ON-THE-FLY AUTOMATIC GENERATION OF SECURITY PROTOCOLS

Shinsaku Kiyomoto, Haruki Ota and Toshiaki Tanaka
KDDI R & D Laboratories Inc., 2-1-15 Ohara, Fujimino-shi Saitama, 356-8502, Japan

Keywords: Security Protocol, Automatic Generation, On-The-Fly Generation, Authentication, Key Exchange.

Abstract: In this paper, we presented an automatic generation tool for authentication and key exchange protocols. Our tool generates a security protocol definition file from input data of requirements for the protocol. The tool constructs a new protocol that combines a pieces of security protocols as building blocks. The transaction time of the tool is less than one second and it is able to realize more rapid generation of security protocols in comparison to existing generation tools.

1 INTRODUCTION

1.1 Background

Ubiquitous is a key phrase often used in relation with new generation IT services. In a ubiquitous network, multi-networks, multi-devices, and multi-services exist. Many services have been provided over communication networks such as the Internet and local communications. In these services, there are many security protocols used for authentication and key exchange that realize secure communication. The security protocol is pre-defined and installed in all entities before entities commence to communicate. Secure communication with unknown entities is impossible where both entities know no common authentication and key exchange protocol. This is a disadvantage in terms of the flexibility of ubiquitous environments.

On the other hand, the design process of a security protocol is often cumbersome and expensive. Even though security protocols are carefully designed, security flaws sometimes occur in the protocol. This must be a serious bottleneck and poses a risk in the construction of new services and may delay or halt provision of such services. If the protocol is flawed, high costs might be incurred, due to forced redesign and update process. The designer may not be able to devise the optimal protocol for given system requirements. Namely, conservative designers might incorporate overly heavy functions just to ensure the protocols are secure. Thus, schemes for automatically generating security protocols have been considered.

If both entities have a function to generate security protocols automatically, the entities can communicate with each other securely. The functions enable an entity to communicate with any other entity at any time and in any place, without previously sharing a common security protocol.

However, existing automatic protocol generation techniques are impractical because the techniques require huge transaction time to generate a security protocol.

1.2 Our Contribution

In this paper, we propose a tool that is capable of generating security protocols automatically. Our tool generates two-party authentication and key exchange protocols to combine medium-size building blocks called components. Previous tools constructs security protocols by randomly to choosing small blocks for security protocols; the existing methods are not suitable for fast generation of security protocols, because the methods have to check a huge number of candidates constructed by the small blocks. Our tool first selects medium-size blocks in accordance with requirements for protocol generation and constructs a security protocol using selected blocks. The tool also has an efficient optimization process to avoid that redundant flows and data is remained in the generated protocol specifications. Thus, an efficient security protocol definition is produced by the tool, although our tool constructs a security protocol using the medium-size blocks.

The transaction time of existing tools are from several tens of second to several hours; however, our tool realizes security protocol generation within one second. Therefore, the tool is able to realize more rapid generation of security protocols in comparison to existing generation tools.

The remainder of this paper is organized as follows: Section 2 introduces works related to automatic security protocol generation. Next, we present a tool for generating security protocols automatically in Section 3. Evaluation results are shown in Section 4. Finally, we conclude this paper in Section 5.

2 RELATED WORK

Automatic security protocol generation is divided into two steps; the first step is generation of a high level protocol definition from security and performance requirements, and the second step is source code generation from the protocol definition by means of a security protocol compiler. Programs for the first step denote **tool**, and programs for the second step denotes **compiler**.

There are some compilers for generating implementations of security protocols that use cryptographic algorithms. F. Muller and J. Millen proposed automatic Java code generation from CAPSL or CIL specification languages. The tool is used for cryptographic protocols that only use symmetric key encryption. ACG (automatic Code Generator) is a part of the AGVI toolkit(Song et al., 2001) and it is an automatic compiler that translates the high level specifications of security protocols into Java source code. COSP-J(Didelot, 2003) is a security protocol compiler that takes a description of a security protocol in a simple, abstract language, and produces a Java implementation of the same protocol based on Casper(Lowe, 1997). ACG-C# produces C# implementation codes for security protocols with Casper. Spi2Java(Pozza et al., 2004) automatically generates Java code implementation cryptographic protocols described in the formal specification language spi calculus(Abadi and Gordon, 1999). SPEAR II(Lukell and Veldman, 2003) provides Java code generation from an abstract protocol specification. These tools require the protocol specification to be described in their specific language.

On the other hand, very little research has been conducted on tools for automatic protocol generation. A. Perrig and D. Song proposed an automatic protocol generation tool(Perrig and Song, 2000a). A protocol designer inputs the specification of the desired security properties and the system requirements and generates a high-level protocol definition using the tool. The tool first generates all protocols below a given cost threshold inputted by the designer. After sorting the generated protocols, the tool tests them. If one protocol satisfies the desired properties, the generation process can stop. Otherwise, more protocols are generated with an increase in the cost threshold and they are tested again. They extended their method so it can be applied for 3-party protocol and improve space reduction techniques of candidate protocols(Perrig and Song, 2000b). However, their scheme is not efficient because their scheme first generates many security protocol candidates and then chooses one protocol to test if the protocol fits the requirements. S. N. Foley and H. Zhou proposed a belief logic approach that allows entities to realize on-the-fly generation of security protocols(Zhou and Foley, 2003). Their automatic security protocol generator uses logic-based synthesis rules to guide it in a backward search for suitable protocols from protocol goals(Foley and Zhou, 2003). Their tool is more efficient than the previous one; however, it is still too slow for the dynamic generation of security protocols because all possible states are searched to generate sub-protocols, and the number of potential protocol candidates is small because of binding free variables only to formulae from known assumptions inputted as a requirements for the security protocol. Furthermore, detailed requirements have to be written in a special language.

In this paper, we present a tool for automatic security protocol generation. The tool generates security protocols from a small set of requirements using a new approach whereby the tool constructs new protocols based on the components that are pieces of protocols.

3 AUTOMATIC SECURITY PROTOCOL GENERATION

Generally, security protocols are designed taking requirements for security and performance into consideration. Our automatic protocol generation tool generates a security protocol based on the requirements for the protocol. The requirements are inputted into the tool, and the tool then automatically generates a security protocol definition for security protocol compilers. In this section, we first explain the language of security protocol definition; next, we present the whole process of security protocol generation; finally, we shows details of the selection process for components and data sets in the generation process and generation process of session key generation functions.

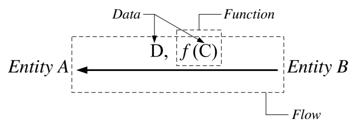

Figure 1: Flow, Function, and Data.

3.1 Language of Security Protocol Definition

The protocol definition is a high-level protocol specification written in XML language. The high-level definition makes it easy to write flows and calculations in the protocol and it is assumed to be independent from environments. Furthermore, the XML description makes it easy to add specific information to the high-level protocol definition. For example, if they wish, protocol designers can describe detailed parameters such as those for cryptographic algorithms and the initial values of variables. The description is highly flexible with regard to adding and removing information for a basic security protocol definition generated by the tool; thus, the protocol specification can include not only a common, high-level specification of the security protocol but also specific information for environments. This flexibility allows compilers to be customized for particular environments such as smart card. The protocol definition file can be used for a range of security protocol compilers.

We describe security protocols based on definitions of data, functions, and flows as Figure 1. A flow is described as all data sending by one transaction between entities and information of sender and receiver. A function indicates a cryptographic operation and it is described as input data and algorithm name. Data is minimum blocks to construct a security protocol definition. Security protocols definition consist of several flows, and these flows consist of data and functions.

Furthermore, we describe additional information in the protocol definitions, such as purpose of the protocol, information of entities on a header field. The definition file includes information for all entities which execute the protocol. Our security protocol definition is divided into five parts:

- Header Information Field
  The header information field defines the purpose of a protocol such as authentication and key exchange, and security parameters.
- Entity Information Field
  The entity information field stores information on each entity. Entity information includes which entity should be authenticated, role of the entity (initiator of the protocol or responder to the protocol), and the data list that the entity knows before the protocol starts. The information also includes *key generation function info* and *validation function info*. The *key generation function info* refers to a function for generating a session key in a key exchange protocol, and the *validation function info* refers to a function for authentication whereby the entity computes the function to verify the authenticity of an entity.
- Data Information Field
  The data information field is for definition of data. Data information consists of data-id, data type, length, creator, label, algorithm, value, usage, source, store, and comment as shown Table 1.
- Function Information Field
  The function information field consists of function definitions. Function information includes function identifier, algorithm type, length of output, and input data. Function information optionally includes detailed information for a cryptographic algorithm such as OID and padding type.
- Flow Information Field
  All protocol flows and computations between flows are written in the flow information field. Flow information includes information of sender, receiver, and sending data and function.

The data type *Identifier*, *Random*, *Password*, *Counter*, and *Text* indicates that the data is an identifier of an entity, random number, password, counter value, and other type of data, respectively. *Temporal Public Key* and *Temporal Private Key* indicates that the data are respectively a public key and private key generated in the protocol.

Functions for the security protocol are defined as the function-id, input data, function type, and length of the output. Flows of the security protocol are defined as flow-id, sender, receiver, and sending data and functions. Data and functions can be referred to data-id and function-id. A flow has a special label for defining action after receiving the flow. For example, a label ”*for authentication*” indicates that the flow is used when the receiver entity judges whether the sender entity is valid or not. A label ”*for key exchange*” means that the receiver entity calculates a session key after receiving the flow. A function for each entity is defined as a special function calculating a session key in the entity information field.

3.2 Requirements for Security Protocols

Our tool generates security protocol definitions from requirements for security protocols. The following parameters are inputted to the tool as security protocol requirements:

Table 1: Data Information.

Field	Man. Opt.	Content
data-id	mandatory	Identifier of data
data type	mandatory	Data type selected from {*Identifier, Random, Text, Counter, Password, Symmetric Key, Public Key, Private Key, Temporary Public Key, Temporary Private Key,* }
length	mandatory	Length of the data
creator	mandatory	Name of creating the data: entity names or public
label	optional	Global name of the data that is used for other security definition files
algorithm	optional	Related algorithm. This field is mandatory, where the data is public key, private key, temporary public key, or temporary private key.
usage	optional	Usage of the data
value	optional	(Initial) value of the data
source	optional	File name or pass to load the data as a initial value
store	optional	File name or pass to save the data when the protocol is finished.
comment	optional	For arbitrary description

- Trustworthiness of entities
  This parameter indicates which an entity should be authenticated. An entity omits the authentication process in the case where the other entity is trusted. Thus, this parameter determines whether the protocol is a single-authentication protocol or mutual- authentication protocol.
- Number of flows
  This parameter indicates the maximum number of flows in the protocol.
- Known keys of entities
  This parameter defines the keys that each entity has. For example, the condition where two entities have a common shared key is described.
- Lists of available cryptographic algorithms in each entity
  This parameter shows a list of cryptographic algorithms that an entity can compute.
- Security parameters
  These parameters choose the security level required for the protocol and assumed adversary models (attack). The security level is defined as the length of a secret key. The length of public/private keys is reduced to the length of a secret key using reduction tables in the protocol generation process. The lengths of other data such as random numbers are also decided according to the level. Our tool has three adversary models: *Known Key Attack*, *Forward Secrecy*, *Unknown Key Share Attack*. The first model assumes that the adversary can obtain previous session keys, and the second model assumes that the adversary can obtain the secret keys and/or private keys of entities after the protocol is terminated. The parameters determine if the tool has to consider each adversary model. The tool generates a security protocol that achieves the security level by means of semantic security of a session key and authenticity under selected adversary models. Details of adversary models are shown in the papers (Bellare and Rogaway, 1994) (Bellare and Rogaway, 1995) (Blake-Wilson et al., 1997).
- Computational power of each entity (optional)
  This parameter is optional. This parameter defines the computational power of each entity. We define the computational power to be one of three levels: Low (assumed for mobile terminals or smart cards), Medium (assumed for laptop PCs), High (assumed for high-end servers). The tool selects appropriate cryptographic algorithms and protocols according to the parameters.
- Size restriction for one flow in the protocol (optional)
  This parameter is optional. This parameter defines the size restriction for one flow. The tool decides data and length of data in flows and divides one flow into two flows according to the parameter.

The two optional parameters used for execution environments. For example, if the protocol has to be used on a smart card, the protocol should be lightweight and data size in one flow is restricted.

The parameters are inputted as XML format data to the tool.

3.3 Automatic Generation Procedure

In this subsection, we explain the whole process of security protocol generation. In our tool, security protocols are generated to combine building blocks called **components**. Specifications of components are defied using **data sets**. We can make several building blocks to replace the data sets at the components. Figure 3 and Figure 4 show component overviews for key-exchange and authentication protocols, and Table 2 shows examples of the data sets. The automatic protocol generation procedure is shown in Figure 2. A protocol is generated from a file that describes the

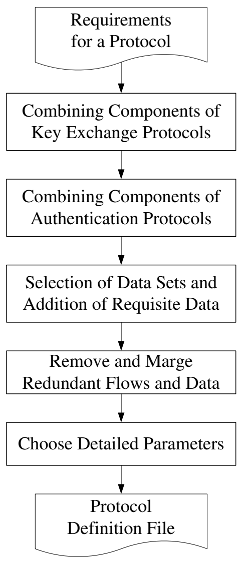
Requirements for a Protocol
Combining Components of Key Exchange Protocols
Combining Components of Authentication Protocols
Selection of Data Sets and Addition of Requisite Data
Remove and Marge Redundant Flows and Data
Choose Detailed Parameters
Protocol Definition File

Figure 2: Automatic Protocol Generation Procedure.

requirements for the protocol. The process consists of the following five steps.

1. Combining Components of Key Exchange Protocols
   This step combines components for constructing key exchange protocols. The components include four data fields X, Y, Z_1, Z_2 as shown in Figure 3. The data fields X and Y denote transaction data in each flow and the data fields Z_1 and Z_2 indicate data that is generated by an entity and held by the entity. Data for Z_1 and Z_2 does not include data sent in the protocol. The data set for the data fields is selected in the next step. Usually, the tool uses one component for key exchange protocols.

2. Combining Components of Authentication Protocols
   This step combines components for constructing authentication protocols. The components include two data fields V, W as shown in Figure 4. The data fields V and W denote transaction data in each flow. For mutual authentication, two components are combined. Data for V, W is selected in the next step. If a protocol is required for mutual authentication, the tool combines two components: the component as Figure 4 and its reverse component that perform the role of entity A and entity B are switched.

3. Selection of Data Sets and Addition of Requisite Data
   This step inputs a data set to data fields of the protocol generated in the previous step. The data set is selected from many data sets of the key exchange protocol and authentication protocol according to requirements for the protocol and selection rules. Examples of data sets are shown in Table 2. $Rand$ and $Rand'$ indicate random numbers, C indicates a time-variant shared value such as a counter value, LK indicates a long-lived key such as a shared key between two entities and a private key or public key for each entity, and $*$ means no data, respectively. The function $F(x)$ is a one-way cryptographic function using a private key or secret key such as MAC and digital signature, $G(x)$ is a encryption function using a public key or secret key, and $H(x)$ is a key exchange function such as the Diffie-Hellman algorithm. Note that a function $I(x)$ exists and the function satisfies the condition $I(Rand, H(Rand')) = I(Rand', H(Rand))$. An explanation of the selection procedure is provided in a later section. In addition, some required data such as public key certificates and identifiers are appended to the protocol in this step.

4. Remove Redundant Flows and Data
   This step removes redundant flows and data to merge and move data. The step consists of five sub-steps as follows;
   (3-a) Check of similarity of data and merge them: The tool searches for data that have the same conditions, same sender and responder, same type data or same type function, and same creator. If two pieces of data satisfy the condition, the tool merges one piece of data with the other piece of data on a former flow.
   (3-b) Combine data: The tool searches for two functions that are of the same type, that use the same key and are calculated by the same entity, and generates a new function provided the input data of the new function is input data of both functions. Then the tool replaces the function on the former flow with the new function.
   (3-c) Move corresponding data: The tool finds data corresponding to data and functions that were removed in the previous sub-steps and moves them to the former flows. If data is sent by a flow is the same as the removed data, the data is moved to a flow that has merged data and functions or has generated a new function. If data is sent by a flow that is one flow before the flow of removed data, the data is moved to a flow that is one flow before the flow of the merged data and functions or combined functions.
   (3-d) Remove known data: Known data is data that an entity has received previously or the entity knows it at the beginning of the protocol. The tool searches for data that are known data for the

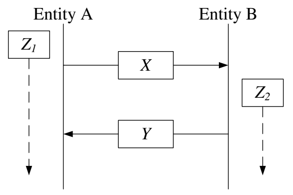

Figure 3: Component for Key Exchange Protocol.

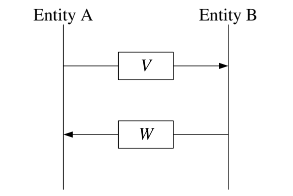

Figure 4: Component for Authentication Protocol.

receiving entity and removes the data from the flows.
(3-e) Remove flows: The tool searches flows that have no data and functions and removes them from the protocol.

5. Choose Detailed Parameters
   This step chooses detailed parameters of the protocol such as data length, key length, and cryptographic algorithm specification, according to the requirements for the protocol. Then, the tool generates a function for computing a session key.

3.4 Constructing Selection Rules

The tool selects one data set from data sets according to selection rules. The selection rules indicate the relationship between data sets and requirements. The selection rules are similar to filtering rules for a firewall, and the rules reduce the number of data set candidates according to the requirements. The data set is arranged in priority order. Thus, the tool selects the highest priority data set when two or more data sets remain under a requirement for the protocol. Examples of selection rules for Table 2 are described as follows.

For Authentication. If authentication with one flow is required, the selection rule removes the No.1 data set and the No.2 data set for authentication. If a message authentication algorithm and a digital signature algorithm are not available for entity B, the selection rule removes the No.1 data set and the No.3 data sets for authentication.

For Key Exchange. If *Forward Secrecy* is required, the selection rule removes the No.1 and No.3 data sets for key exchange and $G(X)$ in the No.2 and No.4 data sets is restricted by public key encryption using a temporal public key. If the protocol has to be secure against a *Known Key Attack*, the selection rule removes the No.1 and No.2 data sets for key exchange. If the protocol needs security against *Unknown Key Share Attack*, the selection rule removes No.1, and No.2, data sets for key exchange. If entities have no shared key, the No.1 and No.3 data sets are removed.

Data sets are also selected according to the performance requirements. For example, if one-flow key exchange and an authentication protocol are required, the tool excludes the No.1 and No.2 data sets for authentication and the No.3, No.4, and No.5 data sets for key exchange from the candidates. If entities have a shared key and have low computational power, the tool removes the No.5 data set for key exchange from the candidates.

Our tool is easily extensible to several security protocols. To add new data sets and selection rules, the tool generates protocols suitable for new security requirements that are currently unknown.

3.5 Selection of Key Generation Functions

A key generation function is related to a selected data set for key exchange. Table 3 shows the relationship between selected data sets and key generation functions. The tool generates a key generation function for each entity by using the relationship. The functions $F(x)$ and $F'(x)$ are one-way cryptographic functions using a private key or secret key such as MAC and digital signature, $G(x)$ is a encryption function using a public key or secret key, and $H(x)$ is a key exchange function such as the Diffie-Hellman algorithm. Note that a function $I(x)$ exists and the function satisfies the condition $I(Rand, H(Rand')) = I(Rand', H(Rand))$.

4 EVALUATION

We evaluated transaction time for security protocol generation on a PC (Pentium 4 2.6GHz, 2GB RAM). The evaluation results are shown in Table 4.

We evaluated three cases: generation of a mutual authentication protocol, generation of a key exchange protocol, and generation of an authenticated key exchange protocol. All transaction time is less than one second. We obtained 2999 different protocol definition files from 9828 different inputs of requirements for security protocols. The protocol definition files was categorized into 98 patterns of protocols.

Table 2: Example of Data Sets.

For Authentication				
No.	V	W	–	–
1	$Rand$	$F(Rand)$	-	-
2	$G(Rand)$	$Rand$	-	-
3	$*$	$F(C)$	-	-
For Key Exchange				
No.	X	Y	$Z1$	$Z2$
1	$*$	$Rand'$, $F(LK, Rand')$	$*$	$Rand'$
2	$*$	$Rand'$, $G(LK, Rand'), F(LK', G(LK, Rand'))$	$*$	$Rand'$
3	$Rand$	$Rand'$, $F(LK, Rand, Rand')$	$Rand$	$Rand'$
4	$Rand$	$G(LK, Rand'), F(LK', Rand, G(LK, Rand'))$	$*$	$Rand'$
5	$H(Rand), F(LK, H(Rand))$	$H(Rand'), F(LK, H(Rand), H(Rand'))$	$Rand$	$Rand'$

Table 3: Example of Key Generation Function.

Data Set	Function for Entity A $SK =$	Function for Entity B $SK =$
No.1	$F'(LK, Rand')$	$F'(LK, Rand')$
No.2	$Rand'$	$Rand'$
No.3	$F'(LK, Rand, Rand')$	$F'(LK, Rand, Rand')$
No.4	$Rand'$	$Rand'$
No.5	$I(Rand, H(Rand'))$	$I(Rand', H(Rand))$

Table 4: Evaluation Results.

Protocol	Transaction Time
Mutual Authentication	398 msec
Key Exchange	617 msec
Mutual Auth. and Key Exch.	875 msec

Transaction time required by the APG to generate mutual authentication and a key exchange protocol is 2 hours using a 400MHz CPU and the ASPB takes 20 sec to generate the protocol using a 1.8GHz CPU (Zhou and Foley, 2003). Thus, we think that our tool is fast enough to generate security protocols.

5 CONCLUSIONS

In this paper, we presented an automatic generation tool for authentication and key exchange protocols. Our tool generates a security protocol definition file from input data of requirements for the protocol. The tool constructs a new protocol to that combines a pieces of security protocols as building blocks. We also implemented the tool and evaluated the transaction time of a security protocol generation by the tool. The transaction time of the tool is less than one second. It is able to realize more rapid generation of security protocols in comparison to existing generation tools. Variations in generated security protocol depend on the number of data sets in our tool, and the number of potential candidate protocols may be reduced; thus, optimization of data sets for real services is a further topic of study for these tools.

ACKNOWLEDGEMENTS

A part of this work was supported by the National Institute of Information and Communication Technology (NICT).

REFERENCES

Abadi, M. and Gordon, D. (1999). A calculus for cryptographic protocols the spi calculus. *Inf. Comput.*, 148(1):1–70.

Bellare, M. and Rogaway, P. (1994). Entity authentication and key distribution. In *Proc. of CRYPTO '93, LNCS*, volume 773, pages 232–249. Springer Verg.

Bellare, M. and Rogaway, P. (1995). Provably secure session key distribution: thethree party case. In *Proc. of 27th Annual Symposium on the Theory of Computing*, pages pp. 57–66. ACM.

Blake-Wilson, S., Johnson, D., and Menesez, A. (1997). Key agreement protocols and their security analysis. In *Proc. of 6th IMA International Conference on Cryptography and Coding, LNCS*, volume 1355, pages pp. 30–45. Springer Verg.

Didelot, X. (2003). A compiler for security protocols. *Available at http:// web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/ Casper/COSPJ/s%ecu.pdf.*

Foley, S. N. and Zhou, H. (2003). Towards a framework for automatic security protocols. In *Proc. of Security Protocol Workshop 2003, LNCS*, volume 3364, pages 49–54. Springer Verg.

Lowe, G. (1997). Casper: A compiler for the analysis of security protocols. In *Proc. of 10th IEEE Computer Security Foundations Workshop*, pages 18–30. IEEE.

Lukell, S. and Veldman, C. (2003). Automated attack analysis and code generation in a multi-dimensional security protocol engineering framework. In *Proc. of*

Southern African Telecommunications Networks and Applications Conference 2003 (SATNAC 2003).

Perrig, A. and Song, D. (2000a). A first step towards the automatic generation of security protocols. In *Proc. of Network and Distributed System Security Symposium NDSS 2000*, pages 73–83.

Perrig, A. and Song, D. (2000b). Looking for diamonds in the desert – extending automatic protocol generation to tree-party authentication and key agreement protocols. In *Proc. of 13th IEEE Computer Security Foundations Workshop*, pages 64–76. IEEE.

Pozza, D., Sisto, R., and Durante, L. (2004). Spi2java: Automatic cryptographic protocol java code generation from spi calculus. In *Proc. of 18th International Conference on Advanced Information Networking and Application (AINA'04)*, pages 400–405. IEEE.

Song, D., Perrig, A., and Phan, D. (2001). Agvi –automatic generation, verification, and implementation of security protocols. In *Proc. of 13th Conference on Computer Aided Verification (CAV)*, pages 241–255. Springer Verg.

Zhou, H. and Foley, S. N. (2003). Fast automatic synthesis of security protocols using backward search. In *Proc. of the 2003 ACM Workshop on Formal Methods for Security Engineering*, pages 1–10. ACM.

APPENDIX

This appendix shows an example of a security protocol generated based on requirements. The requirements for the security protocol are as shown in Table 5. The horizontal rows Purpose, Security, Comp. Power, Algorithm, and Initial Key respectively indicate the purpose of the protocol, required security level and assumed adversary models, computational power of each entity, available cryptographic algorithm for each entity, and keys that the entities have before the protocol starts. The protocol generated by the tool is described as follows.

STEP1 $(A \rightarrow B) : R_1, Pub_A, M_{LKey}(R_1, Pub_A)$
STEP2 $(B \rightarrow A) : I_B, R_2, Enc_{Pub_A}(R_3)$, $M_{LKey}(I_B, R_1, Enc_{Pub_A}(R_3))$
STEP3 $(A \rightarrow B) : I_A, M_{LKey}(I_A, R_3)$

The protocol is a mutual-authenticated key exchange protocol between an entity A and entity B. The function $M_X(Y)$ is a message authentication function of Y input with a key X using the Hmac-SHA256 algorithm. The function $Enc_X(Y)$ is an encryption of Y with a key X using the RSA-OAEP algorithm. Pub_A, Pri_A are the temporal public key and corresponding private key of RSA public key encryption that is generated by entity A at the beginning of the protocol. I_A and I_B are identifiers of the entity A and entity B. R_1, R_2, and R_3 are random numbers, and *LKey* is a shared key between two entities. A session key for both entities is $Hs(R_1, R_3, LKey)$ in this protocol, where $Hs(X)$ is a hash value of X using SHA-256 algorithm. The lengths of RSA keys are 2048 bits and a length of output of Hmac-SHA1 is 256 bits. The lengths of random numbers R_1, R_2, and R_3 are 128 bits. Figure 5 shows a picture of the protocol in the tool GUI.

Table 5: Requirements for a Security Protocol.

	Requirements
Purpose	Mutual Authentication Key Exchange
Security	128-bit level security *Forward Secrecy* Secure against *Known Key Attack* *Unknown Key Share Attack*
Comp. Power	Entity A: High Entity B: Medium
Algorithm	Entity A: SHA256, Hmac-SHA256, RSA-OAEP Entity B: SHA256, Hmac-SHA256, RSA-OAEP
Initial Key	A Shared Key

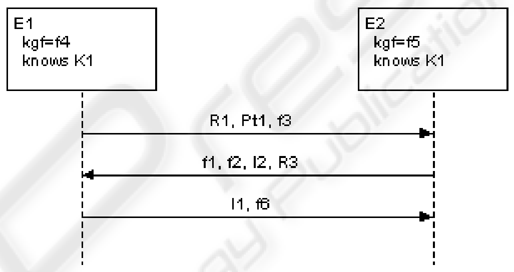

Figure 5: Picture of Generated Protocol.