Human Competitive Security Protocols Synthesis

Hao Chen
University of York
York, UK
chenhao@cs.york.ac.uk

John Clark
University of York
York, UK
jac@cs.york.ac.uk

Jeremy Jacob
University of York
York, UK
jeremy.jacob@cs.york.ac.uk

ABSTRACT

This poster paper outlines a method for a search based approach to the development of provably correct protocols.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols; C.2.0 [**Computer Communication Networks**]: Security and Protection; D.2.2 [**Software Engineering**]: Design Tools and Techniques

Keywords

Security Protocols, Belief Logic, Protocol Synthesis, Heuristic Search, Efficiency Requirements, Protocol Refinement

1. SECURITY PROTOCOLS AND BELIEF LOGICS

The design of secure communication protocols is a very difficult task. It is almost invariably carried out manually. There is little work generating designs automatically. In this paper we shall show how an optimisation based technique can be used successfully to generate abstract security protocols satisfying identified goals. We view the design synthesis as a numerical optimisation problem and use the established global optimisation technique of simulated annealing to perform the search. Our search operates over a space of abstract protocols (described in SvO belief logic [2]). We have developed further procedures to compile concrete implementation level protocols from those abstractions, and also to remove any redundancies. We then subject these concrete protocols to analysis by another technique (CASPER) to ensure that our refinement process has not introduced flaws.

Our synthesis framework makes use of a widely researched approach in protocol security, namely the use of belief logics to model and reason about protocol security. One could view protocol messages as programming language statements (with the belief state of the receiver being updated according to the rules of the logic). We search over the space of candidate abstract protocols, simulating them to evaluate what they actually achieve. A measure of how well the actual achievements match the required goals provides guidance to this search. A highly unusual feature of the approach is that the simulation (execution) of the abstract protocol corresponds to a proof that the protocol actually achieves what it does. When we finally find a protocol whose achievements match those required, the execution of that protocol is a *proof* that the protocol achieves those results. Thus we have evolved a *provably correct protocol.*

Below is a set of initial assumptions that are held by principal A and a key distribution server S, goals and a feasible SvO protocol.

Initial Assumptions:

$$\begin{array}{lll}
A \mid\equiv A \ni K_{as} & A \mid\equiv A \xleftrightarrow{K_{as}} S & A \mid\equiv A \ni N_a \\
A \mid\equiv \#(N_a) & A \mid\equiv (S \mid\Rightarrow A \xleftrightarrow{K_{ab}} B) & A \mid\equiv (S \mid\Rightarrow \#(K_{ab})) \\
S \mid\equiv S \ni K_{as} & S \mid\equiv A \xleftrightarrow{K_{as}} S & S \mid\equiv S \ni K_{ab} \\
S \mid\equiv A \xleftrightarrow{K_{ab}} B & S \mid\equiv \#(K_{ab}) &
\end{array}$$

Goals:

$$A \mid\equiv A \ni K_{ab} \qquad A \mid\equiv A \xleftrightarrow{K_{ab}} B \qquad A \mid\equiv \#(K_{ab})$$

A Feasible SvO Protocol:

$$\begin{array}{ll}
1. & A \rightarrow S : A \mid\equiv A \ni N_a, A \mid\equiv (S \mid\Rightarrow A \xleftrightarrow{K_{ab}} B) \\
2. & S \rightarrow A : \left\{ \begin{array}{l} A \mid\sim A \ni N_a, \\ S \mid\equiv S \ni K_{ab}, \\ S \mid\equiv \#(K_{ab}), \\ S \mid\equiv A \xleftrightarrow{K_{ab}} B \end{array} \right\}_{K_{as}}
\end{array}$$

In this example, a principal A obtains a new secure session key K_{ab} to communicate with another principal B from a key distribution server S. We start from a set of initial assumptions and intend to achieve the above goals. Firstly, A possesses and believes a long term key K_{as} is secure for communicating with S. He also possesses and believes a particular random number N_a (a nonce) is well formed and fresh. A also trusts S to provide K_{ab}, that is to say A believes that S controls the fresh session key. Secondly, S possesses and believes the long term key K_{as} is secure as well as A. As S plays the role of a key distribution server, he also possesses and believes the new session key K_{ab} is fresh and well formed which is secure for A and B to communicate with each other. The goals of this protocol are for A to obtain and believe K_{ab} is fresh and secure for communicating with B. So this is the requirement for a fragment of a key distribution protocol.

A is able to send out whatever he believes. In this example, A believes he possesses N_a, which is a well formed nonce and may include it in the first message together with A believes S controls K_{ab}. When the server S receives this message, he could not deduce any new beliefs since the message is in plaintext. However, he knows that A at some time sent a message that contained N_a and A requested a session key for communicating with B. Now, S may reply to A with the second message that contains four components: A said he possessed N_a and S possesses and believes K_{ab} is a fresh good key for A to communicating with B. S encrypts this message using key K_{as}. Once A receives this message, he may decrypt it to reveal its contents. Then, A believes that he received $A \mid\!\sim A \ni \#(N_a)$, $S \models S \ni K_{ab}$, $S \models \#(K_{ab})$ and $S \models A \xleftrightarrow{K_{ab}} B$. Next, by applying the SvO logic rules, A concludes S said $A \mid\!\sim \#(N_a)$, $S \models S \ni K_{ab}$, $S \models \#(K_{ab})$ and $S \models A \xleftrightarrow{K_{ab}} B$. This message contains an assertion involving N_a, a nonce A believes to be fresh, so A may conclude the whole message is a fresh one. Then A may deduce that S recently says the whole message. By applying the logic, A believes he possesses K_{ab}. Since A believes that S controls the fresh session key, A may now believe $\#(K_{ab}), A \xleftrightarrow{K_{ab}} B$ using the logic.

2. SEARCH BASED SYNTHESIS

Our tools use a standard simulated annealing algorithm. Below we give details of representation and fitness evaluation.

2.1 Protocol Representation

In our protocol synthesis system, a protocol is represented as a sequence of M messages, each of which is represented by an integer sequence. N principals, indexed $0 \ldots N-1$, participate in the protocol. Associated with each of the principals is a vector of its current beliefs. Each of the M messages is represented by $B+3$ integers, $s, r, k, b_1, \ldots, b_B$. These represent the sender, the receiver, the key that the sender used to encrypt this message, and a series of B indices that reference beliefs currently possessed by the sending principal. So, the sender is $s \bmod N$; the receiver is $r \bmod N$; the key is $k \bmod NK$ (where NK is the number of keys possessed by the sender); and the first component in the message is $b_1 \bmod T$ etc., where the sender possesses T beliefs, indexed $0 \ldots T-1$. The vectors of the receiver's current beliefs is updated after each message is sent (see below). In this way, an arbitrary sequence of integers can be interpreted as a feasible protocol (senders only ever send beliefs they actually hold).

This also allows a very simple move function for local search — simply randomly replace some of the integers involved in some messages.

2.2 Executing a Protocol

Assume that a protocol consists of M messages, each of which consists of B beliefs, and we start from the very beginning of this protocol. Firstly, we initialise the belief state of the relevant principals involved in this protocol. Then, for each message in this protocol, we firstly determine the sender, recipient, and the key under which the current message is encrypted. Secondly, we decode each of the B beliefs corresponding to the current message. For instance, the first belief in the message is $b_1 \bmod T$, where the sender currently holds T sendable beliefs. Thirdly, we update the receiver's beliefs vector by applying the SvO axioms. Finally, we record the number of required goals achieved after this message has been analysed. Once a protocol has been executed in the above way, the fitness of this protocol can be calculated as given in Section 2.3.

2.3 The Fitness Function

We use fitness functions, which are sums of a security fitness function and an efficiency fitness function. The security fitness function measures how close it comes to achieving the required security goals and our search seeks to find a protocol that satisfies all these goals. The efficiency fitness function will punish protocols with many messages, more encryption and more numbers of interactions with particular principals.

3. EXPERIMENTAL METHOD & RESULTS

This section reports the results of applying the technique described above to the synthesis of security protocols. The Security Protocols Open Repository (SPORE) at http://www.lsv.ens-cachan.fr/spore/ is an on-line security protocols library. The latest SPORE contains 49 protocols and covers most of the security protocols that appear in the literature. We have used SvO logic to specify 40 out of the 49 protocols. There are several "variations on a theme" in the library. Often, the assumptions and goals of these variants are the same. Thus, there may be one abstract specification and several concrete implementations. The 40 evolved concrete correspond to 25 distinct SvO specifications. Our system succeeded on 23 of these specifications, and failed only on 2. All these 23 protocols are correct at the logic level according to the SvO logic, and the successful synthesis process itself is a proof. In fact, for each of the 4 different efficiency criteria in each of these 23 cases we find a suitable protocol.

For the remaining 9 protocols in the SPORE, 5 of them have features outside of the SvO logic, and so our tool cannot be used. The other 4 were added into the library very recently, and their SvO specifications are currently under investigation.

Our framework's capabilities would seem to exceed those of the only other protocol synthesis technique we know about (the model checking approach of Perrig and Song [1]). Our framework can handle protocols of significantly greater size, for example, we have evolved protocols with up to 9 messages. Model checking approaches would seem limited to protocols with three or four messages. In addition, a secure protocol is generated in only a few minutes. The extensive experimentation shows that our approach is able to produce protocols satisfying the requirements of a great many protocols published over the past 30 years. We believe that the experimental results validate our claim to be human competitive.

4. REFERENCES

[1] A. Perrig and D. Song. Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 64–76. IEEE Computer Society Press, 2000.

[2] P. F. Syverson and P. C. van Oorschot. A unified cryptographic protocol logic. Technical Report NRL Publication 5540–227, Naval Research Lab, 1996.