Design of an Efficient Security Protocol Analyzer

Shinsaku KIYOMOTO†, Haruki OTA†, and Toshiaki TANAKA†

†KDDI R & D Laboratories Inc. Information Security Laboratory
2-1-15 Ohara Fujimino-shi Saitama, 356-8502, JAPAN

Summary
In this paper, we present a efficient security protocol analyzer to verify cryptographic security protocols. Our analyzer verifies security protocols based on notions of provable security. The analyzer only checks whether the core properties of security protocols satisfy the notions, making it faster than previous tools. The analyzer automatically checks whether authentication and key distribution protocols satisfy definitions such as Secure Mutual Authentication, Semantic Security, and Forward Secrecy. A user can design and evaluate security protocols by using our analyzer, according to the condition that the protocol will be used. Furthermore, our analyzer has a sophisticated GUI to set security protocols to be evaluated. Thus, the analyzer is useful for constructing and checking security protocols for many services.

Key words:
Security Protocol, Automatic Verification, Provable Security

1. Introduction

Entity authentication protocols and key distribution protocols are core technologies for secure services in distributed systems. Unfortunately, cryptographic protocols are currently designed through a process of trial and error, and security flaws in a protocol are difficult to locate using only a designer's review. Thus, the history of security research has many examples of security protocols with security flaws. For example, Denning and Sacco found a security flaw in the Needham-Schroeder key distribution protocol, which allows an intruder to re-use a previous session key as a new session key [1].

Needham *et al*. have emphasized the need for automatic verification methods for cryptographic protocols [2][3], accordingly, various security protocol analyzers have been proposed. Many protocol analyzers are based on formal methods, where the analyzer compares possible states of a security protocol with pre-defined states such as vulnerable states and states based on intruder models. Such analyzers usually take a long time to evaluate a security protocol. Furthermore, the computational cost and required memory space may grow in proportion to the number of protocol flows and the states. Therefore, the optimization of analyzer logic and protocol descriptions has been actively discussed by researchers in the area of logic-based automatic verification.

Manuscript received June 5, 2007
Manuscript revised June 20, 2007

Bellare *et al*. have proposed another method for proving protocol security. Their proof is based on provable security in cryptology; thus, if a protocol can be proven secure in a given model, then the protocol is guaranteed to have none of the security flaws addressed by that model. By using the essence of Bellare's proposal, we can describe efficient rules for automatic verification will be described. The essence of their work is also useful for constructing security protocol analyzers; the difficulty lies in translating their concepts into rules that can be interpreted by a computer.

In this paper, we present a efficient security protocol analyzer that enables to verify cryptographic security protocols and describe implementation results of the analyzer.

1.1 Related Work

Protocol verification methods can be categorized into those based on formal verification (see[4] for a survey) and those based on provable security.

1.1.1 Formal Verification

As an example of the former, in the Dolev and Yao model [5] an intruder is a person with complete control over the communication channel: the intruder can eavesdrop on all traffic and can block or send messages to any entity. However, they did not consider the case where an intruder had knowledge of system secrets. Millen proposed a protocol verification system [6] based on the Dolev-Yao model, which uses an exhaustive search of the communication state space. This tool has never found an undefined attack on a cryptographic protocol. A similar tool was proposed by Longley and Rigby [7].

The NRL Protocol Analyzer [8] is also based on the Dolev-Yao model, and uses a similar approach to Millen and Longley-Rigby tools. This analyzer verifies the ability to reach insecure states in a security protocol, and has been used to find security flaws [9][10]. However, the analyzer cannot fully prove security, although it can provide proof

of insecure states being reachable or not. Murφ is an automatic verification tools proposed by Mitchell *et al*. [11] . Murφ searches for insecure points within the state space using a model checker.

A different approach, based on formal verification, is to use model logics, similar to the analysis of the evolution of knowledge and belief in distributed systems. The best-known example was developed by Burrows, Abadi, and Needham, and is commonly known as BAN logic [12]. In the analysis, an initial set of beliefs is adopted, and another set of beliefs is adopted when a message is received in a protocol. If the resulting set of beliefs is acceptable, then the protocol is declared to have been proven correct. The logic cannot be used to prove secrecy, only authenticity, because the logic does not attempt to model knowledge [13]. GNY [14] logic and SVO [15] logic are related to BAN logic and were constructed with automatic verification in mind.

Kindred and Wing were the first to propose an automatic verification tool based on BAN logic [16]. Mathuria *et al*. proposed GNY-based automatic verification [17]. Paulson presented a verification tool [18] based on SVO logic. An automatic verification tool named C3PO also uses SVO logic [19]

Improving the efficiency of logic-based automatic verification is still being studied; methods such as Strand Space [20], Athena [21], STA [22], TRUST [23], OFMC [24], and ASPASyA [25] are a few currently being developed.

Yet another approach to applying formal methods is modeling a protocol as an algebraic system [26] [27].Toussaint proposed an algebraic model where the words describing a protocol are expressed by an isomorphism between a free algebra and a crypto-algebra. Meadows proposed an algebraic model to improve the NRL Protocol Analyzer [28].

Roscoe *et al*. proposed an approach based on the process algebra CSP [29][30]. A protocol translation tool was proposed to use a CSP description [31] and automatic verification tool [32] using a model checker FDR were proposed. CAPSL and MuCAPSL are automatic verification tools based on a intermediate language CIL [33].

Gordon and Abadi proposed the *S*-π calculus [34], in which an intruder is not explicitly modeled; instead, an intruder is represented as an arbitrary *S*-π calculus process.

LOTOS defined a protocol as a state diagram and investigated reachability analysis techniques [35].

1.2 Provable Security

Moving to the latter category, notions of entity authentication and key distribution with provable security were initially proposed by Bellare and Rogaway. They proposed a two party case [36] as well as a three-party case [37] wherein a trusted third party distributes a session key to each entity. Their method of proving the security of protocols was based on the method used for proving semantic security of encryption.

Their models (BR-models) were adapted to the public key setting by Blake-Wilson *et al*. [38][39][40]. Later, Bellare, Pointcheval, and Rogaway proposed the notion of security for password-based authentication and key distribution [41]. Shoup and Rubin [42] adapted a new security model to BR-models, which is used for smart-card-based protocols. Details of these models will be shown in later sections.

Bellare, Canetti, and Krawczyk proposed a different approach to provable security for key-exchange protocols [43]. Their approach is based on theory from the area of multi-party computation. In this model, an idealized version of the protocol is defined, and it is proven that any real world adversary cannot be as powerful as an ideal adversary. This approach was subsequently generalized and applied to many cryptographic settings; this generalization is known as Universal Composability. M. Backes *et al*. have discussed how to apply the concept of proofs in Universal Composability to the Dolev-Yao model [44][45][46][47].

Shoup presented a framework that follows the basic simulatability approach as in Bellare-Canetti-Krawczyk approach, but introduces significant modifications in order to be applicable to other cryptographic protocols [48].

Canetti and Krawczyk presented a formalized model [49] for analyzing key exchange protocols that combine the BR models and the Bellare-Canetti-Krawczyk model.

These approaches are still being studied. The BR models can separate security notions according to an adversary's ability (see 2.2), so we have chosen to adopt them.

1.3 Our Contribution

Despite all this research, no automatic verification tools based on notions of provable security have been proposed to date. Furthermore, the results of existing analyzers either do not directly correspond to security notions for security proofs, or no rules corresponding to security notions such as *Weak* Forward Secrecy exist on the analyzers.

This paper proposes a verification tool based on security notions in BR models. The main difficulty in designing the verification tool was how to extract common verification rules from proofs of security protocols, since proofs traditionally have been customized for each protocol. The key materials for constructing proofs are features of data such that data used for authentication is probabilistically unforgeable by an adversary. We identify the essential parts of the proofs and construct verification rules based on the security notions; for example, in authentication protocols, input data which changes an oracle to an accept

state is considered essential. Bellare's proof verifies whether the data has sufficient randomness and security, so our verification rules also check the same properties.

Some existing analyzers need to define methods of attacks for analysis; in constant, our analyzer models an adversary using queries that will be introduced later session, and needs no rules for specific attacks. Some existing analyzers check whether attacks are possible, but our analyzer checks whether a security protocol is secure under an adversarial model that includes any attacks.

Previous analyzers incurred a heavy computational cost and large rule sets to complete verification of security protocols. Our analyzer only verifies that core properties of security protocols satisfy the conditions of the security proofs. Thus, computational costs such as transaction time and required memory size will be reduced.

Furthermore, we designed our analyzer such that a user could operate it easily without any special skills. Our analyzer has a sophisticated GUI to input security protocols.

We believe our analyzer is useful for constructing and checking security protocols for many services.

1.4 Organization

The rest of the paper is organized as follows. In Section 2, we introduce models of provable security and security notions. Section 3 describes the basic design of our analyzer. Section 4 provides details of implementation, and Section 5 shows our evaluation results. Finally, we present some considerations and conclude this paper in Section 6.

2 Model of Provable Security for Security Protocols

2.1 Adversary Model

The communication model proposed by Bellare and Rogaway is independent of the specification of protocols, and the simulation of communication is constructed using oracles which are defined for each entity and each session. An adversary controls all communications and interaction between a set of oracles. An oracle of entity *U* in session *i* is defined as Π^i_U. The following queries are available to the adversary in the Bellare-Rogaway model:

- $Send(U, i, M)$

  This query allows the adversary to send message *M* to an oracle Π^i_U. The oracle Π^i_U returns the next message of the protocol and the result to the adversary. The result is one of three possible results, *Accept*, *Halt*, or *. *Accept* means that the authentication is successful and/or session key is successfully computed. *Halt* means that the authentication or computation of session key has failed and/or the message *M* is invalid, i.e. not defined as part of the protocol specification. The value * is supported to suggest, for message *M*, that the oracle outputs no result.
- $\mathrm{Re}veal(U, i)$

  This query allows the adversary to obtain a session key without the target party. The oracle $\prod^i_U$ returns the accepted session key to the adversary.
- $Corrupt(U, K)$

  The adversary can know *U*'s current long-lived key (LL-key) of *U*; this key is also known as a permanent key of *U*, and change it to *K*. The adversary also obtains the internal state of *U* by using this query. After using a *Corrupt* query, the adversary cannot use any *Send* query to $\prod^i_U$ and/or the partnering oracle.
- $Test(U, i)$

  When an oracle Π^i_U receives this query, the oracle flips a coin $b \in \{0,1\}$. If $b = 0$, then the oracle returns a session key; otherwise, the oracle returns a random string that has the same length and the same distribution as the session key.

2.2 Definition of Protocol Security

In this subsection, we introduce security notions regarding authentication and key distribution protocols. The definition of secure authentication proposed by Bellare and Rogaway is as follows:

Definition 1. Secure Mutual Authentication

If oracles $\prod^i_{U_1}$ and $\prod^j_{U_2}$ accept in a *Matching Conversation* and the probability that one oracle (or both) accepts without another oracle engaged in a *Matching Conversation* is negligible, then the protocol is a secure mutual authentication protocol.

A detailed definition of *Matching Conversation* may be found in Bellare's paper (see [36][38]).

Under the *Matching Conversation* condition, the adversary transfers communication between the oracles honestly, without any interception and/or alteration. Thus, the adversary's strategy is either to guess the correct authentication code or to divert an expired or fresh authentication code that is received from a non-partnering entity. The probability that an oracle accepts without a *Matching Conversation* is defined as $\Pr\left[Succ^{No-Matching}\right]$ .If $\Pr\left[Succ^{No-Matching}\right]$ is negligible, the security of the protocol can be proven under the assumption that a primitive function is secure.

Definition 2. Semantic Security of key distribution protocols

If an adversary exists and is unable to distinguish the session key from an independent random value with

non-negligible probability, then the protocol satisfies semantic security as a key distribution protocol.

Semantic security includes many security notions based on the adversary's ability, i.e. which queries an adversary can use. We define three security notions for the Semantic Security based on BR models. To separate models, the analyzer can verify each adversary's model and output whether a protocol is secure in that model. Consequently, we can verify that the protocol is secure in an environment assumed an adversary's ability. For example, if we assume that the LL-key is secure in a real environment, we need not to consider forward secrecy notions.

The notion of security against known-key attack (KKA) assumes an adversary can obtain a previous session key or replace a known session key. An adversary of forward secrecy can also obtain an LL-key in addition.

The notion of forward secrecy is separated into two notions, *weak* Forward Secrecy (w-FS), and *strong* Forward Secrecy (s-FS), based on whether an adversary can obtain the LL-key when a protocol has not yet finished. Definition 2.3.2 means that the adversary can steal an LL-key during the processing of the protocol. A protocol that has s-FS is more secure than a protocol having w-FS.

Definition 2.1. Basic Adversary Model

If an adversary that uses the target party as an uncorrupted party and other parties as uncorrupted oracles exists, and if the protocol satisfies semantic security against the adversary, then the protocol is secure against the basic adversary model. In this model, an adversary can use *Send* and *Test* queries, but cannot use *Reveal* and *Corrupt* queries.

Definition 2.2. Secure against Known-key Attacks

If there exists an adversary that knows any expired session keys of the target party and any session keys of other parties, and if the protocol satisfies semantic security against the adversary, then the protocol is secure against known-key attacks. In this model, an adversary can use *Send*, *Test*, and *Reveal* queries, but cannot use the *Corrupt* query.

Definition 2.3.1. weak Forward Secrecy

If there exists an adversary that knows any expired session keys of the target party and any session keys of other parties: if the adversary can know the LL-key after both entities are terminated; and if the protocol satisfies semantic security against the adversary, then the protocol has *weak* Forward Secrecy (w-FS). In this model, an adversary can use *Send*, *Test*, and *Reveal* queries. The adversary may also use a *Corrupt* query to the partner oracle of target oracle after both oracles are terminated.

Definition 2.3.2. strong Forward Secrecy

If a protocol has forward secrecy against an adversary who can know a LL-key during the execution of a key exchange protocol, and get any expired session keys of the target party, and get any session keys of other parties, then the protocol has *strong* Forward Secrecy. In this model, an adversary can use all queries, and *Corrupt* query can be sent to the partner oracle of a target oracle before both entities are terminated.

Furthermore, we define a security notion for key distribution protocols based on unknown key-share attacks, as proposed by S. Blake-Wilson *et al*.[50]. In this model, an adversary attempts to distribute different keys to each entity.

Definition 3. Secure against Unknown Key-Share Attacks

If the probability that oracles $\prod_{U_1}^{i}$ and $\prod_{U_2}^{j}$ accept and output different session key(s) is negligible, then the protocol is secure against unknown key-share attacks (UKSA). This includes the case where one entity outputs no session key.

3 Design of Analysis Method

Our analyzer checks authentication and key-distribution protocols based on rules that take into account provable security. Existing security analyzers have rules that define all secure/insecure conditions, and the analyzer has to verify that all flows and/or all possible states of the protocols are secure. Thus, optimization of rules and protocol descriptions is an important issue for these analyzers. Our analyzer, on the other hand, only checks essential transaction data, selected to satisfy provable security.

An overview of the adversary of security protocols is shown in Figure 1. In a provably-secure protocol, two kind of adversaries are assumed. One adversary receives either a session key or a random string, and must distinguish the two. The other adversary aims to make the target entity accepted by a flow.

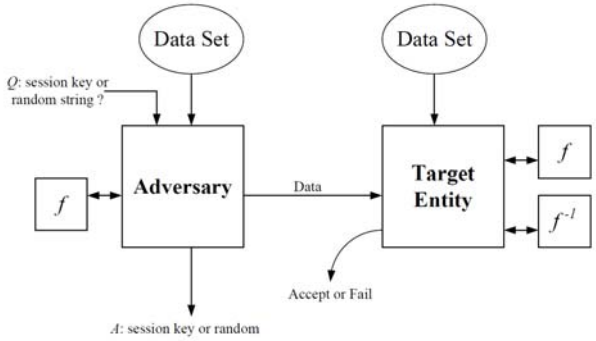


Fig. 1 Adversary of Security Protocols

In the discussion of the latter adversary, we focus on the protocol flow sent just before the entity changes to an accepting state. The adversary can obtain a polynomial-bound data set to eavesdrop on protocol flows and send flows to any entity. The data set includes keys such as previous session keys and LL-keys depending on the adversary's models. The adversary also uses some cryptographic function *f* which also depends on the kind of adversary.

Protocol specifications decide the data set that an adversary can obtain. Thus, the data set given to the adversary affects the probability that the adversary is successful. The analyzer first collects associated (essential) data from the protocol specification, and then computes the probability of a successful attack by the adversary.

Firstly, our analyzer extracts from a security protocol the data sent in each flow, labels the data with some properties, then extracts the essential data and verifies it with the analysis rules. If the essential data satisfies the rules, the protocol is secure.

The concept of checking essential data is based on proofs of security protocols. The security does not depend on other data or flows, only on the essential data. Thus, our analyzer is faster than previous analyzers.

In addition, the analyzer calculates the probability of a successful attack, and compares this to a configured security parameter. Thus, the analyzer can verify that the lengths of the essential data and the LL-key are large enough.

An overview of our analysis process follows: a user inputs a security protocol that consists of some flows, where each flow contains some data. The analyzer first extracts each chunk of data info and labels it with some properties, including the flow from which it originated, as shown in Figure 2. Then, the analyzer verifies the data according to the analysis rules.

In this section, we present our analysis methods, which are defined using the security notions discussed in the previous section.

3.1 Preliminary

Before the discussion, we describe the basic component of analysis rules.

3.1.1 Ideal Function *f*

In our analyzer, an ideal function *f* is an important factor for protocol verification. To verify the protocol, we assume that a security protocol uses secure cryptographic algorithms, and transform the algorithms into a function *f* that is an ideal cryptographic algorithm. The function *f* is an ideal pseudorandom function which takes data and a key as input, and outputs a string. The output is uniformly selected from a domain, and the same input and key generate the same output by the function *f*.

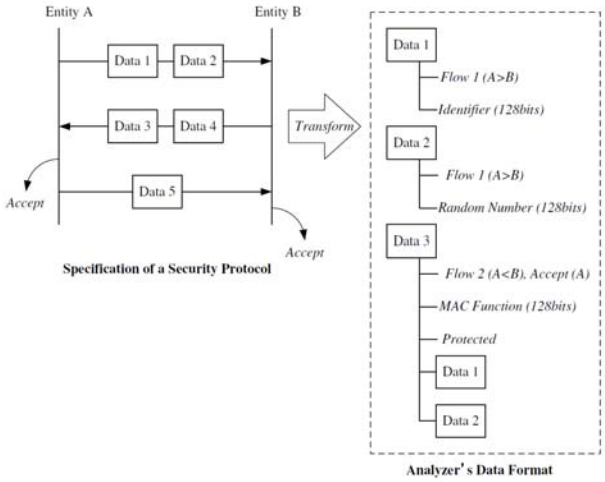


Fig. 2 Transformation from Specification to Analyzer's Data Format

For example, if all data has the same length, the adversary's job is to distinguish an *output* from a random string, and the adversary does not know the randomly selected input or a key, then *output* is indistinguishable from a random string.

Provably-secure public-key cryptographic algorithms provide different outputs even though the input and the key is same. However, we treats all algorithms as though they were the ideal function *f*; that is, we describe all algorithms with a uniform function.

We also model cryptographic functions as ideal functions where the function is nested. If a function is nested---that is, the function has the outputs of other functions as inputs---the analyzer roughly evaluates the function as a composite function. For example,

$$f(f'(x)) \,|\, f(\cdot): \{0,1\}^m \rightarrow \{0,1\}^n, n \geq |x|, |f'(\cdot)| \geq |x|$$

is evaluated as

$$F(x) \,|\, F(\cdot): \{0,1\}^m \rightarrow \{0,1\}^n$$

where the analyzer calculates the probability of successful attacks. Thus, the analyzer cannot compute the exact advantage of adversary, but it is useful for addressing the correctness of the protocols on an abstract (protocol-specification-based) level. Furthermore, this ideal model assumes every function is a random function; thus, a gap between proofs of the protocol and analyzer's evaluation result may exist. We discuss this in Section 6.

In our analyzer, a user defines a security parameter. The security parameter is defined based on the security requirements of the protocol. If the security parameter is *k*, the probability of a successful attack is at most $p(k) \cdot 2^{-k}$, where $p(k)$ is a polynomial. The security parameter is compared to the bit length of *X*, essential data such as outputs of the function *f*, to verify that a protocol is computationally secure.

Table 1: Example of Transformation Table

Symmetric	ECC-based	RSA,DSA
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

The length of a public/private key is mapped to the length of a secret key using a transformation table; see Table 1. The table may have to be changed as attacks on either class of primitives improve.

3.1.2 Data Properties

Data properties are used in analysis rules. We define some common data properties as follows:

- *Sent* : the data was sent in a previous flow.
- *Included*: the data is included in a flow; for example, as data, as a key, or as data input to a function.
- *Temporary*: the data is created dynamically; for example, a random number or a temporary public-key pair.
- *Protected*: the data is covered by the message authentication code or digital signature, the data is encrypted by a secret key, or the data that is inputted to a function which inputs are the data and data that is send with *Protected* and *Concealed* in a previous flow. It is computationally infeasible for an adversary to forge this data.
- *Concealed*: the data is encrypted by a LL-key (a secret key or a public key) and a random number that is sent with *Protected* and *Concealed*, and the data is not transmitted as plaintext. It is computationally infeasible for an adversary to obtain the data.

3.1.3 Additional Rules

Furthermore, the analyzer checks these additional rules as a basic verification:
- a temporary public keys are *Protected* when they are sent.
- all session keys are longer than the security parameter.
- all LL-keys are longer than the security parameter.

3.2 Verification of Authentication Protocols

Generally speaking, "authentication" means verifying that data calculated using an LL-key or related key, such as an authentication code, is correct. An adversary's goal is to send a correct authentication code (AC) to a target entity, and change the entity to the accepting state. The adversary's strategy is either to guess the correct authentication code or to reuse another authentication code. Thereby, the adversary's data set is previous ACs. Reuse of an AC means the adversary previously obtained the AC, and this condition implies replay attacks.

Now, we assume the authentication code is computed using the ideal function *f*. The probability that an adversary succeeds in guessing the correct authentication code depends on the length of LL-keys and the length of the authentication code. The probability that the reuse is successful depends on the randomness of the input data to the function *f*. If the input data is longer than the security parameter and is randomly selected, then the adversary cannot reuse because of the limitation on the adversary's memory. These two conditions imply that an output of the function *f* is indistinguishable from a random string.

With regards to authentication, we add the condition that the random data have to be generated by a target entity, because the target entity can verify the data and AC.

The analyzer evaluates the security of authentication protocols using the following steps. The analyzer selects a flow just before each entity changes to the accepting state, and then checks that the flow includes *protected* data for authentication such as a message authentication code, digital signature, and symmetric encryption. The analyzer also checks that the data is a random data generated by the target entity, and is longer than the security parameter.

3.3 Verification of Key Distribution Protocols

In this subsection, we propose analysis rules for key distribution protocols. Our analyzer separately checks six security notions: Semantic Security on the basic adversary model, Secure against UKSA, Secure against KKA, *basic* Forward Secrecy, *weak* Forward Secrecy, and *strong* Forward Secrecy.

The *basic* Forward Secrecy checks a conventional security notion of forward secrecy, which is not a provable security model. We use an analysis rule of *basic* Forward Secrecy as a primitive for all forward secrecy rules.

3.3.1 Semantic Security in the Basic Adversary Model

Semantic Security means no information is leaked by a key distribution protocol. The adversary either cannot send data including a session key, or the adversary can send the data but cannot obtain any information, whether or not the data is accepted. In addition, it is computationally infeasible for the adversary to compute the correctness of a session key from the communication logs.

The adversary's job is to distinguish whether a given string is a session key or not. The adversary can compute some data using functions. The adversary is also able to obtain responses to data sent to a target or its partner entity. Thus, the data set includes any results that can be obtained any data computed from a session key to target or partner entity; that is, the adversary cannot directly obtain a session key, but may be able to test whether the given string is a session key.

We discuss the adversary's potential strategies in this situation. If the data set only includes session key seeds---that is, the session key is not used on the key exchange protocol---the security of the protocol reduces to the security of the session-key generating function. We have only to check whether outputs of the key generation function are computationally indistinguishable from random strings.

The second strategy is to encourage the target entity to use a session key that is known or computed by an adversary.

The final strategy of the adversary is to try to distinguish session keys from random strings using the data set. Thus, if a key distribution protocol has semantic security, then either the session key is not directly exchanged in the protocol, or an entity accepts data including a session key. Furthermore, the probability that an adversary can distinguish a session key by using only communication logs is negligible. First, the analyzer checks whether a session key is *included*. If the session key is not *included*, the analyzer checks the length of the corresponding data; otherwise, the analyzer verifies that data including a correct session key is indistinguishable from data that does not include the session key.

3.3.2 Secure Against Unknown Key-Share Attacks

The essential data protecting UKSA is temporary data with no alteration and it is secure against replay attacks. It is used as input to a key distribution function that produces a session key. If each entity verifies that its seed is securely shared with other entities, then the adversary cannot alter or replace the session key. Thus, the analyzer checks whether temporary data included by the key distribution function is only the temporary data that *protected* and the *protected* function includes a challenge to protect against replay attacks or the temporary data generated by the entity. This rule is similar to the rule for mutual authentication: if exchanged random values in a mutual authentication protocol are used for the seed when generating a session key, the protocol is secure against UKSA.

3.3.3 Secure against Known Key Attacks

A protocol is secure against KKA if it has semantic security for a session key where the adversary knows previous session keys. The current session key then has no relation to a previous session key and the adversary cannot force an entity to share a known session key by alteration of communication data. The first condition is the same as Semantic Security for the basic adversary model. The second condition is security against a known-key sharing attack (called KKSA).

This condition is clearly weaker than security against UKSA. If a protocol is secure against UKSA, the protocol is also secure against KKSA; for the UKSA condition, the adversary attempts to persuade an entity to believe any session key, but in KKSA, the adversary has to persuade an entity to believe a *specific* session key which they want.

3.3.4 Forward Secrecy

Forward Secrecy means the adversary cannot compute a session key if the adversary later obtains the LL-key of an entity. In our analyzer, we additionally define the adversary of the (lowest) Forward Secrecy level as unable to obtain the communication logs of a key distribution protocol. Thus, if a key generation function includes at least one temporary chunk of data which has been exchanged in the protocol, the protocol has forward secrecy. The analyzer checks the above requirement.

3.3.5 *Weak* Forward Secrecy

An adversary of *weak* Forward Secrecy can obtain all communication logs and an LL-key. In w-FS model, an adversary can use *Corrupt* query after the protocol is finished, and the adversary cannot conduct any active attacks against the target pairs by using an LL-key. However, the adversary can eavesdrop on data flows and obtain data that is encrypted using the LL-key. Thus, the adversary has an advantage over the adversary of KKA, in that the adversary can decrypt data that has been encrypted with the LL-key and distinguish session keys from random strings; thus, the adversary can calculate the session key from communication logs as well as the encrypted data.
To be secure against an adversary of w-FS, a session key has to be calculated from secure data that the adversary cannot obtain in this situation.

Two types of data that the adversary cannot obtain are temporary data not exchanged in the protocol, and temporary data which is encrypted under a temporary public key and exchanged in the protocol. The temporary public/private key pair is erased when an entity finishes to calculating the session key. Thus, the encryption using a temporary public key is secure against the adversary. The former data implicitly indicates Diffie-Hellman key exchange by temporary key pairs. The session key of DH schemes is computed from a temporary private key that is not exchanged on during the protocol.

Thus, the analyzer checks that the key generation function of each entity for computing a session key includes secure data that is *Temporary* and not *Included*, or *Temporary* and *Concealed* using a temporary public key.

3.3.6 *Strong* Forward Secrecy

Strong Forward Secrecy has a more strict definition than *weak* Forward Secrecy. The adversary can get an LL-key and all internal state during a key distribution protocol by using a *Corrupt* query, and therefore is able to obtain a session key to corrupt a partner entity, when the session

key of the target entity is established but the key of the partner entity is not yet established. The adversary can obtain a temporary private key of the partner entity, that is, the adversary can obtain all data which the partner entity uses for computation of a session key. Thus, the adversary can easily compute the session key that the target entity has established.

The protocol requires a flow confirming that the session key calculation is complete. Thus, a session key is established when all entities finish computing the session key.

Note that the above verification rule is faithful to Bellare's model. We are currently studying a new definition of forward secrecy, that can provide a stricter security notion and can be imagined as an real-world adversary; we plan to present these results in another paper.

4. Implementation

In this section, we explain our implementation of the analyzer. Operations of existing analyzers are complicated, so we designed our analyzer such that a user could operate it easily without any special skills. A user can input and display security protocols using a graphical user interface (GUI). Our design criteria for implementation are as follows.

- Cryptographic algorithms are expressed using a common and simple description. A user can easily enter a cryptographic function without inputting a complex formula, since in our security protocol analysis, details of cryptographic functions are not considered.
- Data that has no influence on analysis are expressed using a uniform description. Security protocols are simplified by merging redundant data.
- A key generation function can be defined for each entity. A user can enter an asymmetric key distribution using the GUI.
- A user can input flows of a protocol with the GUI such as drawing tools.
- Trust is expressed as a property of an entity. A user defines the entity as trusted or non-trusted. Trusted entity need not to be authenticated by other entities.
- The analyzer saves the protocol and analysis results as XML files. A user can check and modify the protocol without the analyzer. Furthermore, the analysis is also translated into HTML allowing the user to print the analysis report using a web browser.
- The analyzer stores and shows known data, which indicates an entity's received and previously known data. This information helps a user to find security flaws.

Figure 3 shows the graphical user interface (GUI) of the analyzer. The right side shows a protocol overview, logs of received data and previously known data for each entity. On the left, all data, entities, and flows are described. If a user clicks on any category, the user can create new data, flows, and entities using dialog boxes. The user enters a type, a length, and also creates an entity, when the user adds new data by the GUI.

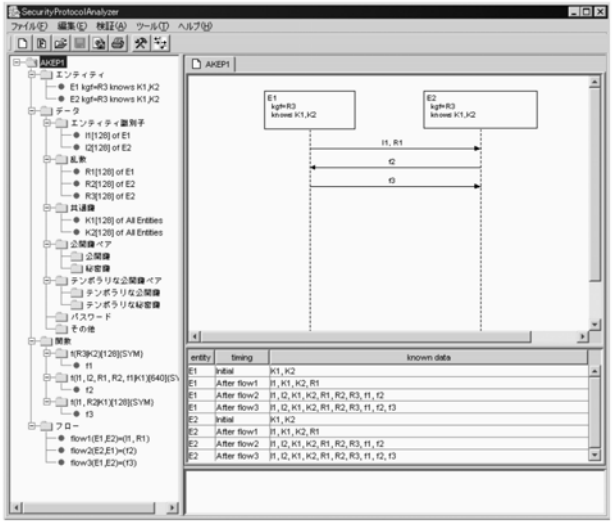


Fig. 3 Analyzer GUI

When a user creates a new function, the user also inputs its length, a key, an algorithm, and arguments. A sender, a receiver, and transported data are required to add a new flow. A user also enters a purpose of the protocol and the security parameter using the GUI, before the verification. Flows of a security protocol are divided into chunks of data, and the chunks are managed as objects in the program.

Our analyzer automatically makes a analysis report of the security protocol. An example of the report is shown in Figure 4.

Fig. 4 Example of Analysis Report

Table 2: Evaluation Results

Protocol	T(ms)	MA	SS	UK	KK	FS	wFS	sFS	Atk
Otway-Rees	≤ 10		X			X			X
Needham-Schroeder	≤ 10					X			X
ISO 11770-2(Protocol 1)	≤ 10		(X)			X			X
AKEP1	≤ 10	X	X	X	X	X			-
AKEP2	≤ 10	X	X	X	X	X			-
Password-Based AKEP	≤ 10	X	X	X	X	X	X	X	-

T: Transaction Time, MA: Secure Mutual Authentication, SS: Semantic Security on the Basic Model, UK: Secure against Unknown Key Share Attacks, KK: Secure against Known key Attacks, FS: Forward Secrecy, wFS: *weak* Forward Secrecy, sFS: *strong* Forward Secrecy, and Atk: Attacks have been proposed.

Table 3: Comparison of Processing Time

Analyzer	Year of Presentation	Time(msec)
AUTLOG[16]	1996	$7.8*10^5$
Athena[26]	1999	160
OFMC[24]	2003	151
STA[25]	2005	55
TRUST[25]	2005	63
ASPASyA[25]	2005	750
Proposed	2004	Less than 10

5. Evaluation Results

We implemented the security analyzer on a Pentium 4 3GHz PC. The total program size, including the XML parser, is about 1.5 MB. We chose six typical authenticated key distribution protocols with which to evaluate the analyzer. The average transaction time to evaluate one security protocol was less than 10 milliseconds (msec). The evaluation results are shown in Table 2. The broken protocols, Otway-Rees [51], Needham-Schroeder [52], and ISO11770-2[53] protocols which have been broken (see [54][55][56][1], respectively, for detailed attacks) were correctly identified as broken.

The check mark in parentheses below indicates that ISO 11770-2, without the optional flows that are used for mutual authentication and the flow leaks the session key, satisfies Semantic Security (SS). AKEP1, and AKEP2 [36] do not satisfy w-FS, nor s-FS. This result implies the symmetric-key-based key exchange cannot intrinsically satisfy those security notions either.

Password-based AKEP proposed by Bellare, Pointcheval, and Rogaway [41] satisfies all the required criteria. The result fits the results of their analysis in theoretical basis.

The evaluation shows that the analyzer can effectively verify security protocols.

Comparisons of the processing time with that of other analyzers are shown in Table 3. We calculated the average processing time for verifying a security protocol by referring to the papers in which the analyzers were presented. This comparison is not based on exactly the same conditions, because detailed information about each evaluation was not described in each paper. However, the evaluation result shows that our analyzer is not slower than other analyzers.

Our analyzer also checks the security of protocols progressively based on the adversary's ability in provable security. For example, our analyzer checks the semantic security of protocols according to four gradual notions, Semantic Security (basic adversary), Semantic Security (known key attack adversary), Semantic Security with weak-Forward Secrecy, and Strong-Forward Secrecy. Thus, a user can design and evaluate security protocols to use our analyzer, according to the condition that the protocol will be used.

Some existing analyzer requires generation of all possible messages by an adversary; however, our analyzer needs no generation of the massages. Algebraic approaches have a similar property to it. The algebraic approaches focuses on the protocol flow description, and our analyzer focuses on data property included in the protocol. Our analyzer has a fast verification process based on optimized rule sets.

Furthermore, no analyzer has user-friendly user interfaces. Our analyzer has a sophisticated GUI to input security protocols.

6. Considerations and Conclusion

In this paper, we presented a new security protocol analyzer based on provable security concepts. We also implemented the analyzer, and showed how the analyzer may be used for analyzing authentication and key distribution protocols.

The analyzer can progressively verify security of authentication and key distribution protocols according to

security notions in provable security. Some situations do not require a protocol to have the strongest security such as w-FS and s-FS. A user can choose a suitable security protocol for his/her security requirements and restrictions by using our analyzer.

Translation from an analysis result to a proof is an open issue. In our analyzer, all cryptographic functions are simplified to a pseudorandom function. Gaps between proofs and the results of our analyzer may exist because of this simplification. Even under the assumption that all real functions are ideally secure, the result of the analyzer is feasible; more detailed analysis will be needed to eliminate gaps between proofs and the analysis results.

Furthermore, the security notions of strong-Forward Secrecy based on BR model should be reconsidered. Related researches on provably-secure protocols have discussed only weak-Forward Secrecy as a security notion. It is an open problem whether s-FS is mandatory for a perfect secure key exchange protocol. The verification of our analyzer may be rather strict verification for security protocols. We continue the evaluation for various security protocols and will consider whether the verification rules can be relaxed.

In our future research, we will continue to consider new security notions for analysis rules and more detailed rules that encode properties of cryptographic functions.

References

[1] D. E. Denning, G. M. Sacco, "Timestamps in key distribution protocols", Communication of the ACM, 24(8), pp. 533-536, 1981.

[2] R. Needham, M.Schroeder, "Authentication Revisited, ACM Operating Systems Review", Vol.21, No.1, 1987.

[3] L. Gong, R. Needham, R. Yahalom, "Reasoning about belief in cryptographic protocols", Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy, pages 234-248. 1990.

[4] C. Meadows, "Formal Verification of Cryptographic Protocols: A Survey", Proc of ASIACRYPT'95, LNCS, Vol.917, pp.133-150, 1996.

[5] D. Delve and A. Yao, "On the Security of Public Key Protocols", IEEE Transactions on Information Theory, Vol.29, No.2, pp.198-208, 1983.

[6] J. K. Millen, S. C. Clark, and S. B. Freedman, "The Interrogator: Protocol Security Analysis", IEEE Transactions on Software Engineering, Se-13, No.2, 1987.

[7] D. Longley and S. Rigby, "An Automatic Search for Security Flaws in Key Management Schemes", Computers and Security, Vol.11, No.1, pp.75-90, 1992.

[8] R. Kemmerer, C. Meadows, and J. Millen, "Three Systems for Cryptographic Protocol Analysis", Journal of Cryptology, Vol.7, No.2, 1994.

[9] C. Meadows, "A system for the Specification and Analysis of Key Management Protocols", Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy, pp. 182-195, 1991.

[10] C. Meadows, "Applying Formal Methods to the Analysis of a Key management Protocol", Journal of Computer Security, Vol.1, pp.5-53, 1992.

[11] J. C. Mitchell, M. Mitchell, U. Stern, "Automated analysis of cryptographic protocols using Murφ", Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy, pp.141-151, 1997.

[12] M. Burrows, M. Abadi, R. Needham, "A Logic of Authentication", ACM Transactions on Computer Systems, Vol. 8, No. 1, 1990.

[13] D. M. Nessett, "A Critique of the Burrows, Abadi, and Needham Logic", Operating Systems Review, Vol.24, No.2, pp. 35-38, 1990.

[14] L. Gong, R. Needham, and R. Yahalom, "Reasoning about Belief in Cryptographic Protocols", Proc. of the 1990 IEEE Computer Society Symposium on Security and Privacy, pp.234-248, 1990.

[15] P. F. Syverson, and P.C. van Oorschot, "On Unifying Some Cryptographic Protocol Logics", Proc of the 1994 IEEE Computer Society Symposium on Security and Privacy, pp.14-28, 1994.

[16] D. Kindred, J. M. Wing, "Fast Automatic Checking of Security Protocols", Proc. of the second USENIX Workshop on Electronic Commerce, pp.41-52, 1996.

[17] A. Mathuria, R. Safavi-Naini, and P. Nickolas, "On the Automation of GNY logic", Proc. of the 1995 Australian Computer Conference, pp.370-379, 1995.

[18] L. Paulson, "The inductive approach to verifying cryptographic protocols", Journal of Computer Security, Vol.6, pp. 85-128, 1998.

[19] A. H. Dekker, "C3PO: A Tool for Automatic Sound Cryptographic Protocol Analysis"v, Proc. of the 13th IEEE Computer Security Foundations Workshop, pp.77-87, 2000.

[20] F. Javier Thayer Fabrega, J. C. Herzog, and J. D. Guttman, "Strand space: Why is a security protocol correct? ", Proc. of the 1998 IEEE Computer Society Svmposium on Security and Privacy, pp.160-171, 1998.

[21] D. Song, "Athena: a new efficient automatic checker for security protocol analysis", Proc. of the 20th IEEE Computer Security Foundations Workshop, pp.192-202, 1999.

[22] M. Boreale, "Symbolic trace analysis of cryptographic protocols", Proc. of ICALP 2001, LNCS, Vol.2076, pp.667-681, 2001.

[23] R. Amadio, D. Lugiez, and V. Vancackere, "On the symbolic reduction of processes with cryptographic functions", Technical report, INRIA-Sophia, 2001.

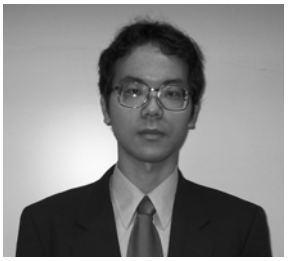
[24] D. Basin, S. Modersheim, and L. Vigano, "Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols", Proc. of the 2003 Workshop on Security Protocols Verification (SPV'03), 2003.

[25] A. Bracciali, G. Baldi, G. Ferrari, and E. Tuosto, "A Coordination-based Methodology for Security Protocol Verification", Proc. of 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP04), Elect. Notes in Theoret. Comp. Science vol. 121, Elsevier Science, 2005.

[26] M. Toussaint, "Deriving the Complete Knowledge of Participants in Cryptographic Protocols", Proc. of CRYPT'91, LNCS, Vol. 576, pp.24-43, 1992.

[27] M. Toussaint, "Separating the Specification and Implementation Phases in Cryptology", Proc. of ESORIC'92, LNCS, Vol. 638, pp. 77-102, 1992.
[28] C. Meadows, "Representing Partial Knowledge in an Algebraic Security Model", Proc of the IEEE Computer Security Foundations Workshop III, PP. 23-31, 1990.
[29] A. W. Roscoe, J. C. P. Woodcock, L. Wulf, "Non-interference through determinism", Proc. of ESORICS'94, LNCS, Vol.875, pp. 33-53, 1994.
[30] A. W. Roscoe, "CSP and determinism in security modeling", Proc of Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy, 1995.
[31] G. Lowe, "Casper: A Compiler for the Analysis of Security", Journal of Computer Security, Vol. 6, p.p. 53-84, 1998.
[32] A. W. Roscoe, "Modeling and verifying key-exchange protocols using CSP and FDR", Proc. of 8th IEEE Computer Security Foundations Workshop, pp.98-107, 1995.
[33] J. K. Millen and G. Denker, "CAPSL and MuCAPSL", J. of Telecommunications and Information Technology, April, 2002.
[34] A. D. Gprdon, and M. Abadi, "A calculus for cryptographic protocols: The spi calculus", Information and Computation, Vol.138, pp.1-70, Academic Press, 1999.
[35] V. Varadharajan, "Verification of Network Security protocols", Computers and Security, pp.693-708, 1989.
[36] M. Bellare, P. Rogaway, "Entity authentication and key distribution", Proc. of CRYPTO'93, LNCS, Vol.773, pp. 232-249, 1994.
[37] M. Bellare, P. Rogaway, "Provably secure session key distribution: thethree party case", Proc. of 27th Annual Symposium on the Theoryof Computing, ACM, pp. 57-66, 1995.
[38] S. Blake-Wilson, D. Johnson, A. Menesez, "Key agreement protocols and their security analysis", Proc. of 6th IMA International Conference on Cryptography and Coding, LNCS, Vol. 1355, pp.30-45, 1997.
[39] S. Blake-Wilson, A. Menesez, "Entity authentication and key transport protocols employing asymmetric techniques", Proc. of 5th International Workshop on Security Protocols, LNCS, Vol.1361, pp. 137-158, 1997.
[40] S. Blake-Wilson, A. Menesez, "Authenticated Diffie-Hellman Key Agreement Protocols", Proc. of SAC'98 (Invited Talk), LNCS, Vol.1556, pp. 339-361, 1998.
[41] M. Bellare, D. Pointcheval, P. Rogaway, "Authenticated Key ExchangeSecure Against Dictionary Attacks", Proc. of Eurocrypt'00, LNCS, Vol. 1807, pp. 139-155, 2000.
[42] V. Shoup, A. Rubin, "Session-key distribution using smart cards", Proc. of Eurocrypt'96, LNCS, Vol.1070, pp. 321-31, 1996.
[43] M. Bellare, R. Canetti, H. Krawczyk, "A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols", Proc. of the 30th Annual Symposium on the Theory of Computing, pp.419-428, ACM, 1998.
[44] M. Backes, B. Pfitzmann, M. Waidner, "A Composable Cryptographic Library with Nested Operations", Proc. of 10th ACM Conference on Compuer and Communications Security, pp.220-230, 2003.
[45] M. Backes, B. Pfitzmann, M. Waidner, "Symmetric Authentication Within a Simulatable Cryptographic Library", Proc. of ESORIC'03, LNCS, Vol.2808, pp.271-290, 2003.
[46] M. Backes, "Unifying Simulatability Definitions in Cryptographic Systems under Different Timing Assumptions", Proc. of CONCUR'03, LNCS, Vol.2761, pp.350-365, 2003.
[47] M. Backes, B. Pfitzmann, "Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library", 17th IEEE Computer Security Foundations Workshop, 2004.
[48] V. Shoup, "On Formal Models for Secure Key Exchange", Theory of Cryptography Library Record 99-12, and invited talk at ACM Computer and Communications Security conference, 1999.
[49] R. Canetti, H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels", Proc. of Eurocrypt'01, LNCS, Vol.2045, pp. 453-474, 2001.
[50] S. Blake-Wilson, A. Menezes, "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol", Proc. of PKC'99, LNCS, Vol.1560, pp.154-170, 1999.
[51] D. Otway, O. Rees, "Efficient and Timely Mutual Authentication, Operating Systems Review", Vol.21, No.1, pp. 8-10, 1987.
[52] R. Needham, M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", Communications of the ACM, Vol.21, pp.393-399, 1978.
[53] ISO, "Information technology-security techniques-key management-part 2:Mechanisms using symmetric techniques", ISO/IEC 11770-2, 1996.
[54] W. Mao, C. Boyd, "Development of Authentication Protocols:Some Misconceptions and a New Approach", Proc. of 7th Computer Security Foundations Workshop, pp. 178-186, 1994.
[55] G. Wang, S. Qing, "Two New Attacks Against Otway-Rees Protocol, In IFIP/SEC2000 Information Security", 16th World Computer Congress 2000, pp.137-139, 2000.
[56] Z. Cheng, R. Comley, "Attacks On An ISO/IEC 11770-2 Key Establishment Protocol", IACR e-Print Archive, 2004.

Shinsaku Kiyomoto received his B.E. in Engineering Sciences, and M.E. in Materials Science, from Tsukuba University, Japan, in 1998 and 2000 respectively. He joined KDD (now KDDI) and has been engaged in the research on stream cipher, cryptographic protocol, and mobile security. He is currently a researcher of the Information Security Lab. in KDDI R & D Laboratories Inc. He received his doctorate of engineering from Kyushu University in 2006. He received the Young Engineer Award from IEICE in 2004. He is a member of JPS, IEICE, and IPSJ.



Haruki Ota received the B.E., Department of Computer Science, and M.E., Department of Communications and Integrated Systems, from Tokyo Institute of Technology, Japan, in 2000 and 2002 respectively. He joined KDDI and has been

engaged in research on cryptographic protocol, biometrics, and information security. He is currently a research engineer of the Information Security Lab. in KDDI R & D Laboratories Inc. He is a member of IEICE and IPSJ.

Toshiaki Tanaka received B.E. and M.E. degrees in communication engineering from Osaka University, Japan, in 1984 and 1986 respectively. He joined KDD (now KDDI) and has been engaged in the research on cryptographic protocol, mobile security, digital rights management, and intrusion detection. He is currently a senior manager of the Information Security Lab. in KDDI R & D Laboratories Inc. He received his doctorate of engineering from Kyushu University in 2007. He is a member of IEICE and IPSJ.

Appendix

A.1 AKEP1 and Our Analyzing Steps

In this section, we show an example of relation between proofs and our analysis rules. First, we describe the AKEP1 protocol proposed by Bellare and Rogaway and an overview of its proof, and then we explain the steps of our analyzer takes as compared to the proof.

Figure 5 shows the AKEP1 protocol, where A and B are identifiers of Entity A and Entity B, R_A and R_B are random numbers, s_1 and s_2 are pre-shared secret keys, $Mac[X]k$ indicates a message authentication code (MAC) of X using a key k, and $Enc[X]k$ indicates the encryption of X using a key k. A session key is indicated by a. $X.Y$ denotes the concatenation of X and Y. This protocol is a typical 3-way authenticated key exchange protocol.

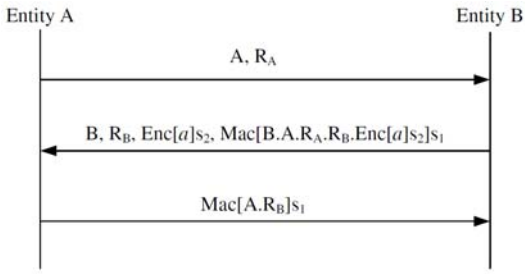


Fig. 5 AKEP1 Protocol

We introduce an overview of the proof of AKEP1 for secure mutual authentication. AKEP1 is a provably-secure authentication protocol under a MAC function that is assumed to be a secure pseudorandom function. An output of a secure pseudorandom function is indistinguishable from that of a truly random function. The proof reduces the security of the authentication protocol to the security of the pseudorandom function. That is, they prove a machine that runs in polynomial time can be constructed by using an adversarial machine that can break the AKEP1 authentication protocol using the pseudorandom function (called a real experiment). The adversary can break in polynomial time under the real experiment, but fails in a random experiment in which a pseudorandom function replaces to a truly random function.

This machine fails to distinguish the pseudorandom function when the adversary can break the security protocol using a truly random function.

Now, we discuss an adversary who spoofs the entity B in AKEP1, when the MAC function replaces a truly random function. The probability that the attack is successful is separated into two cases;

(a) Successful forgery of
$Mac\left[B.A.R_A.R_B.Enc\left[a\right]s_2\right]s_1$.
This probability is expected to be 2^{-k}, where k is the length of the MAC.

(b) The adversary E previously obtains
$Mac\left[B.A.R_A.R_B.Enc\left[a\right]s_2\right]s_1$ happening. In this case, R_A is included in a list that the adversary E makes while recording communications between entities. The probability is at most $(T_E(k)\text{-}1)\cdot 2^{-k}$, where $T_E(k)$ denotes a polynomial bound on the number of communications made by E.

The total probability that the adversary successfully spoofs the entity B in AKEP1 is at most $T_E(k)\cdot 2^{-k}$. In similar fashion, the probability that an adversary successfully spoofs the entity A is at most $T_E(k)\cdot 2^{-k}$. Thus, an adversary breaks AKEP1 with a truly random function.

This probability is included in the probability that the machine trying to distinguish the pseudorandom function from the truly random one failed, because the probability means that the adversary breaks the protocols in the random experiment.

Our analyzer calculates the probability as follows;

(1) The analyzer finds input data that changes the entity to an accepting state and is *protected*; the data is $Mac\left[B.A.R_A.R_B.Enc\left[a\right]s_2\right]s_1$ and $Mac\left[A.R_B\right]s_1$ in AKEP1.

(2) The analyzer extracts the random numbers that are sent by the entity in a previous flow; that is R_A for $Mac\left[B.A.R_A.R_B.Enc\left[a\right]s_2\right]s_1$ and R_B for $Mac\left[A.R_B\right]s_1$. Then, the analyzer calculates the length of the random numbers and compares with the security parameter. This operation corresponds to checking the event (b) in the proof.

(3) The analyzer checks the length of
$Mac\left[B.A.R_A.R_B.Enc\left[a\right]s_2\right]s_1$ and $Mac\left[A.R_B\right]s_1$.

This operation corresponds to checking the event (a) in the proof.

The analyzer checks the security of the protocol based on the proof.

A.2 Detailed Algorithms of Verification

In this section, we describe detailed algorithms of protocol verifications. All verifications are executed for each entity. $X \leftarrow Y$ indicates Y is inputted to X such as data, a flow, and a result. $flow_s(U,U')$ indicates the s-th flow from U to U' in a protocol. The symbols, *data*, *LLK*, *SK*, *TPK*, and *k* indicates data or a function, a LL-key, a session key, a temporary public key, and a security parameter, respectively. f_x denotes a function and it is used to specify a function. $Result_U$ indicates the verification result of U. The symbol $\neg$ means "not"; for example, $\neg$*included* means that data has a *not included* property.

A.2.1 Secure Mutual Authentication

We analyze Secure Mutual Authentication of authentication protocols as follows. The description "Π_U=*accept* by $flow_s$" means the $flow_s$ change the entity Π_U to the accepting state.

input: $data_t | data_t \in flow_s(U', U)$
where $flow_s(U', U) | \prod_U$=*accept* by $flow_s$
output: $Result_U = \{Good, Bad\}$

$Result_U \leftarrow Bad$
for $data_t$ **do**
 if $data_t = \{temporary, sent\}$
 and $data_t = \{protected\}$ by f_x
 and $data_t \in flow^{s'}(U, U') | s' < s$
 and $|data_t| > k$
 and $|f_x| > k$
 then
 $Result_U \leftarrow Good$
 else if $data_t = \{temporary, sent\}$
 and $data_t \in flow^{s'}(U, U') | s' < s$
 and $data_t = \{concealed\}$ by f_x before s
 and $|data_t| > k$
 and $|f_x| > k$
 then
 $Result_U \leftarrow Good$

A.2.2 Semantic Security on the Basic Adversary Model

We analyze Semantic Security of key distribution protocols as follows.

input: $SK, data_t | SK \leftarrow data_t, data_{t'} | data_{t'} \leftarrow SK,$
$data_{t''} | data_{t'} \leftarrow data_{t''}$
output: $Result_U = \{Good, Bad\}$

$Result_U \leftarrow Bad$
if $SK = \{\neg included\}$ in all $flow$
 then
 for $data_t$ **do**
 if $data_t = \{temporary\}$ and $|data_t| > k$
 then
 $Result \leftarrow Good$
 else if $data_t = \{temporary\}$
 and $data_t = \{concealed\}$ by f_x
 and $|data_t| > k$ and $|f_x| > k$
 then
 $Result \leftarrow Good$
else if $SK = \{temporary\}$
 and $SK = \{protected\}$ by f_x
 and $SK = \{concealed\}$ by $f_{x'}$
 and $|f_x| > k$
 then
 $Result_U \leftarrow Good$
else
 for $data_{t'}$ **do**
 $Result_U \leftarrow Bad$
 for $data_{t''}$ **do**
 if $data_{t''} = \{tenmporal, \neg included\}$
 and $|data_{t''}| > k$
 then
 $Result_U \leftarrow Good$
 else if $data_{t''} = \{temporal, included\}$
 and $data_{t''} = \{protected\}$ by f_x
 and $data_{t''} = \{concealed\}$ by $f_{x'}$
 and $|data_{t''}| > k$ and $|f_x| > k$
 then
 $Result_U \leftarrow Good$

A.2.3 Secure against Unknown Key Share Attacks

Each entity is checked for security against UKSA. If all entities are secure against UKSA, then the protocol is secure against UKSA.

input: SK, $data_t|SK \leftarrow data_t$, $data_{t'}$
where $f_x|f_x \leftarrow data_t$ and $data_{t'}$
output: $Result_U = \{Good, Bad\}$

$Result \leftarrow Bad$
if SK is generated by U
then
$Result \leftarrow Good$
else
for $data_t$ **do**
$Result \leftarrow Bad$
if $data_t = \{temporary\}$
and $|data_t| > k$
then
if $data_t$ is generated by U
then
$Result \leftarrow Good$
else
for $data_{t'}$ **do**
if $data_{t'} = \{temporal\}$
and $data_{t'}$ is generated by U
and $data_t = \{protected\}$ by f_x
then
$Result \leftarrow Good$

A.2.4 Secure against Known Key Attacks

The analyzer checks a protocol's security against KKA as follows.

input: $Result_U$ of SS, and $Result_U$ of UK
output: $Result_U = \{Good, Bad\}$

$Result \leftarrow Bad$
if $Result_U$ of SS= $Good$
and $Result_U$ of UK= $Good$
then
$Result_U \leftarrow Good$

A.2.5 Forward Secrecy

The analyzer evaluates forward secrecy as follows.

input: SK, $data_t|SK \leftarrow data_t$
output: $Result_U = \{Good, Bad\}$

$Result \leftarrow Bad$
if $SK = \{temporary\}$
then
$Result \leftarrow Good$
else if $data_t = \{temporary\}$
and $|data_t| > k$
then
$Result \leftarrow Good$

A.2.6 *weak* Forward Secrecy

The analyzer checks the following steps.

input: $data_t|SK \leftarrow data_t$, $Result_U$ of KK, $Result_U$ of FS
output: $Result_U = \{Good, Bad\}$

$Result \leftarrow Bad$
if $Result_U$ of KK = $Good$
and $Result_U$ of FS
then
for $data_t$ **do**
if $data_t = \{temporary, \neg included\}$
and $|data_t| > k$
then
$Result_U \leftarrow Good$
else if $data_t = \{temporal\}$
and $data_t = \{concealed\}$ by f_x using TPK
and $|data_t| > k$ and $|TPK| > k$
then
$Result_U \leftarrow Good$
else
return

A.2.7 *strong* Forward Secrecy

The analyzer evaluates strong Forward Secrecy as follows.

input: all $flow$, $Result_U$ of wFS
output: $Result_U = \{Good, Bad\}$

$Result \leftarrow Bad$
if $Result_U$ of wFS = $Good$
and $flow_s$ exists after session key is
established by both entities
then
$Result_U \leftarrow Good$