are TMN authenticated key exchanged protocol and Micali's contract signing protocol. Surprisingly, it turns out that our simple method is very efficient when the numbers of traces and states are large. Also, we found many new attacks in those protocols.

Keywords: Formal methods for cryptographic protocols, Model checking, Cryptographic protocols.

1 Introduction

Cryptographic protocols are protocols which use cryptographic techniques to achieve certain tasks while preventing malicious parties to attack the protocols. There are many applications of cryptographic protocols, for example, authenticated key exchange protocols, web security protocols, e-payment protocols, e-banking protocols, e-voting protocols, etc.

The design and analysis of cryptographic protocols are difficult to achieve because of the increasingly attacking capabilities and the complex requirement of the applications. Attacks in many cryptographic protocols have been found later after they have been designed [1, 2] and even implemented eg. [3, 4]. Thus, it requires a method to analyze all possible attacks to the protocols. Such kind of method would offer a comprehensive understanding of all vulnerabilities of protocols and certainly would help in developing a better protection for them. Note that in this paper we focus on only message replay attacks [5].

Many model checking methods [6-16] have been developed and applied to analyze cryptographic protocols. Most of them can analyze only one attack trace of a found attack. In fact, all of them employ *off-the-fly* trace generation technique. It means that after a state space is generated either partially or fully and an attack state is found, an attack trace is then computed. This kind of trace generation is called *off-the-fly* since the trace computation occurs after the state space is generated. An attack trace is constructed by searching for a path from an initial state to an attack state. Since the searching for all paths between two states is extremely time-consuming, only one path is searched, instead, in most methods. However, the analysis of single attack trace is rather limited since one path or one attack trace represents only one way amongst many possible ways to carry out an attack. In addition, after one attack trace is found, a *visualization* technique is normally employed to illustrate and analyze the attack, for example, message sequence charts [31] and graphs. Such visualization technique can provide an intuitive analysis of a single attack trace. However, when the number of traces is large, for example thousands, it is hard to analyze them by visualizing, eg. to classify them into groups.

In this paper, we propose a very simple but practical model checking methodology for the analysis of cryptographic protocols. Our methodology offers an efficient analysis of all attack traces for each found attack, and is independent to model checking tools. The study of all attack traces is beneficial since it offers a deep understanding on all attackers' capabilities to compromise a system. Our method contains two novel techniques which are *on-the-fly* trace generation and *textual* trace analysis. In our method, while a state space is generated, attack traces for states are computed at the same time and stored at the states themselves. We call it *on-the-fly* trace generation since the trace computation occurs at the same time as the state space computation. Thus, after the whole state space is computed and an attack is found, then attack traces for the attack can be extracted from attack states of the attack immediately. Thus, all attack traces can be computed very efficiently without any path searching. This technique provides a big improvement in the computation time for all attack traces when the number of attack traces and the number of states are large.

The number of attack traces obtained can be quite large. For example, we found 1,020 traces for an attack in the TMN protocol. So, we propose *textual* trace analysis technique to classify such large number of attack traces. Those attack traces are classified by using attack patterns. Attack patterns are minimal but necessary protocol traces for an attack. Attack traces that contain the same attack pattern are classified into the same group of attack traces. While the development of an attack pattern is manual, the attack classification is automatic. By using our two new techniques, protocol designers could obtain a deep and thorough analysis of all possible attacks to cryptographic protocols.

To demonstrate the practical uses of our approach, we apply our new methodology to two case studies which are Micali's contract signing protocol (ECS1) [17] and TMN authenticated key exchange protocol [18]. We implement our methodology in a model checker tool called CPNTools [19,20]. Note that CPNTools originally provides the *off-the-fly* trace generation only. Then, we compare the results between our *on-the-fly* trace generation and the *off-the-fly* trace generation both in CPNTools. Surprisingly, it turns out that our *on-the-fly* trace generation is more efficient than the

off-the-fly trace generation when the numbers of traces and states are large. For TMN, our result shows that when the numbers of states and traces are 74,244 and 13,056, respectively, our method improves on the computation times for 6,777 %. For ECS1, our result shows that when the numbers of states and traces are 235,564 and 7,032, respectively, our method improves on the computation times for 116.75 %.

Because our method can analyze all attack traces, we found many new attacks in the two protocols in our previous works [21-24]. For Micali's contract signing protocol, we found one new single-session attack [21] and two new multi-session attacks [22]. Also, we found three new attacks [22] of Bao et. al.'s modified version of ECS1 [27]. For TMN protocol, we found two new multi-session attacks [24]. In fact, our new attacks in TMN protocol are quite surprisingly since TMN have been analyzed quite extensively [9, 11, 13, 18, 27-28]. Our preliminary results were reported in [21-24], but this paper extends our previous works by not only generalizing them into the two new techniques, which are the *on-the-fly* trace generation and the *textual* trace analysis, but also analyzing the comparative performance between the two trace generation methods.

In section 2, we provide the background on Micali's ECS1 and TMN protocol. In section 3, we compare our new method with existing related works. In section 4, we present our new CPN methodology and apply it to the two case studies.

2 Background

We use the following notations throughout the paper. $S \rightarrow R : M$ means that user S sends message M to user R. $SIG_X(M)$ represents party X's signature on a message M and we assume that M is always retrievable from $SIG_X(M)$. The encryption of a message M with party X's public key is denoted by $ENC_X(M)$. Also, H(C) stands for the hash of message C, and $E_K(M)$ means symmetric encryption on message M by key K. Note that a single session means the single execution of the protocol whereas multi-sessions mean the multiple and concurrent executions of the protocol.

2.1 TMN authenticated key exchange protocol (TMN) [18]

TMN is a cryptographic key exchange protocol for mobile communication system. TMN allows user A to exchange a session key with user B by the help of server J. The user A is called an initiator, but the user B is called a responder. The detail of TMN is described as follows.

1. $A \rightarrow J : (B, ENC_J(K_{aj}))$, A
2. $J \rightarrow B : A$
3. $B \rightarrow J : (A, ENC_J(K_{ab}))$, B
4. $J \rightarrow A : B, E_{Kaj}(K_{ab})$

Where K_{ab} is an exchanged session key and K_{aj} is A's secret which is used to transport the session key at the last step. Note that the session key is created by user B. In [18], it is suggested that the one-time pad and RSA algorithm are used as the underlying symmetric encryption and the public key encryption, respectively.

2.2 Micali's contract signing protocol (ECS1) [17]

Micali proposed an efficient optimistic fair exchange protocol for contract signing. The protocol aims to ensure that two exchanging parties get each other commitment on an agreed contract or neither of them does. There are three kinds of parties in the protocol : Alice as an initiator of the protocol, Bob as an responder of the protocol and a third trusted party who resolves a dispute between Alice and Bob during the exchange.

We denote Alice, Bob and a trusted party by A, B and TTP, respectively. It is assumed that both Alice and Bob have already agreed on a plaintext contract C before the exchange. Alice is committed to contract C as an initiator if Bob has both $SIG_A(C,Z)$ and M where $Z=ENC_{TTP}(A,B,M)$ and M is random. On the other hand, Bob is committed to C as a responder if Alice has both $SIG_B(C,Z)$ and $SIG_B(Z)$. However, there is no need for Alice to verify Z to prove Bob's commitment.

The following is the detail of a slightly modified version [25] of the original protocol to strengthen the dispute resolution request at step (4).

A1: 1) $A \rightarrow B$: $SIG_A(C,Z)$

B1: 2) $B \rightarrow A$: $SIG_B(C,Z)$, $SIG_B(Z)$

A2: If Bob's signatures in step 2 are both valid, then

3) $A \rightarrow B$: M

B2: If Bob receives valid M such that $Z=ENC_{TTP}(A,B,M)$

then the exchange is completed

else Bob requests TTP to resolve a dispute by the following step

4) $B \rightarrow TTP$: $SIG_A(C,Z)$, $SIG_B(C,Z)$, $SIG_B(Z)$

To resolve the dispute, TTP performs the following.

TTP1: If both Alice's and Bob's signatures in step 4 are valid and $Z=ENC_{TTP}(A,B,M)$ then

5a) $TTP \rightarrow A$: $SIG_B(C,Z)$, $SIG_B(Z)$

5b) $TTP \rightarrow B$: M

3 Related works

3.1 Model checking for cryptographic protocols

Many model checking methods [6-16] have been developed and applied to analyze cryptographic protocols. All of them except for NRL [15] and Proverif [16] can analyze only one attack trace of a found attack. In fact, all of them are based on the *off-the-fly* trace generation which means that an attack trace is computed after a state space is generated either partially or fully. The *off-the-fly* trace generation for all attack traces involves the searching for all paths between two states which is extremely time-consuming. Indeed, the searching for all paths can be seen as a core part of algorithms for solving the traveling salesman problem which is known to be NP-complete.

Avispa [6,7] is a research project which develops four state-of-the-art model checking methods to verify cryptographic protocols. They provide high performance analysis of protocols and a large number of protocols have been analyzed. Surprisingly, they found some new attacks in some protocols. In [8], Spin which is a widely used model checker tool is employed to analyze a cryptographic protocol. A known attack to a protocol can be detected. FDR which is a model checker for CSP has been applied to analyze many cryptographic protocols in [9,10]. It can detect many new attacks successfully in many protocols. In [11,12], Murphi which is a general model checker has also been applied to cryptographic protocols, and it discovered new attacks in some protocols. In [13-14], Petri nets-based model checking methods are employed to analyze cryptographic protocols, and they can detect known attacks only. All of the model checking methods discussed so far can analyze only one attack trace of a found attack.

There are two model checking methods which analyze multiple attack traces of a found attack, namely NRL [15] and Proverif [16]. While NRL computes all attack traces of a found attack, Proverif explores on a restricted set of attack traces which often contains only one trace. In fact, NRL is quite inefficient partly due to the computation of all paths in the *off-the-fly* approach.

3.2 Analysis of TMN and ECS1

In [25], Bao et al. analyzed ECS1 manually and found three message replay attacks in ECS1, and one attack in a simple modification of ECS1. They also proposed an improved ECS1 which can prevent all found attacks. In [26], Zhang and Liu applied a manual model checking technique to analyze ECS1. They found one new single-session attack in Micali's ECS1 and two new multi-session attacks in Bao et. al.'s modified version of ECS1. In fact, their attacks are also independently discovered by our method, but we found more attacks. In particular, we found one new single-session attack of Micali's ECS1, two new multi-session attacks in Micali's ECS1 and three new attacks of Bao's modified version of ECS1 all of which cannot be detected by Zhang and Liu's method. Since Bao et. al.'s and Zhang and Liu's methods are done by hands, their analysis does not cover attacks thoroughly.

TMN has been analyzed quite comprehensively. In [18], Simmon analyzed TMN manually and found a multi-session attack by using the homomorphic property of the underlying public key cryptographic algorithm. In [27], three formal method approaches, namely NRL, Interrogator and Inatest, for cryptographic protocols have been applied to TMN. Both NRL and Interrogator detect a single-session attack. However, Inatest can only reproduce Simmon's attack. In [11], Murφ can reproduce Simmon's attack, and detect a new multiple session attack. In [9], CSP/FDR is used by Lowe and Roscoe to discover one new single session attack and one new multi-session attack. In [13], Al-Azzoni et. al. applied CPN to detect a variant form of the attack found by Murφ [11]. In [26], by using a manual model checking, Zhang and Liu found some variant forms of Lowe and Roscoe's attacks [9] in both a single session and multiple sessions. Even though there have been many analyses on TMN, we found two new attacks on it.

4 Our Model

4.1 Our new methodology

In general, our model checking methodology for the analysis of cryptographic protocols consists of five steps which are (1) protocol and attacker representation, (2) state space and trace generation, (3) characterization and search for attack states, (4) attack trace extraction and (5) attack trace classification. However, our new method for computing all attack traces of a found attack contains two novel techniques which are *on-the-fly* trace generation and *textual* trace analysis. While the *on-the-fly* trace generation is employed in steps 2 and 4, the *textual* trace analysis is used in step 5. We focus our discussion on the two techniques but also explain some relevant steps if necessary. We discuss the *on-the-fly* trace generation first.

Assuming that a protocol and an attacker model are represented. The representation depends on a model checker approach. Then, a state space is generated from the representation. During the state space generation, when a state is generated, an attack trace to the state is computed at the same time and the computed trace is stored at the state. This computation is the core of the *on-the-fly* trace generation. It is important to notice that an attack trace of a state is stored at the state itself. Conceptually, an attack trace for a state is constructed by simply extending an attack trace stored in the previous state. Thus, there is no need to always compute an attack trace from the initial state, and such computation is vey expensive.

For simplicity, we assume that each state stores only one attack trace. Thus, our state space in general may contain more number of states than the state space in the *off-the-fly* trace generation. A state which can be reached by two different attack traces in the *off-the-fly* method becomes two different states in our method. To reduce the size of a computed state space in our method, we employ a decomposition technique. In particular, we define a configuration to compute a decomposed state space. In this paper, we consider the analysis of multi-sessions of protocol execution. A configuration consists of the information for the protocol execution in a multi-session setting, for example, the identities of initiator and responder, the role of attackers, secrets and nounces in each concurrent session, and a schedule of the execution of the multiple concurrent sessions. The schedule specifies that the decomposed state space is computed for one alternating execution of multiple concurrent sessions of protocol runs only, instead of all possible alternating executions. Exploring all possible alternating executions within a state space is expensive and causes a huge state space. However, we can explore each attack scenario, eg. a specific alternating execution or a specific initiator and responder, one by one by computing a decomposed state space with a specific configuration.

After the state space is obtained, attack states for each kind of attacks are searched in the state space. An attack is characterized by a vulnerability event which is an event potentially leading to a compromise of protocols. Vulnerability events are protocol-dependent. There can be many attack states which belong to the same vulnerability event and thus the same attack. When an attack state is found in the state space, an attack trace is extracted from the state immediately. By searching for all

attack states of the same attack, all attack traces of the attack can be obtained without any path searching. In other words, the computation for all attack traces is reduced to the searching for attack states which can be done efficiently. This *on-the-fly* trace generation technique provides a big improvement in the computation time of all attack traces when the number of attack traces and the number of states are large.

The number of attack traces obtained can be quite large. For example, we found 1,020 traces for an attack in the TMN protocol. So, we propose *textual* trace analysis technique to classify such large number of attack traces. Those attack traces are classified by using attack patterns. Attack patterns are minimal but necessary protocol traces for an attack. The development of an attack pattern is manual because an attack pattern is protocol-dependent. In an attack pattern, some parts of the protocol messages are fixed due to the protocol specification, but others can be varied in some ways.

While the development of an attack pattern is manual, the attack classification is automatic. Attack traces that contain the same attack pattern are classified into the same group of attack traces. As a result, a large amount of attack traces is reduced to a reasonable amount of attack patterns which are easier to analyze. Moreover, the attack classification process is iterative in that when a new attack pattern is found, it is used together with the existing patterns to filter the remaining attack traces. By using our two new techniques, protocol designers obtain a deep and thorough analysis of all possible attacks to cryptographic protocols.

Our two new techniques are independent to model checking tools. We implement them in a model checker tool called CPNTools [19,20]. Originally, CPNTools provides the *off-the-fly* trace generation only and the search mechanism for only one attack trace. We employ the simplest approach to implement the *on-the-fly* trace generation in CPNTools by using a protocol representation which records incrementally a protocol trace by users and attackers into each state. The *textual* trace analysis for attack classification is realized in CPNTools by writing an ML-like program to extract attack traces from attack states and process them.

4.2 Our analysis for TMN

Our method for TMN. In this section, we discuss the assumptions of our protocol analysis. We also describe vulnerability events of TMN, and provide a definition of a configuration of the protocol execution. Finally, we discuss attack patterns.

Definition 1 : The assumptions of the protocol execution
The following are the assumptions of the execution of the TMN protocol.

1. There are three users who are an initiator, a responder and a server. And all the users follow the protocol specification strictly and honestly.
2. There is one attacker whose abilities are defined below.
3. The underlying encryption is perfect in that nothing can be inferred from a ciphertext without the knowledge of the correct key. This is known as Dolev and Yao's assumption [29]. Also, we consider a general public key encryption scheme, instead of RSA algorithm.

4. We consider the execution of two concurrent sessions of the protocol where such execution can be performed in an alternating and non-sequential style.
5. Initiator and responder involve in one session only, but the server may involve in more than one session.
6. In a session, there must be at least one authentic user.

In assumption 5), the sessions that initiator and responder involve may not be the same. In 6), it means that there is at least one victim user in a session.

Definition 2 : The attacker abilities
The attacker in our model is capable of the following:
1. The attacker can eavesdrop, modify and drop messages during the transmission between users.
2. The attacker can send any message to a user.
3. The attacker can either initiate a new session with users or take part in an existing session with users.
4. The attacker can impersonate any user.
5. The attacker can perform any cryptographic computation, eg. encryption and decryption, by using known keys, known messages and known ciphertexts with a reasonable power.
6. The attacker does not attack himself.
7. There is at most one attacker who performs the attack ability in 1) on a protocol step in a session.

The assumptions 1) and 4) mean that the attacker can act as an external observer or an impersonator, respectively. In 7), any message that is sent from an attacker will not be modified further by any other attacker.

Attack states are characterized by vulnerability events. For the TMN protocol, there are two basic vulnerability events which are secret disclosure by an attacker and session key commitment by initiator and responder. Based on the two basic events, the following combined and interesting vulnerability events can be created.

Definition 3 : The combined and interesting vulnerability events.
There are three combined vulnerability events.
1. The attacker learns K_{ab} and K_{aj}, and both A and B commit on K_{ab}. $[K_{ab},K_{aj}][K_{ab}][K_{ab}]$
2. The attacker learns K_{ab} and K_{aj}, and A is fooled to commit on K_i but B commits on K_{ab}. $[K_{ab},K_{aj}][K_i][K_{ab}]$
3. The attacker learns K_{ab} and K_{aj}, and A is fooled to commit on K_{aj} but B commits on K_{ab}. $[K_{ab},K_{aj}][K_{aj}][K_{ab}]$

We use the notation $[KB_1][KB_2][KB_3]$ to describe each combined vulnerability event where KB_1 stands for keys that are known by the attacker, and KB_2 and KB_3 stands for keys that are committed by users A and B, respectively, at the completion of the protocol.

In the combined event 1, the attacker learns all later communication between A and B, because the attacker obtains the session key between A and B. Lowe and Roscoe's

multi-session attack [9] belongs to this event. The combined events 2 and 3 are our new attacks. The result of the events 2 and 3 can be seen as a kind of the *man-in-the-middle* attacks where the attacker situates between A and B. In event 2, the attacker can impersonate B to A by using key K_i, while the attacker can impersonate A to B by using key K_{ab}. Then, the attacker learns the later communication between A and B. The event 3 is similar to the event 2, but B impersonation to A is done by using key K_{aj}.

In the following, we provide the definition of a configuration for computing a decomposed state space for TMN protocol.

Definition 4 : A configuration of the state space computation for TMN
A configuration of a decomposed state space computation consists of $((S_1, S_2,\ldots,S_n), Sch, Tr)$ and $S_i = (s,I,R,T,K,N)$ for $1 \leq i \leq n$ where n is the number of sessions, and

1. S_i is a session information for the i-th session which consists of
   1.1. s is a session identity
   1.2. I, R and T are identities for an initiator, a responder and a server, respectively.
   1.3. K is keys for each party (including attacker) which consists of a pair of public and private keys, and a shared key with a specific party
   1.4. N is nounces used by each party
2. *Sch* is a multi-session schedule which contains a specific alternating execution of multiple concurrent sessions of protocol runs
3. *Tr* is a list of attack traces and their vulnerability events

In the configuration, S_i and *Sch* are input parameters for the state space computation while *Tr* is the output from the state space computation. In this paper, we consider the multi-session schedule for the *man-in-the-middle* attack [30] where the attacker participates in two sessions and replays messages between them synchronously.

We consider the following four configurations of two concurrent sessions which are all possible configurations regarding to our assumptions. Note that in the configurations, *K*, *N*, *Sch* and *Tr* are omitted for simplicity.

1. *(1,A,B,J) & (2,In,In,J)*
2. *(1,A,In,J) & (2,In,B,J)*
3. *(1,In,B,J) & (2,A,In,J)*
4. *(1,In,In,J) & (2,A,B,J)*

There are two roles of our attacker which are an external observer and an impersonator. In configuration *(1,A,B,J)*, the attacker behaves explicitly as an external observer on the communication amongst A, B and J, In *(1,A,In,J)* and *(1,In,B,J)*, the attacker explicitly impersonates B and A, respectively. But in any configuration, the attacker can impersonate implicitly any users according to our attacker model.

In TMN, an attack pattern for a session consists of the four protocol steps where ciphertexts in steps 1, 3 and 4 are of appropriate types of encryption and they are obtained from any plaintexts. But identities of initiator in step 2 and responder in step 4 are fixed to A and B, respectively. Also, identities of initiator and responder between steps 1 and 3 must be consistent, but can be anything. In fact, the important parts of the attack pattern for TMN are ciphertexts in steps 1, 3 and 4 since they contains session and encryption keys that attackers want to disclose and to forge to

compromise the system, respectively. Thus, we have an attack pattern for each possible plaintext of the ciphertexts in the three steps. The following shows one attack pattern of our new attack which corresponds to the combined event 2.

1) $A \rightarrow In(J) : (B, \{K_{aj}\}PK\text{-}J), A$
   $In(J) \rightarrow J : (X2, \{K_i\}PK\text{-}J), X1$
1') $In(A) \rightarrow J : (X4, \{K_i\}PK\text{-}J), X3$
2') $J \rightarrow In(B) : X3$
2) $J \rightarrow In(B) : X1$
   $In(B) \rightarrow B : A$
3) $B \rightarrow J : (X1, \{K_{ab}\}PK\text{-}J), X2$
3') $In(B) \rightarrow J : (X3, \{K_{aj}\}PK\text{-}J), X4$
4') $J \rightarrow In(A) : X4, E_{Ki}(K_{aj})$
4) $J \rightarrow In(A) : X2, E_{Ki}(K_{ab})$
   $In(A) \rightarrow A : B, E_{Kaj}(K_i)$

where K_i is attacker's secret keys.

While 1) – 4) describe protocol steps in the 1st session, 1') – 4') indicate protocol steps in the 2nd session. *X1, X2, X3* and *X4* stand for arbitrary identities that the attacker creates. In step 1), the message that A sends to J is modified by the attacker. The original message is indicated by $A \rightarrow In(J)$, but the modified message by the attacker is indicated by $In(J) \rightarrow J$. Also, the messages at steps 2) and 4) are modified by the attacker.

We found 10 attack patterns for each of the events 2 and 3 which are our new attacks. The details of the attack patterns can be found in [24].

Performance. In the following, we compare the results between our *on-the-fly* and the *off-the-fly* trace generation methods both of which are implemented in CPNTools model checker. The experiment is done by using a PC with Intel Core2 Duo 2.33 Ghz and 2 GB of RAM.

In table 1, we show the comparison of the sizes of the state spaces between the two methods for the four configurations. In the configurations, session and server identities are ignored. In the worst case the number of states and arcs in the *on-the-fly* method are increased for 40.5 % and 37.9 %, respectively. However, in the best case the number of states and arcs are increased for only 9.8 % and 6.2%, respectively.

In tables 2 and 3, we compare the computation times for state spaces (*st*) and traces (*tr*) in the two methods for two configurations. Tables 2 and 3 are for the cases of the large number and the small number of states, respectively. The event 4 represented by $[K_{aj}][K_{aj}][K_{ab}]$ means that the attacker learns A's secret and fools A to commit to A's secret as a session key. The events R1 and R2 are the remaining vulnerability events where the attacker learns K_{ab} and K_{aj}, respectively. Note that in the events 2 and 3 in table 3, information is omitted since no attack trace is found for the events.

It is clear that our *on-the-fly* method improves the total computation times tremendously. When the numbers of states and traces are large, for example in the event R2 of table 2, it takes about 16 minutes (1,002 sec.) in our method, but about 19 hours (68,913 sec.) in the *off-the-fly* method. When the numbers of states and traces are small, for example in the event 4 of table 3, it takes about 2 minutes in our method, but about 11 minutes (5,376 sec.) in the *off-the-fly* method.

Table 1. The comparison of the sizes of state spaces

Configurations	On-the-fly		Off-the-fly		Increment (%)	
	nodes	arcs	nodes	arcs	nodes	arcs
1. (A,B)(In,In)	104,346	109,476	74,244	79,344	40.5	37.9
2. (A,In)(In,B)	73,806	77,568	55,656	59,730	32.6	29.8
3. (In,B)(A,In)	51,212	52,639	46,637	49,543	9.8	6.2
4. (In,In)(A,B)	34,160	35,095	30,974	33,061	10.2	6.15

Table 2. The comparison of the computation times for configuration *(1,A,B,J) & (2,In,In,J)*

Events	Attack Traces	On-the-fly Time (sec)			Off-the-fly Time (sec)			Improvement %
		st	tr	total	st	tr	total	
1. Event 2	360	976	0	976	369	1839	2208	126
2. Event 3	360	976	0	976	369	1774	2143	119.56
3. Event 4	1,020	976	0	976	369	5239	5608	474
4. Event R1	8,226	976	10	986	369	40028	40397	4,039
5. Event R2	13,056	976	26	1002	369	68544	68913	6,777

Table 3. The comparison of the computation times for configuration *(1,In,In,J) & (2,A,B,J)*

Events	Attack Traces	On-the-fly Time (sec)			Off-the-fly Time(sec)			Improvement %
		st	tr	Total	st	tr	total	
1. Event 2	0	-	-	-	-	-	-	-
2. Event 3	0	-	-	-	-	-	-	-
3. Event 4	360	120	0	120	80	568	688	473.33
4. Event R1	684	120	0	120	80	1556	1,636	1,263.33
5. Event R2	2,388	120	1	121	80	5296	5,376	4,380

Indeed, our *on-the-fly* method requires more times for state space computation, but the *off-the-fly* method requires more times for trace generation. However, the time for trace generation in the *off-the-fly* exceeds greatly the time for state space computation in the *on-the-fly* method. It should be noticed that in both tables, when the number of traces is increased, the time for trace generation in the *off-the-fly* method grows greatly, but the time for trace generation in our method is almost constant.

4.3 Our analysis for ECS1

Our method for ECS1. Our method for the analysis of ECS1 is similar to that for TMN discussed previously. So, we discuss only the main differences between them here.

The assumptions of the protocol execution for ECS1 are similar to those assumptions in definition 1 except for assumption 5. For ECS1, the same initiator and responder may participate in more than one session. We assume two kinds of

attackers: *I* and *Ar*. The attacker *I* is exactly the same as the attacker *In* discussed in definition 2. However, *Ar* is different and is a malicious user who participates in a session and conspires with attacker *I* by sharing some information. More specifically, *Ar* can be either an initiator or a responder, but not an external observer. Note that one attack found by Bao et. al. [24] involves these two kinds of attackers.

There is one vulnerability event in ECS1 protocol which is an unfair exchange state. An unfair state means that one party, who is either initiator or responder, gets another party commitment, but the latter does not get the former commitment. There are two unfair states.

- The initiator has the responder's commitment, but the responder does not have the initiator's commitment.
- The responder has the initiator's commitment, but the initiator does not have the responder's commitment.

We found one new single-session attack and two new multi-session attacks of Micali's ECS1, and three new attacks of Bao's modified version of ECS1. The details can be found in [22].

Performance. In the following, we compare the results between the two methods implemented in CPNTools. The experiment is done by using a notebook computer with Intel Core2 Duo 2 Ghz and 3 GB of RAM.

Table 4. The comparison of the sizes of state spaces

Configuration	On-the-fly		Off-the-fly		Increment %	
	nodes	Arcs	nodes	arcs	node	arc
1.(I,Ar,c1,mi1)(I,B,c1,mi2)	235564	235563	235564	235563	0	0
2.(I,B,c1,mi1)(I,Ar,c1,mi1)	118774	119049	118774	119049	0	0
3.(I,B,c1,mi1)(Ar,I,c1,mi2)	92498	92497	92498	92497	0	0
4.(Ar,I,c1,mi1)(I,B,c1,mi2)	86470	86469	85582	85705	1.03	0.89
5.(I,B,c1,mi1)(I,B,c2,mi2)	70082	70081	68509	68629	2.29	2.11
6.(I,B,c1,mi1) (Ar,I,c1,mi1)	68694	68693	68110	68173	0.85	0.76
7.(I,B,c1,mi1) (Ar,I,c2,mi1)	68694	68693	68110	68173	0.85	0.76
8.(I,B,c1,mi1)(I,B,c1,mi1)	48728	49355	38828	39488	25.5	24.9
9.(A,B,c1,ma1)(I,Ar,c2,mi1)	34930	34929	34930	34929	0	0

In table 4, we show the comparison of the sizes of the state spaces between the *on-the-fly* and the *off-the-fly* trace generations for some configurations. Each configuration consists of the information of two concurrent sessions, and each session is represented by (i_1,i_2,c,m) where i_1 and i_2 are identities for initiator and responder, *c* is a contract and *m* is the random. The table shows that in most cases the number of states and arcs in the two methods are identical. However, in the worst case the number of states and arcs are increased for only 25.5 % and 24.9%, respectively.

In table 5, we compare the computation times for state spaces (*St*) and traces (*Tr*) in the two methods for the same configurations as table 4. The attack traces in the table are for the two unfair states in the vulnerability event.

It is clear that our *on-the-fly* method improves the total computation times greatly when the number of states and traces are large. In particular, for the best case the

improvement is 116.75%. However, when the number of states and traces are small in some cases, for example in the configuration 9 which contains 34,930 nodes, the *on-the-fly* method performs better. Note that when the number of attack traces is increased, the time for trace generation in the *off-the-fly* method grows greatly, but the time in our method grows very slowly.

Table 5. The comparison of the computation times

Configuration	attack traces	On-the-fly Time (sec)			Off-the-fly Time(sec)			Improve-ment %
		St	Tr	Total	St	Tr	Total	
1.(I,Ar)(I,B)	7,032	9863	307	10,170	7090	14954	22,044	116.75
2.(I,B)(I,Ar)	2,664	2871	139	3,010	1865	3096	4,961	64.91
3.(I,B)(Ar,I)	3648	1721	114	1,835	994	1563	2,557	39.34
4.(Ar,I)(I,B)	4104	1548	125	1,673	863	1565	2,428	45.12
5.(I,B)(I,B)	1272	1091	133	1,224	694	1207	1,901	55.31
6.(I,B) (Ar,I)	1,876	1077	88	1,165	693	663	1,356	16.39
7.(I,B) (Ar,I)	1,876	1077	182	1,259	642	714	1,356	7.7
8.(I,B)(I,B)	1,116	610	42	652	251	414	665	1.99
9.(A,B)(I,Ar)	1,210	740	41	781	186	256	422	-76.69

Similar to the result in the analysis of TMN, our *on-the-fly* method requires more times for state space computation, but the *off-the-fly* method requires more times for trace generation. But here there is a case which is a configuration 9 where the time for trace generation in the *off-the-fly* does not exceed the time for state space computation in our method.

5 Discussion

According to the results obtained, we argue that our *on-the-fly* method is complementary to the *off-the-fly* method, and should be used to deal with the case for a large state space and a large number of attacks traces. Similarly, our *textual* trace analysis is also complementary to visualization technique in that when the number of traces is large, it is more suitable to employ the *textual* trace analysis. However, when the number of traces is very small, visualization technique can provide some intuitive illustration of the traces.

It is true that our *on-the-fly* trace generation method requires more amount of memory than the *off-the-fly* method. In particular, each state in our method is augmented with an attack trace. Moreover, a state which can be reached by two different attack traces in the *off-the-fly* method becomes two different states in our method. However, the *off-the-fly* method also requires a large amount of memory to store and process attack traces during the path searching for all attack traces. But our method avoids the complex path-searching computation and speeds up the whole computation time.

We hope that our very simple method would be useful for other applications of model checking for the analysis of *all errors* in any system. As a future work, we aim

to optimize our method for the memory requirement, and to apply our method to analyze for other cryptographic protocols.

6 Conclusion

In this paper, we propose a very simple but practical model checking methodology for the analysis of cryptographic protocols. Our methodology offers an efficient analysis of all attack traces for each found attack, and is independent to model checking tools. It contains two novel techniques which are *on-the-fly* trace generation and *textual* trace analysis. We apply our method to two case studies. The result shows that when the numbers of states and traces are large, our method is more efficient. In some case, our method improves the computation time over the *off-the-fly* method for 6,777%. In addition, we found many new attacks in the two case studies.

Acknowledgments. The work reported here is supported financially by National Research Council of Thailand. A previous version of CPNTools model for ECS1 was developed by Panupong Sornkom. I would like to thank Kurt Jensen and his group to suggest the names of the on-the-fly trace generation and the textual trace analysis after my presentation at CPN'09 workshop. Also, I would like to thank anonymous reviewers for their comments.

References

1. Clark, J., Jacob, J., A survey on Authentication Protocols : Research report, University of York, http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz (1997)
2. Meadows, C., Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends: IEEE Journal on Selected Areas in Communications, 21(1), 44--54 (2003)
3. Meyer, U., Wetzel, S., A man-in-the-middle attack on UMTS: In the 3rd ACM workshop on Wireless security, pp. 90-97 (2004)
4. Cervesato, I., Jaggard, A.D., Scedrov, A., Tsay, J., Walstad, C, Breaking and fixing public-key Kerberos: Information and Computation, 206(2-4), 402-424 (2008)
5. Syverson, P.F., A Taxonomy of Replay Attacks: In the 7th IEEE Computer Security Foundations Workshop, pp. 187-191 (1994)
6. The AVISPA project, http://avispa-project.org
7. Viganò, L., Automated Security Protocol Analysis With the AVISPA Tool. Electr. Notes Theor. Comput. Sci. 155: 61-86 (2006)
8. Maggi, P., Sisto, R., Using SPIN to Verify Security Properties of Cryptographic Protocols: In 9th International SPIN Workshop, LNCS, vol. 2318, pp.85--87, Springer (2002)
9. Lowe, G., Roscoe, B., Using CSP to Detect Errors in the TMN Protocol: IEEE Transactions on Software Engineering, 23(10), 659--669, (1997)
10. Lowe, G., Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR: Software - Concepts and Tools 17(3): 93-102 (1996)
11. Mitchell, J., Mitchell, M., Stern, U., Automated analysis of cryptographic protocols using Murϕ: In 1997 IEEE Symposium on Security and privacy, pp.141--151, (1997)
12. Shmatikov, V., Mitchell, J.C., Finite-state analysis of two contract signing protocols: Theor. Comput. Sci. 283(2): 419-450 (2002)

13. Al-Azzoni, I., Down, D.G., and Khedri, R., Modeling and Verification of Cryptographic Protocols Using Coloured Petri Nets and Design/CPN: Nordic Journal of Computing, 12(3), 201--228 (2005)
14. Bouroulet, R., Devillers, R., Klaudel, H., Pelz, E., Pommereau, F., Modeling and Analysis of Security Protocols Using Role Based Specifications and Petri Nets: In PETRI NETS 2008, LNCS, vol. 5062, pp. 72—91, Springer (2008)
15. Meadows, C., The NRL protocol analyzer: An overview: Journal of Logic Programming, 26(2), pp. 113- 131 (1996)
16. Blanchet, B., An Efficient Cryptographic Protocol Verifier Based on Prolog Rules: In 14th IEEE Computer Security Foundations Workshop, pp. 82-96, IEEE press (2001)
17. Micali, S., Simple and Fast Optimistic Protocols for Fair Electronics Exchange: In 21st Symposium on Principles of Distributed Computing, ACM press, pp. 12-19 (2003)
18. Tatebayashi, M., Matsuzaki, N., Newman, D., Key Distribution Protocol for Digital Mobile Communication Systems, In CRYPTO-89, LNCS, vol. 435, pp. 324-334, Springer (1990)
19. Jensen, K., Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use: Vol.1. Monographs in Theoretical Computer Science, Springer-Verlag (1997)
20. CPNTools, http://wiki.daimi.au.dk/cpntools/
21. Sornkhom, P., Permpoontanalarp, Y., Security Analysis of Micali's Fair Contract Signing Protocol by Using Coloured Petri Nets: In the 9th ACIS-SNPD 2008, pp. 329-334, IEEE press, Thailand (2008)
22. Sornkhom, P., Permpoontanalarp, Y., Security Analysis of Micali's Fair Contract Signing Protocol by Using Coloured Petri Nets : Multi-session case: In the 5th International Workshop on Security in Systems and Networks, pp. 1-8, IEEE press, Italy (2009)
23. Permpoontanalarp, Y., Sornkhom, P., A New Coloured Petri Net Methodology for the Security Analysis of Cryptographic Protocols: In the 10th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Denmark, pp. 81-100 (2009)
24. Permpoontanalarp, Y., Security Analysis of the TMN protocol by using Coloured Petri Nets : Multi-Session Case: In the 10th International Conference on Intelligent Technologies, pp.401-410, China (2009)
25. Bao, F., Wang, G., Zhou, J., Zhu, Z., Analysis and Improvement of Micali's Fair Contract Signing Protocol: In 9th Australasian Conference on Information Security and Privacy, LNCS, vol. 3108, pp. 176-187, Springer Verlag (2004)
26. Zhang, Y., Wang, Z., Yang, B., The Running-Mode Analysis of Two-Party Optimistic Fair Exchange Protocols: International Conference on Computational Intelligence and Security, LNCS, vol. 3802, pp. 137-142, Springer Verlag, China (2005)
27. Kemmerer, R., Meadows, C., Millen, J., Three systems for cryptographic protocol analysis: Journal of Cryptology, 7(2) (1994)
28. Zhang, Y., Liu, X., An approach to the formal analysis of TMN protocol: Progress on Cryptography: 25 years of Cryptography in China, LNCS, vol.769, pp.235--243, Springer Verlag (2004)
29. Dolev, D., Yao, A., On the security of public key protocols: IEEE Transactions on Information Theory, 29(2): 198--207 (1983)
30. Lowe, G., An Attack on the Needham-Schroeder Public-Key Authentication Protocol: Information Processing Letters, 56(3), 131-133 (1995)
31. ITU-T, Recommendation Z.120: Message Sequence Chart (MSC), ITU-T, Geneva, 1996.