Fast Automatic Synthesis of Security Protocols Using Backward Search

Hongbin Zhou
Department of Computer Science
University College, Cork
Cork, Ireland
zhou@cs.ucc.ie

Simon N. Foley
Department of Computer Science
University College, Cork
Cork, Ireland
s.foley@cs.ucc.ie

ABSTRACT

An automatic security protocol generator is proposed that uses logic-based synthesis rules to guide it in a backward search for suitable protocols from protocol goals. The approach taken is unlike existing automatic protocol generators which typically carry out a forward search for candidate protocols from the protocol assumptions. A prototype generator has been built that performs well in the automatic generation of authentication and key exchange protocols.

In solving a problem of this sort, the grand thing is to be able to reason backward.
—Sir Arthur Conan Doyle (Sherlock Holmes),
A Study in Scarlet, 1887.

Categories and Subject Descriptors

C.2 [**Computer-communication networks**]: Security and protection; C.2.2 [**Network Protocols**]: Protocol Verification; K.6.5 [**Security and Protection**]: Authentication; D.2.4 [**Software/Program Verification**]: Formal methods

General Terms

Security, Design

Keywords

security protocols, automatic generation, backward search, belief logic

1. INTRODUCTION

Security Protocols are widely used in distributed systems for authentication, key exchange and other security requirements. Designing well behaved security protocols is a challenging task since protocols often contain subtle flaws that are difficult to find. In the last 20 years, many approaches for verifying properties of security protocols have been developed such as [5, 10, 9, 19, 17, 22]. However, little work has been carried out on systematic approaches to the design and development of security protocols

Abadi and Needham [2] set out ten principles that help designers avoid classes of known protocol flaws. However, the principles are neither necessary nor sufficient: designers can not necessarily design new protocols by obeying only these principles. A number of formal design approaches for security protocols have been proposed. [3] describes a weakest-precondition based approach for the design of security protocols. The Simple/BSW logic [6] is a BAN-like logic that provides synthesis rules to guide the protocol designer in the manual systematic calculation of a protocol from its goals.

We are interested in the automatic generation of a protocol from its protocol goals and assumptions. Existing research includes Clark and Jacob's evolutionary search [8] and Perrig and Song's Automatic Protocol Generator (APG) [18]. Both approaches automatically search a large space of candidate protocols that is far larger than could be considered via a manual design.

Starting from a set of assumptions, [8] uses the original inference rules of the BAN logic to systematically test whether candidate protocols uphold the protocol goals. The evolutionary approach starts from a set of assumptions, and uses a fitness function to guide the application of BAN rules in a forward manner until a valid protocol is arrived at. However, providing an accurate fitness function for a protocol is very difficult, and in cases potentially impossible [8]. Perrig and Song's APG [18] uses heuristics to select random candidate protocols that are in turn checked for validity using the Athena [20] security protocol checker.

Guttman [11] proposes a manual protocol design methodology that is based on authentication tests [12]. For different goals, individual subprotocols are generated that are combined together to form the final protocol. However, it is a manual design process and relies on the skill of the protocol designer.

In this paper, we propose the Automatic Synthesis Protocol Builder (ASPB) that uses a novel approach to automatically generating security protocols. The approach combines and automates the manual synthesis rules from the BSW Logic with Guttman's manual design process. We adapt the synthesis rules of the BSW logic to guide an automatic backwards search for a sub-protocol from a single goal. Given a number of individual goals, an automated

FMSE'03, October 30, 2003, Washington, DC, USA.

technique is proposed to combine synthesized sub-protocols into final candidate protocols.

This paper is organized as follows. The BSW logic is outlined in Section 2. Section 3 describes the automatic protocol synthesizer in detail; this section also describes how the synthesis rules can be used as heuristics to guide the automatic backward search for protocols within the logic. We discuss the advantages of our approach and compare it with other approaches in Section 4. The Appendix contains an example of a protocol requirement specification and a number of sample protocols that were generated.

2. THE BSW LOGIC

The BSW Logic [6] is a BAN-style belief logic that uses abstract channels similar to the Spi Calculus [1] to represent keyed communication between principals. The set of principals that can receive and can send messages via a channel C is denoted by its reader set $r(C)$ and its writer set $w(C)$, respectively. For example, $r(C) = \Omega$ and $w(C) = \{P\}$ represents an authentic channel, whereby any principal (from Ω) can authenticate messages signed by the private key of principal P.

The logic uses the following basic formulae. P,Q range over principals; C represents the channel; X represents a message which can be data or formulae or both; ϕ represents a formula.

- $P \triangleleft X$: Principal P sees message X. Someone has sent X via a channel that P can read.
- $P \triangleleft C(X)$[1]: P sees $C(X)$. Someone has sent a message X via channel C. If P can not read C then P can not discover the content of X.
- $P \mid\sim X$: P once said X. P sent a message contained X at some point in the past. We do not know exactly when the message was sent.
- $P \parallel\!\sim X$: P says X. P sent X in the current run of the protocol.
- $\sharp(X)$: Message X is fresh. X has never been said before the current run of the protocol. This is usually true for messages containing nonces.
- $P \models \phi$: P believes that ϕ is true. It does not mean that ϕ is really true, but P believes it.

The BSW logic also uses the conventional logic operators $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication) from propositional logic and some basic notation from set theory.

In the BAN logic principals are treated as trustworthy and in GNY there are fixed axioms for reasoning about the trustworthiness of a principal. In BSW, the rules about the trustworthiness of a principal are expressed as formulae as part of the assumptions of the protocol.

EXAMPLE 1. *(Adapted from [6]). Mutual authentication between principals A and B may be expressed as goals:*

$$G_1 \triangleq A \models (B \parallel\!\sim (A, Na))$$
$$G_2 \triangleq B \models (A \parallel\!\sim (B, Nb))$$

where Na is a nonce (and assumptions include $A \models \sharp(Na)$, and so forth). We assume that A and B share symmetric keys (abstracted as channels Cas and Cbs, respectively) with third party S. These assumptions are defined as follows.

$$r(Cas) = \{A, S\}; r(Cas) = \{A, S\};$$

$$A \models (w(Cas) = \{A, S\}); S \models (w(Cas) = \{A, S\});$$

A further assumption is that A trusts S as a trusted third party:

$$A \models ((S \parallel\!\sim \phi_1) \rightarrow (A \models \phi_1))$$
$$A \models ((S \parallel\!\sim (B \mid\sim \phi_2)) \rightarrow (B \mid\sim \phi_2))$$

for arbitrary ϕ_1, ϕ_2. These formulae reflect A's belief that S is honest and that S is competent in deciding whether B at some time in the past said some message. B has similar beliefs. △

In addition to basic axioms about sets, the logic uses four core axioms.

S1 Seeing. If P receives a message X via a channel C, and P can read this channel, then P can see the message.

$$\frac{P \triangleleft C(X), P \in r(C)}{P \triangleleft X}$$

F1 Freshness. If P believes another principal Q once said a message X and P believes that X is fresh, then P believes that Q says X.

$$\frac{P \models (Q \mid\sim X), P \models \sharp(X)}{P \models (Q \parallel\!\sim X)}$$

I1 Interpretation. If P believes he receives a message via C, then he believes that the message was said by someone, who he believes is able to write channel C except himself.

$$\frac{P \triangleleft C(X), P \in r(C), P \models (w(C) = \mathcal{W})}{P \models \bigvee_{\forall Q_i \in \mathcal{W} \setminus \{P\}} (Q_i \mid\sim X))}$$

R1 Rationality. This is the well-known K axiom of modal logic: if P believes ϕ_1 implies ϕ_2, and believes that ϕ_1 is true, then he believes that ϕ_2 is true.

$$\frac{P \models (\phi_1 \rightarrow \phi_2), P \models \phi_1}{P \models \phi_2}$$

3. ASPB

Figure 1 outlines the the architecture of ASPB. A protocol specification (assumptions and goals) are parsed and decomposed into a series of single goal protocol requirements from which a collection of subprotocols are synthesized from individual goals using the Single Goal Synthesizer. The Protocol Composer merges the subprotocols to form complete candidate protocols. The Protocol Selector selects what is considered the most suitable protocol from the candidate protocols.

3.1 The Requirement Specification

The protocol specification defines the known assumptions and goals for the protocol to be designed. We use the BSW logic syntax developed in [15] to represent necessary declarations, assumptions and goals. Appendix A gives the complete specification for a mutual authentication symmetric key protocol that uses a trusted third party (TTP) (introduced in Example 1).

[1]Note that we do not use the related formula $P \triangleleft X \mid C$ from [6] which defines that P sees message X via channel C since it can be replaced by the other formulae in the deduction axioms without any loss of expressiveness.

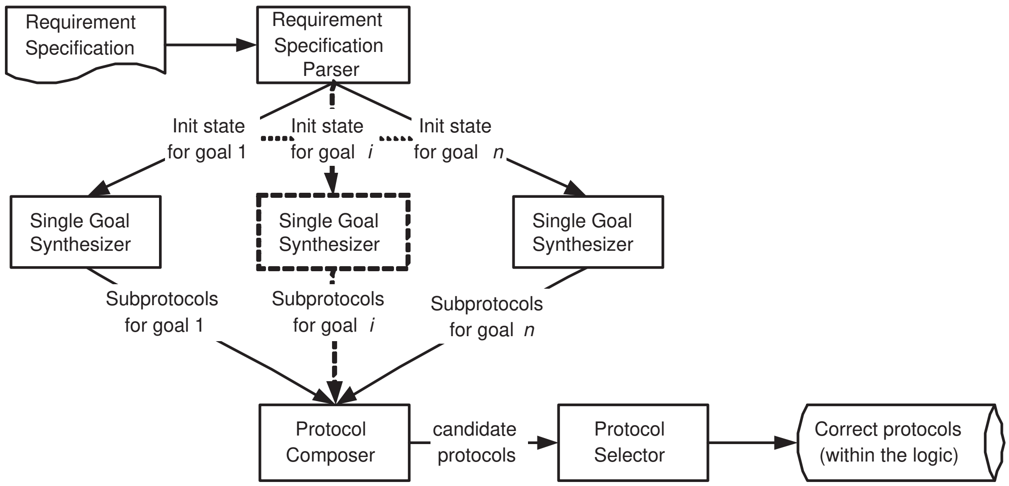

Figure 1: Overview of ASPB

3.2 The Requirement Specification Parser

The Requirement Specification Parser parses the specification and uses basic set theory axioms to deduce all available assumptions from the specified assumptions and any currently established goals. For each goal to be proven, the parser generates an initial state that includes all available assumptions and a formula tree whose root is a goal in the requirement specification.

3.3 Heuristic Rules for Single Goal Synthesis

The BSW Logic includes a synthesis technique that can be used to guide the (manual) systematic calculation of a protocol from its goals. Synthesis rules take the general form

$$\begin{array}{ll} G & \\ \hookrightarrow & G_1 \\ \hookrightarrow & ... \end{array}$$

which means that in order to reach the goal G, all subgoals G_1, G_2, ... have to be reached. A goal G can have the form G'/G'', which means that either G' or G'' have to be reached. Nine synthesis rules of this form are proposed in [6]. We use these synthesis rules to develop core heuristics to guide the automatic backwards search for candidate protocols from their goals. These heuristics are derived as theorems within the logic.

Heur1 To see message X, P must receive $C(X)$ and be able to read C.

$$\begin{array}{ll} P \triangleleft X & \\ \hookrightarrow & P \triangleleft C(X) \\ \hookrightarrow & P \in r(C) \end{array}$$

Heur2 To believe Q says X, P must believe that Q said X and X is fresh.

$$\begin{array}{ll} P \models Q \Vdash\!\sim X & \\ \hookrightarrow & P \models Q \mid\!\sim X \\ \hookrightarrow & P \models \sharp(X) \end{array}$$

Heur3 To believe that message X is fresh, P must believe that some part X' of X is fresh.

$$\begin{array}{ll} P \models \sharp(X) & \\ \hookrightarrow & P \models \sharp(X') \end{array}$$

Heur4 To believe that Q said X, P has to receive X via a channel C that he can read and that he believes it can be written only by Q, or P and Q. Furthermore, Q has to see X.

$$\begin{array}{ll} P \models Q \mid\!\sim X & \\ \hookrightarrow & P \triangleleft C(X) \\ \hookrightarrow & P \in r(C) \\ \hookrightarrow & P \models (w(C) = \{Q\})/ \\ & P \models (w(C) = \{P, Q\}) \\ \hookrightarrow & Q \triangleleft X \end{array}$$

If X is a formula and P believes that Q is honest, then Q must also believe X.

$$\begin{array}{ll} P \models Q \mid\!\sim X & \\ \hookrightarrow & P \triangleleft C(X) \\ \hookrightarrow & P \in r(C) \\ \hookrightarrow & P \models (w(C) = \{Q\})/ \\ & P \models (w(C) = \{P, Q\}) \\ \hookrightarrow & Q \models X \\ \hookrightarrow & P \models ((Q \Vdash\!\sim X) \to (Q \models X)) \end{array}$$

Heur5 To believes ϕ_1, P must believe ϕ_2 and $\phi_2 \to \phi_1$.

$$\begin{array}{ll} P \models \phi_1 & \\ \hookrightarrow & P \models \phi_2 \\ \hookrightarrow & P \models (\phi_2 \to \phi_1) \end{array}$$

3.4 The Single Goal Synthesizer

An automatic verification tool [15] for the BSW Logic has been implemented using Theory Generation [13]. This tool [15] also supports (manually guided) synthesis of protocols using the synthesis rules described in [6]. The Single Goal Synthesizer described in this paper builds on this manual tool in [15] by automatically carrying out the synthesis process.

The Single Goal Synthesizer accepts an initial state from the Requirement Specification Parser, and decomposes its goal by the heuristic rules above. Algorithm 1 describes this process.

Algorithm 1 StateSet syn(State initState)

```
State s;
StateSet L = {initState};
StateSet R = φ;
while ¬ empty(L) do
  s = choose(L);
  L = L \ s;
  if hasGoal(s) then
    S' = subsyn(s);
    L = add(L, S');
  else
    R = add(R, s);
  end if
end while
return R;
```

Operation $choose(L)$ picks an arbitrary state from the state set L.

Operation $subsyn(s)$ returns the next transition state set of s. It selects an interim subgoal of s, and applies the currently applicable heuristic rules to the subgoal. For the given assumptions it generates all conclusions by all applicable logical rules.

If no heuristic rule can be applied to a subgoal then it is a leaf in the tree. If a leaf matches an assumption, or has the format $P \triangleleft C(X)$ (a protocol step), then it is a reachable leaf, also called it a terminal subgoal. If all leaves in a tree are reachable subgoals, then there is a possible subprotocol for the goal which is the root of the tree and it is added to the set of returned protocols R.

After the application of a heuristic rule, the Single Goal Synthesizer checks whether the resulting subgoals are interim or terminal subgoals. If the subgoal matches a protocol message or an assumption then it is a terminal subgoal and the searching at this point will complete. Otherwise, in the case of an interim subgoal it added to the current state s.

Figure 2 gives an example of the formula tree generated from the goal G_1 of Example 1 (using assumptions from the complete specification in Appendix A). Initially, two new subgoals $A \models \sharp(A, Na)$ and $A \models (B \mid\!\sim (A, Na))$ are deduced by applying Huer 2 to G_1. Applying Huer 3 to subgoal $A \models \sharp(A, Na)$, formula $A \models \sharp(Na)$ is deduced which is an assumption in the protocol requirement specification and thus the search on this branch terminates (reachable leaf). The heuristic rules for other subgoals are similarly applied, resulting in the tree in Figure 2. Note that the temporal ordering of subgoals in the heuristics is important. When Heur 4 is applied to the subgoal $S \models (B \mid\!\sim (A, Na))$ in Figure 2, B must see the message (A, Na) *before* it is seen by S, otherwise B can not write X to channel C_{bs}.

EXAMPLE 2. *Continuing Example 1, two subprotocols are automatically synthesized from Goal G_1. Subprotocol 1.1 corresponds to the protocol messages from the search tree in Figure 2.*

Subprotocol 1.1

$$\begin{array}{lcl} A & , & \\ B & \triangleleft & C_p(A, Na), \\ S & \triangleleft & C_{bs}(A, Na), \\ A & \triangleleft & C_{as}(B \mid\!\sim (A, Na)). \end{array}$$

Synthesis generates a further search tree with corresponding Subprotocol 1.2.

Subprotocol 1.2

$$\begin{array}{lcl} A & , & \\ S & \triangleleft & C_{as}(A, Na), \\ B & \triangleleft & C_{bs}(A, Na), \\ A & \triangleleft & C_p(A, Na). \end{array}$$

Note that the first line of a subprotocol indicates the subprotocol initiator. The synthesis of the symmetrically similar goal G_2 generates two similar search trees from which symmetrically similar subprotocols 2.1 and 2.2 are obtained(See Appendix A.2). △

The Single Goal Synthesizer uses the heuristics to direct its backwards search for valid protocols from a protocol goal. This optimal strategy ensures that all candidate protocols obtained from the search tree are valid in that they uphold the goal within the logic. This contrasts with the forward searching approaches that may process many invalid candidate protocols before encountering a valid protocol.

However, unlike the forward searching techniques, our approach generates protocols in a deterministic manner, due in part to the treatment of free variables by the heuristics (for example, Heur 5). In theory these free variables could be bound to any logical formula, resulting in an infinite state space. In practice, we match those formulae only with held assumptions. Despite this restriction, we can still generate useful protocols; investigating alternative strategies for binding free variables is a topic for future research.

3.5 The Protocol Composer

The Single Goal Synthesizer is used to generate protocols from a single goal. While not considered in the original paper [6], it is possible in theory to use the BSW synthesis rules to synthesize multiple goals. However, in practice, building a search tree for multiple goals results in a potential state explosion as each step must consider the application of all possible combinations of heuristic rules that could be applied in the current state. ASPB avoids this problem by synthesizing only single goal protocols and uses the Protocol Composer to, in turn, merge the resulting subprotocols that are generated from individual protocol goals into a single candidate protocols that meets the composition of goals.

A security protocol is a sequence of message exchanges between principals to achieve some security goals. At an abstract level, these message exchanges can be described just in terms of principal sequencing. For example, subprotocol 1.1 (Example 1) has principal sequence $A \rightarrow B \rightarrow S \rightarrow A$; subprotocol 2.1 (in Appendix) has principal sequence $B \rightarrow A \rightarrow S \rightarrow B$; The Protocol Composer uses principal sequencing to guide the construction of new protocols by merging subprotocols messages.

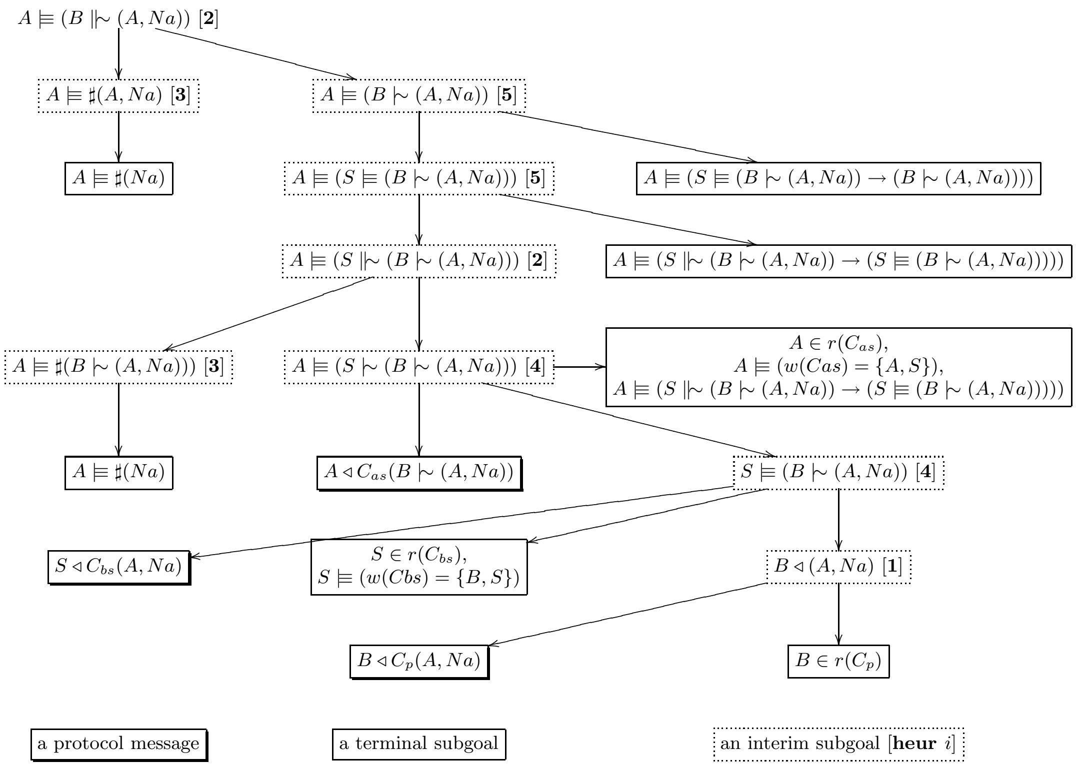

Figure 2: A Searching State for Goal $G_1 \triangleq A \models (B \Vdash\!\sim (A, Na))$

We define the merge of two subprotocols as follows. Define a principal sequence to be a sequence of principal names that specifies the order of message exchanges in a protocol. Given principal sequences X and Y, then sequence X *covers* sequence Y if Y appears as a fragmented subsequence of X. For example, $A \rightarrow B \rightarrow A \rightarrow B \rightarrow C$ covers $A \rightarrow C$. Let $s(P)$ define the principal sequence of protocol P. A merge of subprotocols P_1 and P_2 is given by a protocol P where $s(P)$ covers the sequences $s(P_1)$ and $s(P_2)$.

There can be many possible ways to merge two subprotocols. The easiest way to merge two subprotocols would be to append one to the end of the other. However, this may lead to lengthly and inefficient protocols. This merging can be optimized if the sub-protocols contain common sequence fragments. For example, subprotocol sequences $A \rightarrow B \rightarrow S \rightarrow A$ and $B \rightarrow A \rightarrow S \rightarrow B$ include fragments of the form $A \rightarrow \cdots \rightarrow S$ and $A \rightarrow \cdots \rightarrow B$. These can be used to guide the merging of the subprotocols to give an shorter composition $A \rightarrow B \rightarrow A \rightarrow S \rightarrow A \rightarrow B$ and covers original subprotocol sequences.

Given a collection of subprotocols $P_1 \ldots P_n$ that meet goals $G_1 \ldots G_n$ of a protocol specification, respectively, then the Protocol Composer searches for the shortest principal sequence that covers the sequences of the subprotocols $P_1 \ldots P_n$. This is done using a variation of the shortest subsequence algorithm. The final candidate protocol P is constructed using the generated principal sequence $s(P)$ and the original subprotocols $P_1 \ldots P_n$ according to the following rules.

- *Early appearing rule.* A message from a subprotocol P_i should appear in the candidate protocol P as early as possible, constrained only by the principal sequences.
- *Merging rule.* Messages on common channels in the subprotocols should be merged in the candidate protocol, subject to the constraints of the principal sequences. For example, message $C_1(X_1), C_2(X_2, Y_1)$ and $C_1(X_3), C_2(X_2, Y_2)$ merge into resulting message $C_1(X_1, X_3), C_2(X_2, X_2, Y_1, Y_2)$.
- *Reducing rule.* Any redundant message components should be reduced. For example, message $C(X_2, X_2)$ should be reduced to $C(X_2)$.

The Single Goal Synthesizer generates messages expressed as formulae within the logic. The final implementation of these protocols will be in terms of conventional messages. For example, message $Cas(A \mid\!\sim Na)$ and $Cas(A \triangleleft Na)$ should be expressed by the same format $Cas(A, Na)$. To minimize the potential for replay attacks where 'similar' messages appear in different parts of a protocol, the messages are modified to make them distinct from one another. For example, $Cas(A, Na, 0)$ and $Cas(A, Na, 1)$ or $Cas(A, Na)$ and $Cas(Na, A)$.

EXAMPLE 3. *The synthesis of goals G_1 and G_2 generate subprotocols 1.1 and 1.2 and subprotocols 2.1 and 2.2, respectively. Thus there are 2×2 possible combinations of the protocols to be considered for merging. Furthermore, for each pair of subprotocols, we must find the shortest merge of the two subprotocols. ASPB generates the following 'best'* *protocol (in 3.1 seconds):*

$$\begin{array}{lll} A & , & \\ B & \triangleleft & C_p(A, Na), \\ S & \triangleleft & C_{bs}(A, B, Na, Nb), \\ A & \triangleleft & C_{as}(A, B, Na, Nb), \\ B & \triangleleft & C_p(B, Nb). \end{array}$$

This protocol is the merge of subprotocols 1.1 and 2.2 from Example 1. △

In general, given subprotocols SP_1 and SP_2 that uphold goals G_1 and G_2, then the monotonicity of the BSW logic ensures that the resulting merged candidate protocol as outlined above also upholds the goals G_1 and G_2 within the logic. However, there are many examples of secure protocols, which when composed are vulnerable to attack. For example, the Protocol Composer generates the following simple mutual authentication protocol.

$$\begin{array}{lll} Msg1 & A \to B: & A, Na, \\ Msg2 & B \to A: & B, Nb, \{A, Na\}_{K_{ab}} \\ Msg3 & A \to B: & \{B, Nb\}_{K_{ab}} \end{array}$$

While secure within BSW and many other belief logics, this protocol is subject to a reflection/oracle attack. We suggest that techniques such as [4] may prove useful in making candidate protocols robust against such attacks. For example, by ensuring that the initiator challenge looks different to the respondent challenge. This is the purpose of the Protocol Selector. Alternatively, the generated protocols could be reanalyzed using a more sophisticated protocol analysis tools. In this latter case, the ASPB would be used to narrow down the set of candidate protocols to be verified.

3.6 The Protocol Selector

All the protocols generated by the Protocol Composer are valid within our logic. However, belief logics do have weaknesses and it is useful to consider verification of additional ad-hoc properties. For example, the non-injective agreement property [14]: *"For certain data items ds, if each time a principal B completes a run of the protocol as responder using ds, apparently with A, then there is a unique run of the protocol with the principal A as initiator using ds, apparently with B."* Such unsuitable protocols that are easy to identify are discarded following verification by typical protocols checkers such as FDR, Athena. The Protocol Selector's role is to similarly discard these unsuitable candidates.

Example 4. *ASPB uses heuristics from [23] to remove redundant information from messages. For example, the logic formulae in the generated protocol from Example 3 are reduced and then simplified using Heuristic 2 [23] to give:*

$$\begin{array}{lll} Msg1 & A \to B: & A, Na, \\ Msg2 & B \to S: & B, \{A, Na, Nb\}_{K_{bs}}, \\ Msg3 & S \to A: & \{Na, Nb, B\}_{K_{as}}, \\ Msg4 & A \to B: & Nb. \end{array}$$

△

4. DISCUSSION AND EVALUATION

In this section, we evaluate ASPB and compare it with existing approaches, in particular, the Automatic Protocol Generator (APG) [18].

ASPB generates protocols that are similar to those generated by APG. However, ASPB does not generate protocols in the Protocol-Set S2 for authentication and key agreement from [18] (this includes the original Yahalom protocol). The reason for is that the protocols in Protocol-Set S2 have the assumption that B believes that A is honest. If B believes A accepts a session key then B will accept it. This assumption is not made in the requirement specification in Appendix A which was used to conduct our experiments.

We use "Heuristic 3" defined in [23] as: *"Encrypted replies from one of the parties need not be nested inside encrypted messages of the other party."*. Thus, for example, the message $\{M', \{M\}_{K_{XA}}\}_{K_{YA}}$ should be replaced by the corresponding message $(\{M\}_{K_{XA}}, \{M'\}_{K_{YA}})$.

4.1 Four Message Protocols

ASPB synthesizes the following four-message mutual authentication protocols from goals G_1 and G_2 (Example 1).

Protocol 3.1:

$$\begin{array}{lll} Msg1 & A \to B: & A, Na, \\ Msg2 & B \to S: & B, \{A, Na, Nb\}_{K_{bs}}, \\ Msg3 & S \to A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\ Msg4 & A \to B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}. \end{array}$$

Protocol 3.2:

$$\begin{array}{lll} Msg1 & A \to B: & A, Na, \\ Msg2 & B \to S: & B, \{A, Na, Nb\}_{K_{bs}}, \\ Msg3 & S \to A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\ Msg4 & A \to B: & Nb, \{Nb, K_{ab}\}_{K_{bs}}. \end{array}$$

Protocol 3.3:

$$\begin{array}{lll} Msg1 & A \to B: & A, Na, \{B, Na\}_{K_{as}}, \\ Msg2 & B \to S: & B, \{B, Na\}_{K_{as}}, \{A, Na, Nb\}_{K_{bs}}, \\ Msg3 & S \to A: & \{Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}\}_{K_{as}} \\ Msg4 & A \to B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}. \end{array}$$

Protocols 3.1 and 3.2 also appear in Protocol-Sets S1 and S3 from [18]. In Protocol 3.1 B believes that A receives K_{ab} from S. However, this is not the case in Protocol 3.2 since A does not use the key K_{ab} (to encrypt the nonce). In the new Protocol 3.3, when the TTP S receives $Msg2$, S may check whether two principals knows who the other party is in the current round. If S finds that one of attempts to cheat the other one, then it can stop the current round as early as possible. While Protocol 3.3 has a higher cost (in terms of message size) than the other protocols, it provides a more powerful TTP. Appendix B.2 describes further protocols that were generated in this category.

protocol purpose		ASPB		APG[a]
		Stage 1[b]	Stage 2[c]	
mutual authentication		3 sec.	4 sec.	10 min.
mutual authentication and key agreement	(4 messages)	15 sec.	20 sec.	2 hr.
	(5 messages)	25 sec.	80 sec.	N/A

[a]APG timing is based on generating the best protocol result running on on a 400MHz Intel Pentium III [18].
[b]Time to synthesize and generate all candidate protocols running on a 1.8GHz Intel Pentium IV.
[c]Estimated time that the Athena [20] checker would take to further validate the ASPB generated candidate protocols.

Table 1: The time performance comparison between ASPB and APG

4.2 Five Message Protocols

Carlsen describes a five message protocol, the Secret Key Initiator Protocol [7], as follows:

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow S: & A, B, Na, Nb, \\ Msg3 & S \rightarrow B: & \{Na, B, K_{ab}\}_{K_{as}}, \{A, Nb, K_{ab}\}_{K_{bs}}, \\ Msg4 & B \rightarrow A: & \{Na, B, K_{ab}\}_{K_{as}}, \{Na\}_{K_{ab}}, N'b, \\ Msg5 & A \rightarrow B: & \{N'b\}_{K_{ab}}. \end{array}$$

In this protocol Principal B uses two nonces. The protocol requirement specification (Appendix A) that formed the basis of our experiments specified that Principal B uses one nonce. A consequence of this is that the exact Carlsen protocol is not generated; however a number of not dissimilar protocols were generated by ASPB. We conjecture that the Protocol Composer generate the Carlsen protocol if the protocol specification was extended to include B use of two nonces.

Protocol 4.1:

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow S: & A, B, Na, Nb, \\ Msg3 & S \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\ Msg4 & B \rightarrow A: & \{Na, Nb, K_{ab}, B\}_{K_{as}}, \{Na\}_{K_{ab}}, \\ Msg5 & A \rightarrow B: & \{Nb\}_{K_{ab}}. \end{array}$$

Protocol 4.2:

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\ Msg3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}}, \\ Msg4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}, \\ Msg5 & B \rightarrow A: & \{Na\}_{K_{ab}}. \end{array}$$

Protocol 4.3:

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow A: & B, Nb, \{A, Na\}_{K_{bs}}, \\ Msg3 & A \rightarrow S: & A, \{Nb, B\}_{K_{as}}, \{Na, A\}_{K_{bs}}, \\ Msg4 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{Nb, K_{ab}\}_{K_{bs}}, \\ Msg5 & A \rightarrow B: & \{Nb, K_{ab}\}_{K_{bs}}, \{Nb\}_{K_{ab}}. \end{array}$$

Protocol 4.1 is similar to Carlsen's Secret Key Initiator Protocol. While B generates only one nonce Nb in Protocol 4.1, it achieves the same result as Carlsen's protocol. Once B receives $Msg3$, he can check whether S believes that B needs a session key with A, and S believes Nb is B's nonce. When A receives $Msg4$, she may believe that Nb is generated by B (otherwise, B may not generate $\{Na\}_{K_{ab}}$).

The first four message of Protocol 4.2 are the same as Protocol 3.1. From the additional message $Msg5$, Protocol 4.2 meets an additional goal that A believes B has received the session key Kab.

Protocol 4.3 is a novel protocol. When S receives $Msg3$, he may check whether the components have been generated by A and B. If this is the case then S sends $Msg4$. Further five-message protocols that were generated by ASPB are given in Appendix B.2.

4.3 Discussion

Table 1 provides a time performance comparison between ASPB and APG [18]. The first row gives the performance results for generating the mutual authentication protocol described in Example 1 (and specified in Appendix A). The other experiment was for mutual authentication and key agreement with TTP. The second row gives the result for 4 message protocols. The last row gives the result for 5 message protocols. The results of similar experiments were reported in [18].

It is clear from Table 1, that ASPB is faster than APG at generating correct protocols under comparable conditions. APG has not been tested for five message protocols; in this case we conjecture that the forward search approach of APG would result in a very large and potentially infeasible search space. ASPB generates approximately 500 valid protocols, however, on inspection, many of these protocols are similar, containing minor textual and redundant variations. On inspection, we estimate that in this set there are 24 reasonably distinct four-message candidate protocols and 76 reasonably distinct five-message protocols, though some of them are not the minimal cost protocols. We select several protocols for each category to demonstrate our results in this paper.

Since the Single Goal Synthesizer is completely independent of the Protocol Composer, our approach does not depend on the BSW logic. In future research, we can extend the logic (or change the basic logic to others) to suit a more complex environment.

Finally, by parallelizing the single goal synthesis and composition steps across separate processors it would be straightforward achieve increased performance.

5. CONCLUSION

In this paper we describe how the synthesis process of the BSW logic can be used to guide the automatic generation of security protocols. It uses a backward search approach: searching for suitable protocols from the protocol goals. This is unlike existing approaches which typically search in a forward manner from protocol assumptions for protocols that meet the required goals. This backward search approach limits the size of the search space and preliminary results described in this paper indicate a better performance than existing forward search techniques.

This backwards search forms the heart of the ASPB protocol generating tool. Single goals are synthesized to subprotocols, which are in turn composed to form the final protocols. Various heuristics are used to guide the selection and design of candidate protocols.

However, the approach does have some limitations. Firstly, by binding free variables only to formulae from known assumptions, the set of potential candidate protocols is reduced. Whether this is a significant constraint is a topic for future research. Secondly, the BSW logic is based on a belief logic and, as such, does have limitations when compared with other techniques such as [19, 17]. However, ASPB could be used narrow down the set of candidate protocols to be verified by a more sophisticated checker.

6. ACKNOWLEDGMENTS

We are grateful for helpful feedback from the anonymous referees. This work is supported by the Boole Centre for Research in Informatics, University College Cork under the HEA-PRTLI scheme.

7. REFERENCES

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privace*, pages 122–136. IEEE Computer Society Press, 1994.

[3] J. Alves-Foss and T. Soule. A weakest precondition calculus for analysis of cryptographic protocols. In *DIMACS Protocols Workshop*, 1997.

[4] Tuomas Aura. Strategies against replay attacks. In *Computer Security Foundations Workshop*, pages 29–68, 1997.

[5] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[6] L. Buttyán, S. Staamann, and U. Wilhelm. A simple logic for authentication protocol design. In *11th IEEE Computer Security Foundations Workshop*, pages 153–162. IEEE Computer Society Press, 1998.

[7] Ulf Carlsen. Optimal privacy and authentication on a portable communications system. *Operating systems review*, 28(3):16–23, 1994.

[8] John A Clark and Jeremy L Jacob. Searching for a solution: Engineering tradeoffs and the evolution of provable secure protocols. In *Proceedings 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2000.

[9] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

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

[11] Joshua D. Guttman. Security protocol design via authentication tests. In *Proceedings of 15th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, April 2002.

[12] Joshua D. Guttman and F. Javier Thayer. Authentication tests. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2000.

[13] D. Kindred and J.M. Wing. Theory generation for security protocols. *ACM TOPLAS*, July 1999.

[14] G. Lowe. A hierarchy of authentication specifications. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[15] D. O'Crualaoich and S.N. Foley. Theory generation for the simple logic. Technical report, University College Cork, 2002. In preparation.

[16] L. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. In *TR, Cambridge University, England*, 1998.

[17] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[18] A. Perrig and D.X. Song. Looking for diamonds in the desert: extending automatic protocol generation to three-party authenticatoin and key agreement protocols. In *Proceedings of 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000.

[19] A.W. Roscoe. Using intensional specifications of security protocols. In *Proceedings of the Computer Security Foundations Workshop*, pages 28–38. IEEE Press, 1996.

[20] D. X. Song. Athena: a new efficient automated checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999.

[21] P. Syverson. A taxonomy of replay attacks. In *Proceedings of IEEE Computer Security Foudations Workshop*, pages 187–191, 1994.

[22] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, 1998.

[23] T. Y. C. Woo and S. S. Lam. A lesson on authentication protocol design. *Operating System Review*, pages 24–37, 1994.

APPENDIX

A. MUTUAL AUTHENTICATION WITH TTP

A.1 Requirement Specification

Note that we express the public channel by the symbol Cp. The reason is that $P \triangleleft C(X)$ is the only terminal state which can appear in protocols. $P \triangleleft X$ is always translated to $P \triangleleft Cp(X)$ and $P \in r(C)$. We hold the similar assumptions about Cp in all specifications.

declarations {
Channel Cas, Cbs, Cp;
Principal A, B, S;
Nonce Na, Nb;
Message X;
Formula ϕ;
}
assumptions {
$A \models (w(Cas) = \{A, S\})$;
$S \models (w(Cas) = \{A, S\})$;
$B \models (w(Cbs) = \{B, S\})$;
$S \models (w(Cbs) = \{B, S\})$;
$A \in r(Cas)$; $S \in r(Cas)$;
$B \in r(Cbs)$; $S \in r(Cbs)$;
$A \in r(Cp)$; $A \in w(Cp)$;
$B \in r(Cp)$; $B \in w(Cp)$;
$S \in r(Cp)$; $S \in w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \models ((S \Vdash\!\sim \phi) \rightarrow (S \models \phi))$;
$B \models ((S \Vdash\!\sim \phi) \rightarrow (S \models \phi))$;
$A \models ((S \models (B \mid\!\sim X)) \rightarrow (B \mid\!\sim X))$;
$B \models ((S \models (A \mid\!\sim X)) \rightarrow (A \mid\!\sim X))$;
}
goals {
$A \models (B \Vdash\!\sim (A, Na))$; /* G_1*/
$B \models (A \Vdash\!\sim (B, Nb))$; /* G_2*/
}

A.2 Subprotocols for G_2

Subprotocol 2.1

$$\begin{array}{lll} B & , & \\ A & \triangleleft & C_p(B, Nb), \\ S & \triangleleft & C_{as}(B, Nb), \\ B & \triangleleft & C_{bs}(A \mid\!\sim (B, Nb)). \end{array}$$

Subprotocol 2.2.

$$\begin{array}{lll} B & , & \\ S & \triangleleft & C_{bs}(B, Nb), \\ A & \triangleleft & C_{as}(B, Nb), \\ B & \triangleleft & C_p(B, Nb). \end{array}$$

B. SAMPLE PROTOCOLS GENERATED

B.1 Mutual Authentication without TTP

B.1.1 Using symmetric keys

ASPB generated four correct protocols in 1.2 seconds.

Protocol 2.1.

$$\begin{array}{lll} Msg1 & A \rightarrow B: & \{A, Na\}_{Kab}, \\ Msg2 & B \rightarrow A: & \{Na, Nb\}_{K_{ab}}, \\ Msg3 & A \rightarrow B: & Nb. \end{array}$$

Protocol 2.2.

$$\begin{array}{lll} Msg1: & A \rightarrow B: & A, Na, \\ Msg2: & B \rightarrow A: & \{B, Na, Nb\}_{K_{ab}}, \\ Msg3: & A \rightarrow B: & Nb. \end{array}$$

Changing the last message of Protocols 2.1 and 2.2 to $\{Nb\}_{K_{ab}}$, results in the other two protocols that were generated.

B.1.2 Using signature keys

ASPB generated two correct protocols in 1.2 seconds.

Protocol 2.3.

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow A: & \{A, Na, Nb\}_{SK_b}, \\ Msg3 & A \rightarrow B: & \{B, Nb\}_{SK_a}. \end{array}$$

Protocol 2.4.

$$\begin{array}{lll} Msg1 & A \rightarrow B: & \{A, Na\}_{SK_a}, \\ Msg2 & B \rightarrow A: & \{A, Na, Nb\}_{SK_b}, \\ Msg3 & A \rightarrow B: & \{B, Nb\}_{SK_a}. \end{array}$$

B.2 Mutual Authentication and Key Exchange Protocols with TTP using symmetric keys

B.2.1 Four message protocols

Protocol 3.4.

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\ Msg3 & S \rightarrow A: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\ Msg4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Nb, K_{ab}\}_{K_{bs}}. \end{array}$$

Protocol 3.5.

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\ Msg3 & S \rightarrow A: & Nb, \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, K_{ab}\}_{K_{as}} \\ Msg4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}. \end{array}$$

Protocol 3.6.(BAN optimized Yahalom protocol [5])

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\ Msg3 & S \rightarrow A: & Nb, \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, K_{ab}, B\}_{K_{as}} \\ Msg4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Nb, K_{ab}\}_{K_{bs}}. \end{array}$$

Protocol 3.7.(Paulson amended Yahalom Protocol [16])

$$\begin{array}{lll} Msg1 & A \rightarrow B: & A, Na, \\ Msg2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\ Msg3 & S \rightarrow A: & Nb, \{A, B, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, K_{ab}\}_{K_{as}} \\ Msg4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, B, Nb, K_{ab}\}_{K_{bs}}. \end{array}$$

Syverson[21] found a flaw of Protocol 3.6, however, ASPB uses the following assumptions.

- A principal can recognize his own nonces of the running rounds, and refuse to use them as other principal's nonces.
- If a principal may understand a message context, the principal may distinguish the format of different components, such as principal name, nonce, key, etc.

By these assumptions, Protocol 3.6 is also a correct protocol.

B.2.2 Five Message Protocols

In this section, we select a further three five-message protocols from those that were generated by ASPB.

Protocol 4.4.

$$
\begin{array}{lll}
Msg1 & A \rightarrow B: & A, Na, \{A, Na\}_{K_{as}}, \\
Msg2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \{A, Na\}_{K_{as}}, \\
Msg3 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}}, \\
Msg4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Msg5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$

Protocol 4.5.

$$
\begin{array}{lll}
Msg1 & A \rightarrow B: & A, \{B, Na\}_{K_{as}}, \\
Msg2 & B \rightarrow S: & B, \{A, Nb\}_{K_{as}}, \{B, Na\}_{K_{bs}}, \\
Msg3 & S \rightarrow B: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Msg4 & B \rightarrow A: & Na, \{Na, Nb, K_{ab}\}_{K_{as}}, \\
Msg5 & A \rightarrow B: & Nb.
\end{array}
$$

Protocol 4.6.

$$
\begin{array}{lll}
Msg1 & A \rightarrow B: & A, \{B, Na\}_{K_{as}}, \\
Msg2 & B \rightarrow S: & B, \{A, Nb\}_{K_{as}}, \{B, Na\}_{K_{bs}}, \\
Msg3 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Msg4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Na, Nb, K_{ab}\}_{K_{as}}, \\
Msg5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$