Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 125 (2005) 25–41
www.elsevier.com/locate/entcs

Synthesising Efficient and Effective Security Protocols

Chen Hao[1] John A. Clark[2] Jeremy L. Jacob[3]

Department of Computer Science
University of York
YORK, YO10 5DD, United Kingdom

Abstract

The logical correctness of security protocols is important. So are efficiency and cost. This paper shows that meta-heuristic search techniques can be used to synthesise protocols that are both provably correct and satisfy various non-functional efficiency criteria. Our work uses a subset of the SVO logic, which we view as a specification language and proof system and also as a "protocol programming language". Our system starts from a set of initial security assumptions, carries out meta-heuristic search in the design space, and ends with a protocol (described at the logic level) that satisfies desired goals.

Keywords: Security Protocols, Belief Logic, Automated Protocol Synthesis, Meta-heuristic Search, Non-functional Requirements.

1 Introduction

The correctness of a security protocol is a crucial design goal, but other, non-functional, criteria (most typically efficiency criteria) are also of considerable importance. The definition of "efficiency" will vary according to circumstance. Furthermore, designers often wish to find an efficient way of implementing a specification and they may also want to explore the consequences of making different initial assumptions. This paper demonstrates a framework for the automated synthesis of provably secure *and* efficient security protocols. Our

[1] E-mail: chenhao@cs.york.ac.uk
[2] E-mail: jac@cs.york.ac.uk
[3] E-mail: jeremy.jacob@cs.york.ac.uk

doi:10.1016/j.entcs.2004.05.017

framework views design synthesis as a numerical optimisation problem. We use the established global optimisation technique of simulated annealing [10] (see subsection 4.1 for a description) to perform the search. The framework extends previous work of the authors [5,6,7] which addressed only issues of logical correctness. Our previous work used BAN logic. In this paper, we use SVO logic [13], a more sophisticated logic than BAN logic, and thus giving greater confidence in the practical security of discovered protocols.

2 Protocols and Belief Logics

Since the late 1980's, formal methods have been used in the field of security protocols [4]. Recent approaches to the use of formal methods in the design of security protocols include finite-state model checking and belief logics [12]. In this paper, we will concentrate on protocol security belief logics, which formalise what a principal may infer from messages received.

Belief logics have played a crucial role in the development of protocol analysis as a research topic. The earliest, and best known, is the logic due to Burrows, Abadi and Needham [3], generally referred to as BAN logic. In BAN logic, the assumptions and goals of the protocols are specified as beliefs of principals. A message comprises a set of beliefs held by its sender. The logic provides various inference rules that define how a recipient's belief state may legitimately evolve upon receipt of a message. Informally, a message of a sender's beliefs can be received, but it may be encrypted. If a principal has the key, then the plaintext may be obtained. However, the principal has no assurance where the plaintext comes from, unless the message is signed. If a message is signed, then a principal may assume that the signing principal once said the plaintext. But he does not know when the message was said, unless it is fresh, in which case, the principal may assume the signer effectively, says the message at the same time it is received (that is, the message is timely, and thus not a replay). Still, the message may not come from a principal trusted to have the authority to make such statements. If it does, then whatever the message asserts is considered accurate.

BAN logic is simple and elegant and allows for very short abstract proofs. However, it does not allow analysis at the level of sophistication required by some analysts. Accordingly, various other, related, logics have been developed to fill some of the perceived gaps. This has generally been at the expense of increased complexity (for example, GNY logic [9] has more than 50 inference rules). Perhaps the most significant of subsequent logics has been SVO which aims to efficiently unify previous logics (BAN [3], GNY [9], AT [1] and VO [14]). SVO does not simply add on new notation and rules to capture

the additional scope of these logics. Rather, its aim is to produce a model of computation and a logic that is sound with respect to that model, whilst still keeping the expressiveness of the various BAN extensions. We shall use a subset of SVO logic as the basis for our protocol synthesis framework. The parts of SVO we need for the examples in this paper are given in subsection 2.1.

2.1 SVO Notation

The language of SVO logic consists of the following expressions:

- P believes X: P may act as if X is true.
- P has X: X is a message that P can see. This includes messages initially available to P, received by P, freshly generated by P, and constructible by P from the above.
- P received X: P has received a message that contains X, and P can retrieve X from the message; this may require decryption.
- P said X: The principal P at some time sent a message including the statement X.
- P says X: X is a statement P said very recently. That is, P must have said X since the beginning of the current epoch.
- P controls X: P has jurisdiction over X. The principal P is an authority on X and should be trusted on this matter. An example of jurisdiction is that principals may believe that a key distribution server has jurisdiction over statements about the quality of keys.
- fresh(X): X has not been sent in a message at any time before the current run of the protocol. This is usually true for nonces.
- $P \overset{k}{\longleftrightarrow} Q$: k will never be discovered by any principal but P, Q, or a principal which is trusted by P or Q.
- $\{X\}_k$: This is the notation for encryption of X by key k. It is assumed that encrypted messages are uniquely readable and verifiable by holders of the right keys. Similarly, encrypted messages can only be created by a principal with the appropriate keys.

2.2 SVO Axioms

When a principal receives a message, the logic provides twenty two axioms that indicate what new beliefs this principal may infer from the message contents. The axioms we we need for the examples in this paper are given below.

Belief Axiom

1. $(P \text{ believes } \varphi \wedge P \text{ believes } (\varphi \Rightarrow \psi)) \Rightarrow P \text{ believes } \psi$.

Source Association Axiom

2. $(P \overset{k}{\longleftrightarrow} Q \wedge R \text{ received } \{X^Q\}_k) \Rightarrow (Q \text{ said } X \wedge Q \text{ has } k)$.

Keys are used to deduce the identity of the sender of a message. The superscript Q in the axiom indicates that the message is from Q (rather than P). The axiom applies when any principal R receives $\{X^Q\}_k$.

Receiving Axioms

3. $P \text{ received } (X_1, \ldots, X_n) \Rightarrow P \text{ received } X_i$, for each $i \in \{1, \ldots, n\}$.

If a principal received a concatenation of messages, then the principal has received each component.

4. $(P \text{ received } \{X\}_k \wedge P \text{ has } k) \Rightarrow P \text{ received } X$.

If a principal received an encrypted message and has the key, then the principal also received the decrypted message.

Possession Axioms

5. $P \text{ received } X \Rightarrow P \text{ has } X$

A principal possesses anything he received.

6. $P \text{ has } (X_1, \ldots, X_n) \Rightarrow P \text{ has } X_i$, for each $i \in \{1, \ldots, n\}$.

If a principal possesses a concatenation of messages, then the principal possesses each component.

Saying Axioms

7. $P \text{ said } (X_1, \ldots, X_n) \Rightarrow (P \text{ said } X_i \wedge P \text{ has } X_i)$, for each $i \in \{1, \ldots, n\}$.

8. $P \text{ says } (X_1, \ldots, X_n) \Rightarrow (P \text{ said } (X_1, \ldots, X_n) \wedge P \text{ says } X_i)$, for each $i \in \{1, \ldots, n\}$.

Freshness Axiom

9. $\text{fresh}(X_i) \Rightarrow \text{fresh}(X_1, \ldots, X_n)$, for any $i \in \{1, \ldots, n\}$.

A concatenated message is fresh if one of its components is fresh.

Jurisdiction Axiom

10. $(P \text{ controls } \varphi \wedge P \text{ says } \varphi) \Rightarrow \varphi$.

Nonce-Verification Axiom

11. $(\text{fresh}(X) \wedge P \text{ said } X) \Rightarrow P \text{ says } X$.

Freshness promotes a message from having been said (sometime) to having been said during the current epoch.

Symmetric Goodness Axiom

12. $P \overset{k}{\longleftrightarrow} Q \equiv Q \overset{k}{\longleftrightarrow} P$.

2.3 GNY Recognisability Rule and Message Extension

SVO logic represents a protocol at the abstract level by asserting comprehension of received messages and interpretation of comprehended messages. These premises almost preclude automated reasoning in that a message of the same format may carry different, context dependant, meaning in different protocols. We avoid this limit by using the GNY *Recognisability Rule* and *Message Extension* [9].

2.3.1 GNY Recognisability Rule

A principal P would recognise X if P has certain expectations about the contents of X before he actually receives X. P may recognise a particular value (for example, his own identifier or nonce), a particular structure (for example, the format of a timestamp), or a particular form of redundancy. The GNY recognisability rule specifies the formulæ that a principal can believe to be recognisable, when given his beliefs about the recognisability of other formulæ.

$$P \text{ believes } \phi(X_i) \Rightarrow P \text{ believes } \phi(X_1, \ldots, X_n), \text{ for any } i \in \{1, \ldots, n\}.$$

If a principal P believes that a formula X is recognisable, then he is entitled to believe that the whole message is recognisable. In this paper, we use this rule as a replacement of the SVO comprehension assertion.

2.3.2 GNY Message Extension

Typical protocol specifications often include verbal description to the effect that a principal should proceed only if certain conditions hold or only if he holds certain beliefs. This can be regarded as a precondition of a message. The precondition of a formula X, represented by statement C, is described as $X \hookrightarrow (C)$, where C is called a *message extension*. When a receiver receives a message that contains $X \hookrightarrow (C)$, he should interpret it as such. We use the notion of *message extension* to replace the SVO interpretation assertion.

Initial Assumptions

A has (A, B, S, N_a, k_{as}) $\quad$ S has $(A, B, S, k_{as}, k_{ab})$

A believes $A \overset{k_{as}}{\longleftrightarrow} S$ $\quad$ A believes fresh(N_a) $\quad$ A believes $\phi(N_a)$

S believes $A \overset{k_{as}}{\longleftrightarrow} S$ $\quad$ S believes $A \overset{k_{ab}}{\longleftrightarrow} B$

A believes $(S$ controls $A \overset{k_{ab}}{\longleftrightarrow} B)$

Goals

A has k_{ab} $\quad$ A believes $A \overset{k_{ab}}{\longleftrightarrow} B$

A Feasible Protocol

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{as}}$

Fig. 1. Initial assumptions, goals and a feasible protocol

2.4 Illustrative Example

Figure 1 gives a set of initial assumptions that are held by principal A and a key distribution server S, and a feasible protocol.

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 their identifiers (that is, A, B and S). He also possesses a particular random number N_a (also known as a *nonce*), and a long term key k_{as} for communicating with S. Secondly, S possesses principals identifiers (A, B and S) and the long term key k_{as} as well as A. As S plays the role of a key distribution server, he also possesses the new session key k_{ab}. Thirdly, A believes that k_{as} is secure and N_a is a well formed nonce which is both fresh and recognisable. A also trusts S to provide k_{ab}, that is to say A believes that S has jurisdiction over the new session key. And finally, S believes that k_{as} is secure for communicating with A and k_{ab} is a good key that can be used between A and B.

The goals of this protocol are for A to obtain and believe k_{ab} is secure for communicating with B, that is, A has k_{ab} and A believes $A \overset{k_{ab}}{\longleftrightarrow} B$. So the protocol is a fragment of some key distribution protocol.

A believes N_a is a well formed nonce and may include it in the first message together with identifiers A and B. When the server S receives this message, he can possess the nonce N_a later. However, S could not infer anything from this message since the message is in plaintext. Now, S may reply to A with

the second message that contains two components: the newly received N_a and $k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)$. S encrypts this message using key k_{as}. Once A receives this message, he may decrypt it to reveal its contents. Then, by the *GNY Recognisability Rule*, the whole message is recognisable by A because N_a is recognisable. That is, A believes that he received $(N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B))$. So, A has k_{ab} is achieved by applying the SVO possession axioms. Then, by *GNY Message Extension*, A believes that he received $(N_a, A \overset{k_{ab}}{\longleftrightarrow} B)$. Next, using the *Source Association Axiom*, A concludes that S said $(N_a, A \overset{k_{ab}}{\longleftrightarrow} B)$. 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 says the whole message using the *Nonce-Verification Axiom*. In detail, A believes S says $(N_a, A \overset{k_{ab}}{\longleftrightarrow} B)$. Since A believes that S has jurisdiction over the new session key, A may now believe $A \overset{k_{ab}}{\longleftrightarrow} B$ using the *Jurisdiction Axiom*. Note that this protocol is at an abstract logic level and it is secure with respect to SVO logic at that level.

A possible concrete version of this protocol is:

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, k_{ab}\}_{k_{as}}$

3 Efficiency of Security Protocols

Current research in security protocols has largely focused on the *security* of protocols, and there is very little published discussion on the issue of protocol *efficiency* (notable exceptions are Boyd & Mathuria [2] and Gong [8]). The treatment of efficiency or performance is generally given a low priority and is often rather ad hoc. One possible reason is that security protocols normally involve only a few messages, thus optimisation is not seen as a very urgent requirement. However, as Gong [8] points out, it is natural and beneficial to investigate whether a protocol that achieves security requirements in a particular environment is also in some sense minimal or optimal. For example, reducing one message from a five-message protocol represents a 20% reduction in the number of messages and possibly a similar amount of reduction of the overall running time of the protocol.

Boyd and Mathuria [2] define two sorts of efficiency: computational efficiency and communications efficiency. Computational efficiency is concerned with the computations that the principals need to engage in to complete the protocol. This will largely depend on the algorithms used to provide the cryptographic services. Communications efficiency is concerned with the number

and length of messages that need to be sent and received during the running of a protocol. In some sense, computational efficiency is an issue that can be discussed at the concrete level whilst communications efficiency can be expressed at the abstract level. In our system, we define a collection of fitness functions for expressing efficiency issues (number of messages, encryption method, server interaction etc.) and ways of determining how these concerns have been achieved. Details of the fitness function are given in subsection 4.4.

4 Search Strategy

Given a specification (assumptions as precondition, goals as postcondition) we wish to search the space of protocols for those satisfying that specification. Any series of honest exchanges between two or more principals defines a feasible (with respect to the logic) protocol. We consider this set of protocols as the design space. It is clear that this space grows exponentially as the number of messages or the number of principals rise. The choices of the content of messages introduces further combinatorial complexity. For a synthesis technique to be scalable, it cannot be based on simple enumeration. Our previous work has shown that the protocol synthesis problem can be couched as a numerical optimisation problem [5,6,7]. This work was concerned only with the synthesis of logically correct protocols. In this paper we show how the approach can be extended to encompass non-functional criteria too. In the experiments reported in this paper we have used the well-established technique of *simulated annealing* [10] (although our implementation allows rapid interchange of optimisation techniques). Below is a brief introduction to the simulated annealing algorithm.

4.1 Optimisation Technique

In 1983 Kirkpatrick et al. [10] proposed *simulated annealing*, a search technique inspired by the cooling processes of molten metals. It merges hill-climbing with the probabilistic acceptance of non-improving moves to find a good state $S \in State$. The basic algorithm is shown in Figure 2.

The search starts at some initial state $S_0 \in State$. There is a control parameter $T \in \mathbb{R}^+$ known as the *temperature*. This starts high at T_0 and is gradually lowered, typically by *geometric cooling* (that is, by multiplying by a *cooling factor*, $\alpha \in (0, 1)$ at each iteration).

At each temperature, a number *MIL* (Moves in Inner Loop) of moves to new states are attempted. A candidate state Y is randomly selected from the neighbourhood $N(S)$ of the current state. The new state Y is accepted if it is better or only slightly worse than S, as measured by a fitness function

$S := S_0$
$T := T_0$
repeat until stopping criterion is met
    repeat *MIL* **times**
        Pick $Y \in N(S)$ with uniform probability
        Pick $U \in (0, 1)$ with uniform probability
        if $f(Y) > f(S) + T \ln U$ **then** $S := Y$
    $T := \alpha \times T$

Fig. 2. Basic Simulated Annealing for Maximisation Problems

$f \in State \to \mathbb{R}$. By "slightly worse" is meant "no more than $|T \ln U|$ lower". Here $U \in (0, 1)$ is a random variable, and so $T \ln U \in (-\infty, 0)$; the smaller T is, the more likely that this term is close to 0 and eventually only improving moves are accepted (that is, the technique reduces to hill climbing).

The algorithm terminates when some stopping criterion is met. Common stopping criteria, and the ones used for the work in this paper, are to stop the search after a fixed number *MaxIL* of inner loops have been executed, or else when some maximum number *MUL* of consecutive unproductive inner loops have been executed (that is, without a single move having been accepted).

Generally the best state achieved so far is also recorded (since the search may actually move out of it and subsequently be unable to find a state of similar quality). Another common improvement to efficiency is to generate U only when $f(Y) < f(S)$.

In the experiments, we used a geometric cooling rate of 0.97; the number of attempted moves at each temperature was 400, with a maximum of 300 iterations (temperature reductions) and maximum number of 50 consecutive unproductive iterations (that is, with no move being accepted). Further details of the annealing algorithm we used in our system can be found in the earlier papers [5,6,7].

4.2 Protocol Representation and a Move Function

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 are vectors of its current beliefs and formulæ. Each of the M messages is represented by $F + 3$ integers, $s, r, k, f_1, \ldots, f_F$. These represent the sender, the receiver, the key that the sender used to encrypt this message, and a series of F indices that reference formulæ currently possessed by the sending principal. So, the sender is $s \bmod N$; the receiver is $r \bmod N$; the

key is k mod NK (where NK is the number of keys possessed by the sender); and the first component in the message is f_1 mod T etc., where the sender possesses T formulæ, indexed $0 \ldots T-1$. Formula 0 is the null formula (which allows us to model messages with fewer than F "real" formulæ). Key 0 is the null key (message in plaintext is treated as "encrypted" by the null key). The vectors of the receiver's current beliefs and formulæ are 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 formulæ they actually hold).

The above strategy allows a very simple move function for local search — simply randomly replace some of the integers involved in some messages. [4] Although any integer sequence gives rise to a feasible protocol, the protocol may not satisfy our required goals. The fitness function below measures how close it comes to achieving the required goals and our search seeks to find a protocol that satisfies all these goals.

4.3 Executing a Protocol

Assume that a protocol consists of M messages, each of which consists of F formulæ, and we start from the very beginning of this protocol. Firstly, we initialise the belief and formula state of the relevant principals involved in this protocol. Then, for each message in this protocol, we follow the steps below.

(i) Determine the sender, receiver, and the key under which the current message is encrypted. If this key is an appropriate one for communication between the sender and the receiver, then proceed with the rest of the current message, else ignore this message and proceed to the next message. The method of decoding the sender, receiver, and key is given in subsection 4.2.

(ii) Decode each of the F formulæ corresponding to the current message. For instance, the first formula in the message is f_1 mod T, where the sender currently holds T sendable formulæ.

(iii) Update the receiver's beliefs vector by applying the SVO axioms. Here we demonstrate what a principal A will do after it receives a message $\{X, B, N_a\}_{k_{ab}}$ from another principal B (assume both A and B initially hold k_{ab}, and they both believe that k_{ab} is a good key for communication). Firstly, after decryption, A recognises that this message is comprehensible, thus B said X, B said B, B said N_a, B has X, B has B and

[4] Note: A perturbed sequence may actually give rise to the same protocol as the original sequence. For example, if an integer denoting a sender has its value changed by a multiple of N it will be interpreted as the same principal as before.

B has N_a are added to A's belief vector (this represents A believes B said X, A believes B said B, A believes B said N_a, A believes B has X, A believes B has B and A believes B has N_a) whilst X, B and N_a are added to A's formula vector.[5] After this, A examines the set of received formulæ to see whether any of them are believed to be fresh. In this case, A retrieves the formula N_a, and as A believes the nonce N_a is fresh, then the whole message is regarded as fresh.[6] As the message is fresh, B says X, B says B and B says N_a are added to A's belief vector.[7] Similarly, other axioms now may be applied to deduce further beliefs until no further beliefs can be created.

(iv) 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 subsection 4.4.

4.4 The Fitness Function

The fitness function is used to guide the search for a 'good' solution, that is, the fitness function must tell us how good a candidate solution is. We use fitness functions, $f(P)$, which are sums of a *security fitness function*, s and an *efficiency fitness function*, e.

$$f(P) = s(P) + e(P)$$

The function $s(P)$ is defined as follows:

$$s(P) = \sum_{i=1}^{N} (\sigma + \delta(i)) \times G(P, i)$$

Here N is the maximum number of messages we allow in any protocol; $G(P, i)$ is the number of new required security goals that message i of P achieves; σ is, typically, a large constant that weights security much more heavily than efficiency and the $\delta(i)$ are weights among the individual messages. These weights, $\delta(i)$, were chosen to represent one of two strategies for finding protocols, *early credit (EC)* for achieving goals early in the protocol and *uniform credit (UC)* which does not favour one message over another; see Table 1. Further details of these strategies are given by Clark and Jacob [6,7].

[5] By applying the SVO *Receiving Axioms*, *Source Association Axiom* and *Possession Axiom*.
[6] By applying the SVO *Freshness Axiom*.
[7] By applying the SVO *Nonce-verification Axiom*.

Strategy	Weight						
	$\delta(1)$	$\delta(2)$	$\delta(3)$	$\delta(4)$	$\delta(5)$	$\delta(6)$	$\delta(7)$
EC	640	320	160	80	40	20	10
UC	160	160	160	160	160	160	160

Table 1
Weighting Strategies for $N = 7$

The function $e(P)$ is a also a sum of fitness functions, one for each kind of fitness we consider.

$$e(P) = m(P) + c(P) + r(P)$$

The fitness function $m(P)$ punishes protocols with many messages.

$$m(P) = \mu \times M(P)$$

where $M(P)$ is the highest index of a message of P that contributes new goals and $\mu < 0$ is the weight we give to this.

The function $c(P)$ punishes protocols with more encryption.

$$c(P) = \kappa \times C(P)$$

where $C(P)$ is the number of encryptions in P and $\kappa < 0$ is the weight we give to this.

The function $r(P)$ punishes numbers of interactions with particular principals.

$$r(P) = \sum_{a \in A(P)} \rho(a) \times R(P, a)$$

where $R(P, a)$ is the number of rounds involving principal a in P, $A(P)$ is the set of principals in P and $\rho(a) < 0$ are weights which allow us to declare that interactions with some principals (for example, the server) are worse than others.

5 Experimental Method and Results

This section reports the results of applying the technique described above to the derivation of three-party symmetric key distribution protocols.

5.1 Assumptions

Three parties participate in this key distribution protocol: A, B and S. Their identifiers are held by each other initially.

A holds a long term secure key k_{as} that he believes to be secure for communicating with the key distribution server S. A maintains his own nonce N_a that he believes to be fresh. A is able to recognise his own identifier A and nonce N_a.

Similar to principal A, principal B holds a long term secure key k_{bs} that he believes to be secure for communicating with S. B maintains his own nonce N_b that he believes to be fresh. B is able to recognise his own identifier B and nonce N_b.

The server S holds k_{as}, k_{bs} and believes that they are secure for communicating with A and B respectively. S also holds k_{ab} and believes that it is a good key that can be used between A and B. The assumptions are:

A has (A, B, S)	A has k_{as}	A believes $A \overset{k_{as}}{\longleftrightarrow} S$
A has N_a	A believes fresh(N_a)	
A believes $\phi(A)$	A believes $\phi(N_a)$	A believes S controls $A \overset{k_{ab}}{\longleftrightarrow} B$
B has (A, B, S)	B has k_{bs}	B believes $B \overset{k_{bs}}{\longleftrightarrow} S$
B has N_b	B believes fresh(N_b)	
B believes $\phi(B)$	B believes $\phi(N_b)$	B believes S controls $A \overset{k_{ab}}{\longleftrightarrow} B$
S has (A, B, S)		
S has k_{as}	S believes $B \overset{k_{as}}{\longleftrightarrow} S$	
S has k_{bs}	S believes $B \overset{k_{bs}}{\longleftrightarrow} S$	
S has k_{ab}	S believes $B \overset{k_{ab}}{\longleftrightarrow} S$	

5.2 Goals

The first set of goals requires that at the end of the protocol run A and B must each believe that it possesses a good key k_{ab} for session communication, and that each believes the other possesses the same key.

A has k_{ab}	A believes $A \overset{k_{ab}}{\longleftrightarrow} B$
B has k_{ab}	B believes $A \overset{k_{ab}}{\longleftrightarrow} B$
A believes B has k_{ab}	B believes A has k_{ab}

σ	3000	Correctness is more important than efficiency
$\delta(i)$	EC	See Table 1 for the definition
μ	-200	
κ	-100	
$\rho(S)$	-100	S is the server
$\rho(A)$	-50	A is a client
$\rho(B)$	-50	B is a client

Table 2
Fitness function weightings for the first search

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{as}}$
3. $B \rightarrow S : B, A, N_b$
4. $S \rightarrow B : \{N_b, N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{bs}}$
5. $B \rightarrow A : \{B, N_b, N_a\}_{k_{ab}}$
6. $A \rightarrow B : \{N_b, A\}_{k_{ab}}$

Fig. 3. A symmetric key protocol found by our tool in the first search

5.3 Results

5.3.1 The first search

In our first search the annealing parameters given in subsection 4.1 were used. We took the maximum number of messages, $N = 7$. The weights we used for the fitness function are given in Table 2. We used the *EC* strategy in order to find a protocol that is able to satisfy security requirements as soon as possible.

Figure 3 shows one of the symmetric key protocols generated by the program. Only the core security relevant components of the protocol are presented. That is, we have removed components from the description that do not contribute to the goals. In addition, redundant components (where the same components are included twice or more in one message) have also been removed. (Currently, redundant components are removed by hand; automating their removal is under investigation.)

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : A, N_a, B, N_b$
3. $S \rightarrow B : \{N_b, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{bs}}$
4. $S \rightarrow A : \{N_a, N_b, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{as}}$
5. $A \rightarrow B : \{A, N_a, N_b\}_{k_{ab}}$
6. $B \rightarrow A : \{B, N_a\}_{k_{ab}}$

Fig. 4. A protocol with fewer server interactions, found by our tool when asked to try harder to minimise server interactions

5.3.2 The second search

For our second search we asked our tool to try harder to minimise server interactions. To do this we doubled each of $\rho(S)$, $\rho(A)$ and $\rho(B)$ (that is, we let fitness function $r(P)$ carry more weight relative to the other properties) and used the *UC* strategy for the $\delta(i)$ (see Table 1). All other parameters are as in Table 2.

A new protocol (see Figure 4) with fewer server interactions than any protocol generated in the first search was found by our tool.

The first search might have produced the protocol in Figure 4, but without the demands given in the second search, it did not.

5.4 Comments

It is possible to ask our tool to find a protocol for an assumptions/goals pair that cannot be met, or is difficult to meet. In such cases, the search will terminate without producing a protocol or produce a protocol that only satisfies some of the goals.

One pleasing feature of the work is the speed at which protocols are generated. A typical run takes two minutes or less (on a Pentium Mobile 1.5G processor with 512MB memory, *Java*™ 2 SDK 1.4.2 and Borland JBuilder 9). This compares very favourably with other design synthesis approaches, for example model checking [11].

6 Conclusions and Further Work

The above work shows that meta-heuristic search approaches to secure protocol synthesis are potentially powerful and have the benefit of being rapid. Our tools generate candidate protocols rapidly and concrete refinements of them could be subjected to more detailed and sophisticated analysis (such as

that provided by current model checking approaches). This would provide an interesting synthesis of current techniques. The protocols generated, although simple, are typical abstractions of protocols in the literature.

There is very little published work in the field of protocol efficiency. With our protocol synthesis approach, we have provided a framework for incorporating efficiency concerns into the synthesis approach (in terms of the number of messages, server interactions etc.).

However, we note that the efficiency of a protocol cannot be fully characterised independently of its implementation details. Thus, one topic of further work is to investigate more efficiency issues of security protocols and how to make tradeoffs among these concerns.

In this paper, we demonstrate that our system has the ability to synthesise three-party authentication and key transport protocols using symmetric key cryptography. Key agreement protocols have become much more popular than key transport protocols in recent years. SVO logic has the ability to analysis key agreement protocols. We believe that our synthesis system can easily be extended to encompass public key scheme and synthesise key agreement protocols.

References

[1] Martin Abadi and Mark Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th ACM Symposium on Principals of Distributed Computing*, pages 201–216. ACM Press, 1991.

[2] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[3] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, 1989.

[4] Levente Buttyán. Formal methods in the design of cryptographic protocols (state of the art). Technical Report SSC/1999/038, EPFL SSC, 1999.

[5] Hao Chen, John A. Clark, and Jeremy L. Jacob. Automated design of security protocols. In *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 2181–2188. IEEE Press, 2003.

[6] John A. Clark and Jeremy L. Jacob. Search for a solution: Engineering tradeoffs and the evolution of provably secure protocols. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, pages 82–95. IEEE Computer Society Press, 2000.

[7] John A. Clark and Jeremy L. Jacob. Protocols are programs too: The meta-heuristic search for security protocols. *Information and Software Technology*, 43(14):891–904, 2001.

[8] Li Gong. Efficient network authentication protocols: Lower bounds and optimal implementations. *Distributed Computing*, 9(3):131–145, 1995.

[9] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of 1990 IEEE Symposium on Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[11] Adrian Perrig and Dawn 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.

[12] Paul Syverson and Iliano Cervesato. The logic of authentication protocols. *Lecture Notes in Computer Science*, 2171:63–136, 2001.

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

[14] Paul van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 232–243. ACM Press, 1993.