A Formal Verification Using Graph-Database for Security Protocols

Authors: *Hisashi Mori*, Kazuki Yonemochi**, Manabu Misawa****

Information Technology R&D Center, ** Itami Works, * Mitsubishi Electric Digital Innovation Corporation*

Abstract

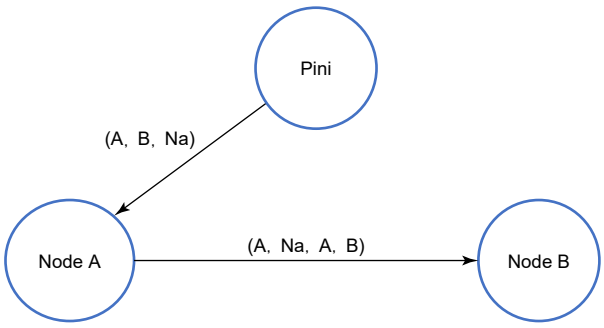
When designing a system, security verification is a method for ensuring that vulnerabilities such as information leakage and unauthorized access are not present. Conventional verification methods require advanced expertise such as formal language theory and tool-specific languages, and suffer from the issue that when systems become complex, verification does not complete within a practical amount of time. Accordingly, rather than using specialized language expressions that can be difficult to immediately understand, we developed a method that enables security verification with data insertion and searching operations by using a general-purpose graph database (hereafter, "graph DB") that provides a visual understanding of relationships among data. As a result of verifying a specific security protocol, intuitive verification with graph visualization detected vulnerabilities similar to those found using conventional techniques. Furthermore, we theoretically demonstrated that verification time can be reduced.

1. Introduction

There is growing demand for ensuring system safety and security. To address such demand, methods are needed that guarantee at the design stage that systems are safe and secure. Security protocols are an example of components that constitute a system, but it is difficult to manually analyze whether a given protocol operates correctly against attacks. To address this problem, automating analysis using verification methods based on formal methods is one effective approach. In verification of security protocols, the protocol and the intruder's behavior with respect to the protocol are modeled in a formal language, the security requirements that the protocol must satisfy are modeled as verification properties in another formal language, and verification is executed using a tool. Such tools include dedicated security protocol verification tools and general-purpose model checkers. Among these, SPIN[1] and other general-purpose model checkers can describe various verification properties, including security, with multiple case studies of security verification existing. Thus, this study focuses on security verification using general-purpose model checkers. A variety of security protocols have been verified with SPIN, and the usefulness of security verification using SPIN has been demonstrated. For example, Maggi et al. conducted verification on the NSPK (Needham-Schroeder Public Key) protocol and derived vulnerabilities[2]. Furthermore, for security protocols in general, Henda logically defined an intruder who performs eavesdropping and insertion as well as retransmission of eavesdropped messages. Based on this definition, verification was conducted on three

```
mtype= {A, B, Na} ;
Chan ca= [0] of {mtype, mtype, mtype, mtype} ;
proctype Plni (mtype self; mtype party; mtype nonce)
{
        mtype g1 ;
        ca ! self, nonce, self, party;
}
proctype PRes (mtype self; mtype nonce)
{
mtype g2, g3;
        ca ? eval (self), g2, g3, eval(self);
}
init
{
        run Plni(A, B, Na)
}
```

(a) Example of a program for SPIN



(b) Example of a graph using a graph-databases

Fig. 1 Examples of describing a target system by the existing method SPIN and the proposed method

types of security protocols, including the NSPK protocol, and known vulnerabilities were derived[3]. However, these conventional techniques require learning specialized languages, making adoption relatively difficult. Furthermore, as the target system increases in complexity, the number of internal states representing the protocol and the intruder's behavior grows exponentially, and verification cannot be completed within a practical timeframe.

In this study, we adopted Neo4j[4]*1, one of the most widely used graph DBs and implemented model checking using a graph DB. Figure 1 shows an example description of the target system using the conventional technique SPIN, and a conceptual diagram representing the same target system as a graph. Figure 1 shows sender A (in the figure, $Node\ A$) sending to receiver B (in the figure, $Node\ B$), the message Na and A, encrypted with receiver B's public key and sent (in the figure, $Node\ A$ to $Node\ B$ arrow) representing this communication. With conventional techniques, after converting the behavior of the target system into a state transition model, that state transition model is programmed in text (Fig. 1 (a)). Whether the program correctly represents the target system needs to be deciphered during verification. Meanwhile, with the graph visualization in the proposed method, the behavior of the system can be intuitively and visually understood (Fig. 1 (b)). Here, $Pini$ denotes the start of the protocol. Also, the characters on the arrow indicate, from left, "sender A, message Na and A, encrypted with receiver B's public key." The burden of conducting security verification can be reduced by being able to understand the target system in an intuitive visual manner.

2. Proposed Method: Security Verification Method Using a Graph Database

We propose a new security verification method that enables intuitive verification and reduces verification time. Section 2.1 describes how, in the proposed method, we define the target system, attack model, and verification properties, which are the input information for the model checking used for this study. Section 2.2 outines the implementation method using a graph DB and the results of actually performing verification.

2.1 Definition of the formal expression of the proposed method

In the proposed approach, we express the security protocol to be verified and the intruder's behavior as a directed graph, and express the properties to be verified as graph DB queries, thereby enabling intuitive verification. In this section, we explain how to define the target system, attack model, and verification properties on the graph DB, and what the verification results indicate for those inputs.

Target system The target system in the proposed approach is a directed graph equivalent to the state transition model of existing model checkers. We express the target system as a directed graph (V, E). Here, V is the set of vertices, E is the set of directed edges. We define the vertices and directed edges as follows.

Vertices are defined to include r (a field indicating the sender/receiver) and s (a field indicating which state in the protocol) as labels. We express the reception of a protocol-conformant message by sender (or receiver) as a single vertex. If the message content differs, each is expressed by a distinct vertex. In addition to those, we add two types of vertices, $Init$ representing protocol start and End representing protocol end.

Directed edges are defined to include t (a field indicating a cipher text or a plain text), m (a field indicating the transmitted message content), k (a field indicating the key that encrypts the message). We regard the communication from the protocol sender to the receiver as a single directed edge.

Attack model The Dolev–Yao model is a well-known attack model that can exhaustively examine vulnerabilities in security protocols, and it is also adopted in Henda's study described in Section 1[3]. Similarly, the proposed approach uses Henda's attack model based on the Dolev–Yao model. That is, we verify whether the security properties are satisfied on the target system augmented with Henda's attack model. For reasons of space, please refer to the original paper[3] for the rigorous definition of Henda's attack model.

We show the procedure for adding the attack model to the target system. First, as preparation, the intruder's set of known information $\mathbb{K}$ is partitioned into the subsets $ACTOR$ (sender/receiver/intruder), $PLAIN$ (plain text), $CIPHER$ (cipher text), $PKEY$ (public key), $SKEY$ (secret key). The procedure for creating the vertices and directed edges that represent attacks when a particular state is used as the source is as follows.

*1 Neo4j is a registered trademark of Neo4j, Inc.

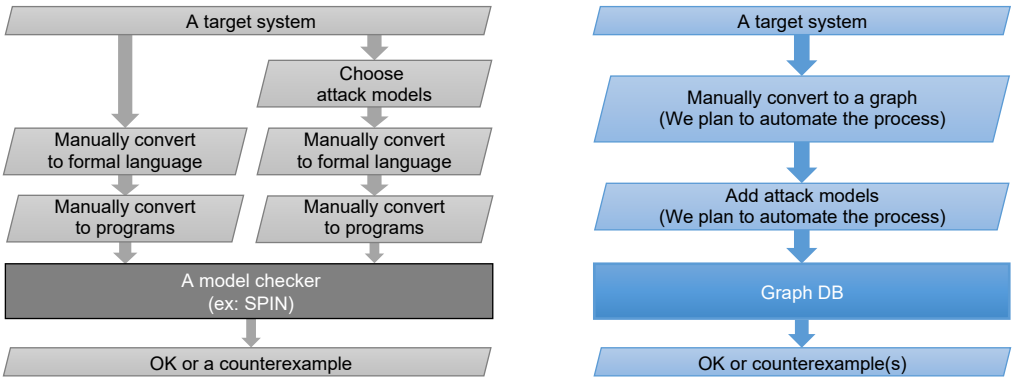
(1) *ACTOR*, *PLAIN*, *PKEY*: select an element from each of them and create cipher texts for all combinations
(2) If the destination vertex (sender or receiver) to send to does not exist, create the destination vertex
(3) Use each cipher text created or the cipher texts already stored in *CIPHER* as labels, and create a directed edge from the source to the destination

Verification property In the proposed approach, as with the conventional technique SPIN, we assume LTL (Linear Temporal Logic) as the formal language for describing verification properties. As in the conventional technique[5], we describe LTL-equivalent formulas in the graph DB. As an example of a verification property, we address user authentication. According to prior work[2] [3], the definition of user authentication in model checking (that sender A indeed authenticates receiver B) is: "Whenever sender A has completed execution of the protocol, the other party B is executing the protocol."

Verification results On the directed graph representing the target system and the attack model, execute the query that represents the verification property; if a path that does not satisfy the query is output, that verification property is not satisfied. As noted earlier, the directed graph and queries in the proposed approach are respectively equivalent to the state transition model and verification properties in conventional techniques (model checkers). Therefore, the output of the queries is equivalent to the verification results of conventional techniques.

2.2 Workflow of model checking using a graph DB

This section describes the workflow of model checking using a graph DB, the proposed method (Fig. 2).



(a) Workflow with the existing methods (b) Workflow with the proposed method

Fig. 2 Workflows for security verification with the existing and the proposed methods

In the conventional technique (Fig. 2 (a)), after selecting the system specifications and attack model to be verified, it was necessary to manually convert them into a formal language, then convert them into the model checker's program, and only then execute verification on the model checker. On the other hand, with the proposed method (Fig. 2 (b)), the system specifications and attack model to be verified are selected, and the verification simply executed to obtain the results. The proposed method is an algorithm that converts the protocol and attack model into a graph DB for any security protocol. Once the algorithm is established, automating the conversion is readily possible. Going forward, we plan to automate the conversion of protocols and attack models into the graph DB. Using a graph DB in the model checking method enables designers to perform model checking with knowledge of ordinary database operations—adding vertices and directed edges, and performing path searches. Also, by using a graph DB and appropriately reducing the number of states of the intruder, the variables representing information sent and received in the protocol, and forged messages, verification using the proposed method can reduce the number of states required for verification compared with the conventional techniques.

As an example demonstrating gragh DB being implemented, we targeted the NSPK protocol mentioned in Chapter 1 and verified authentication, the verification property described in Section 2.1. Specifically, we expressed the NSPK protocol, the attack model, and the logical formula expressing authentication on the graph DB following the procedure shown in Section 2.1, and executed both a method to find the shortest path among the paths that do not satisfy authentication and a method to find all paths. Here, when the verification target and the verification property is input and the verification is run, if the verification target

does not satisfy that verification property, paths in the verification target that do not satisfy the verification property are searched and output. Conversely, if the verification target satisfies the verification property, nothing is output. As a result of running the verificatoin, counterexamples were output by both methods. By visually confirming that the two types of paths do not satisfy the verification property, we verified that the results are correct. Note that, due to space limitations, the graphs expressing the NSPK protocol and the verification results are omitted.

3. Assessment

Both the verification method using SPIN as the conventional technique and the verification method using a graph DB in the proposed method create a model of the behavior of the target system as a state transition model, and output a counterexample when a state that does not satisfy the verification property is reached. That is, the verification time depends on the number of states generated. If there are too many states, there is a risk that the verification will not complete in a practical amount of time. Therefore, in this chapter, as a logical evaluation of execution time, we compare the number of states generated by the conventional technique and by the proposed method. As a result of comparing the number of states, we confirmed that the proposed method always yields fewer states than the conventional technique. For example, for the NSPK protocol (number of messages m = 3), the number of states in the conventional technique is 15,275, while the proposed method has 20 states, which is about 1/763 (Fig. 3). The number of states generated by the conventional technique SPIN is the product of the number of states of the target system and the number of states for the verification property. By contrast, the number of states in the proposed method is just the number of states of the target system. This is because, thanks to the graph DB's search functionality, verification can be performed by searching only states of the target system. The results of calculating and comparing the number of states for the conventional technique and the proposed method, respectively are showing as follows.

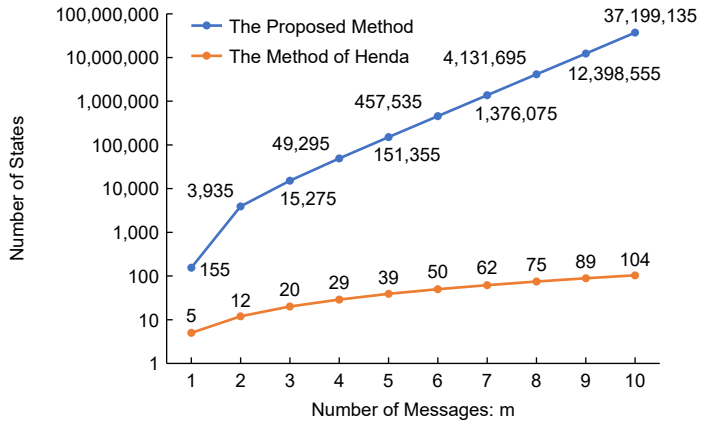


Fig. 3 Comparison of the number of states for the existing and the proposed methods

In Henda's conventional approach, the number of states is given by the following equation.

$$\{m+3+\sum_{i=0}^{m}\{(|Knows|-|PKEY|)^{(t-t_{|PKEY|}+(i-1))} \times\{(|Knows|-|PKEY|)^{(t-t_{PKEY})}\times|PKEY|+1\}\}\} \times(2^{|TL|}+1)$$

Here, m is the number of messages; $|Knows|$ is the number of elements of the information known by the intruder; $|PKEY|$ is the number of elements among the intruder's known information that are public keys; t is the number of terms per message; $t_{|PKEY|}$ is the number of terms per message that represent public keys; $|TL|$ is, if we define TL as the set of formulas in which the outermost operator of each subformula of the verification property is a modal operator, the number of elements of TL.

Meanwhile, in the proposed approach, the number of states is given by the following formula.

$$(m+3)+|Init|+\sum_{i=0}^{m}((|PLAIN|^{t_{PLAIN}}+i+1))$$

Here, | $Init$ | is the number of states of the initial sender; | $PLAIN$ | is the number of elements in the entire plain text; t_{PLAIN} is the number of terms per message that express plain text.

Based on the above formula, for Henda's method and the proposed approach, a graph comparing the change in the number of states with respect to the number of messages m is shown in Fig. 3. Overall, the proposed approach results in a smaller number of states. On a typical PC, the verification limit is said to be around 500,000 states[6]. With the conventional approach, when the number of messages is 7, the number of states exceeds 500,000, whereas with the proposed approach it is kept at 62.

4. Conclusion

Using a security protocol as an example, we proposed a model checking method that describes the target system as a directed graph in a graph DB and specifies the verification property as a query in the graph DB. Using general graph DB knowledge, one can create graphs, and visualization functions allow for an intuitive understanding. In addition, by using the graph DB's all-path search function, it is also possible to execute model checking that outputs all counterexamples. Furthermore, by directly specifying the verification property as a query over the directed graph and searching, the number of states during model checking can be reduced, making automated verification feasible within a practical time.

Going forward, when this is applied to other security protocols, we plan to evaluate how much the execution results and the number of states differ between the conventional technique SPIN and the proposed approach. Furthermore, we plan to show that the proposed approach can also define and verify properties other than authentication. We also plan to extend the formalization of the target system and verification property so that verification including system safety can be performed. Going forward, systems will become increasingly complex, making it difficult to manually determine whether security functions are correctly implemented. By using the proposed approach to automatically assure system security, the workload in the design phase can be significantly reduced.

References

(1) Holzmann, G.J.: The SPIN Model Checker, Addison-Wesley (2004)
(2) Maggi, P., et al.: Using SPIN to Verify Security Properties of Cryptographic Protocols, Proceedings of the 9th International SPIN Workshops on Model Checking on Software, LNCS 2318, 187–204 (2002)
(3) Henda, B,N.: Generic and Efficient Attacker Models in SPIN, Proceedings of the 2014 International SPIN Symposium on Model Checking of Software, 77–86 (2014)
(4) Robinson, I., et al.: Graph Databases, 2nd Edition, O'Reilly Media (2015)
(5) Kuno, K., et al.: A Model Checking Approach Using Graph Database, Information Processing Society of Japan Technical Report (SE), 2018-SE-198 (2018)
(6) Jøsang, A.: Security Protocol Verification using SPIN, the first SPIN Workshop (SPIN'95), 1–9 (1995)