Fast Automatic Security Protocol Generation

Hongbin Zhou Simon N. Foley*
Department of Computer Science
University College, Cork, Ireland
{zhou,s.foley}@cs.ucc.ie

Abstract

An automatic security protocol generator is described that uses logic-based heuristic 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.

1 Introduction

Security protocols are widely used in distributed systems for authentication, key exchange and other security requirements. Building security protocols can be a challenge since it is relatively easy to introduce subtle design-flaws that are difficult to find. While many approaches for *analyzing* security protocols have been developed [7, 13, 14, 28, 31, 36], relatively little work has been carried out on systematic approaches to the *design* and *implementation* of security protocols.

Abadi and Needham [4] set out ten principles to help designers avoid classes of known protocol flaws. The principles represent best practice and, however, are neither necessary nor sufficient: designers do not necessarily design new protocols by obeying only these principles. Formal design

*Corresponding Author

approaches for security protocols have also been proposed. A weakest-precondition-based approach is proposed in [5] that can be used to synthesise protocols from authentication goals. The Simple (BSW) logic [8] 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.

Guttman [16] proposes a manual protocol design methodology that is based on authentication tests [17]. For different goals, individual subprotocols are generated that are combined together to form the final protocol. It is a manual design process and relies on the skill of the protocol designer. Saidi [32] proposes a semi-automatic design tool based on BAN logic. It provides several commands for human intervention during the generation process. While it may simplify the protocol design process, it does not fully automatically genereate protocols, and still relies on the skill of the protocol designer.

We are interested in the automatic generation of security protocols from their protocol goals and assumptions. Existing research on this topic includes Clark and Jacob's evolutionary search [10, 12], Perrig and Song's Automatic Protocol Generator (APG) [30], and the authors' Automatic Security Protocol Builder (ASPB) [38]. All the above approaches automatically construct and 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, [12] uses the original inference rules of the BAN logic to systematically test whether candidate protocols uphold all protocol goals. A fitness function is used in their evolutionary approach to guide the application of BAN inference rules in a forward manner until a valid protocol is arrived. Their subsequent work [10] is based on the SVO logic [35], on the basis that the SVO logic is more suitable for describing and analysising key agreement protocols than the BAN logic.

Protocols that are generated using [10, 12] are implicitly verified, within their corresponding logic, to meet their goals. Perrig and Song's Automatic Protocol Generator (APG) [30] uses syntactic restriction rules to select random candidate protocols that are in turn checked for validity using the Athena [33] security protocol checker.

The above approaches can be regarded as generating candidate protocols in a *forward* manner: protocol messages are generated in the same order as their eventual execution in the protocol. Intuitively, a candidate first message is selected, along with its resulting inferences, and a recursive search for subsequent messages for the protocol. This forward search is bounded by a predefined maximum number of protocol messages, and terminates when the protocol goal is satisfied is met for the current sequence.

When a protocol fragment (an incomplete protocol) is generated in a forward manner, it is hard and potentially impractical, to determine whether the protocol fragment may result in a validate protocol which meets all protocol goals. Therefore, all generated protocol fragments should be kept before they extend to complete candidate protocols for further verification. It may be impractical to keep all generated protocol fragments especially for protocols that have more than four messages due to the large number of generated protocol fragments. For example, even with fitness functions [10,12] or cost functions [30], these forward searching approaches must verify a large number of unlikely protocols and may not meet all goals. Moreover, providing an accurate fitness function for a protocol is very difficult, and in some cases potentially impossible [10, 12].

In this paper we consider the automatic generation of security protocols from requirements. This paper is an extended and significantly revised version of [38], and makes a number of contributions. A technique for efficient protocol generation based on *backward* search is proposed. Heuristic rules—based on an extended version of the BSW logic [8] to provide a 'first cut' definition of protocol security—are used to guide the search for potentially secure protocols. Unlike the original BSW-logic, the BSW-ZF logic used in this paper supports reasoning about message secrecy, fresh channels, and component holding statements. The backward search approach results in an efficient search of the protocol space since the heuristics—derived from the logic axioms—guide the search and quickly eliminate regions of the search space than could not be synthesized from the protocol goals.

These heuristic rules generate sub-protocol fragments based on individual protocol goals and these are, in turn, composed into collections of candidate protocols using a novel strategy based on shortest common subsequence. The heuristics are also augmented by extending protocol selection/design strategies [16, 21, 22] in order to restrict the generated protocols to a collection of candidate protocols that are likely to be secure. While the collection of candidate protocols are secure under the BSW-ZF logic, the intention is that more sophisticated protocol analysis tools, such as [20, 23–25], are then used to select appropriate protocols from the collection.

The paper is organized as follows. The BSW-ZF logic, a modified and extended BSW logic, is outlined in Section 2. This section also describes how the synthesis rules are used as heuristics to guide the automatic backward search for protocols within the logic. The high-level architecture of ASPB is outlined in Section 3. Section 4 describes how the BSW-ZF heuristics are used to generate sub-protocols from single goals. The sub-protocols of multiple goals are composed into candidate protocols using the algorithm

described in Section 5. Section 6 outline a number of additional strategies used by ASPB to further refine and the candidate protocols to a collection of valid protocols. ASPB is evaluated and compared to other approaches in Section 7. Appendix A provides a complete list of the BSW-ZF Heuristic rules and Appendix B provides provides samples of generated protocol.

2 The BSW-ZF logic

The BSW-ZF logic improves and extends the BSW logic [8] to support reasoning about message secrecy, fresh channels, and 'holding' statements. While the inference rules of the BSW-ZF logic can be used to verify properties of security protocols [38], it is their corresponding heuristic rules that are used in ASPB as the core technique to guide the automatic backward search for candidate subprotocols from their goals. Therefore, for reasons of space we do not provide the inference rules of BSW-ZF as they can be inferred from the definitions of the heuristic rules, which are provided in Appendix A.

2.1 Syntax

The BSW-ZF logic uses abstract channels that are similar to the Spi Calculus [3] in order to represent keyed communication between principals. The capability to write into (for example, using the encryption key) and to read from (e.g., using the decryption key) a channel C is denoted by $w(C)$ and $r(C)$, respectively. The formula $P \ni r(C)$ means that principal P has the capability to receive messages from channel C, and correspondingly, $P \ni w(C)$ means that principal P has the capability to send messages to channel C.

Let $\sigma(X)$ denote the set of principals that share a secret X. The secret X can be a temporary secret that is held during several protocol steps, or a long term secret. We assume that the secret X is never leaked to any principal outside $\sigma(X)$, other than to those principals that are trusted by the members of $\sigma(X)$. For example, $P \models (\sigma(X) = \{P, Q\})$ means that P believes X is a secret shared between P, Q, and any third parties who (trusted by P and/or Q) may be privy to X. In this case, it is assumed that a trusted principal will neither use X as a proof of identity nor as a channel to communicate with. With this assumption, only two principals, P and Q, may use X as a proof of identity or as a channel to communicate with. When either of them receives X in a message, the principal may determine whether the message was sent by itself or by the other party. Thus, the message origin can be safely determined. Also note that this assumption

implies that principals trusted by other protocol participants will not attack the current protocol.

The set of principals that can receive and can send messages via a channel C is denoted by its reader set $\sigma(r(C))$ and its writer set $\sigma(w(C))$, respectively. For example, if Ω represents the set of all principals, then $\sigma(r(C)) = \Omega$ and $\sigma(w(C)) = \{P\}$ represents an authentic channel, whereby any principal can authenticate messages signed by the private key of principal P.

In the following, P,Q range over principals; C represents a channel; X represents a message which can be data or formulae or both; ϕ represents a formula. Data, such as principal identities, nonces, and read and write channel properties, are atomic messages that may be held by principals. The BSW-ZF logic uses the following basic formulae.

$P \triangleleft X$**:** Principal P sees message X. Someone has sent X via a channel that P can read.

$P \triangleleft C(X)$**:** P sees X on channel C. Someone has sent a message X via channel C. If P can not read C then P can not discover the contents of X.

$P \mid\!\sim X$**:** P once said X. P sent a message containing X at some point in the past. We do not know exactly when the message was sent.

$P \mid\!\mid\!\sim X$**:** 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 fresh nonces or messages sent using a fresh session key (channel).

$P \models \phi$**:** P believes that ϕ is true. It does not mean that ϕ is actually true, rather, P believes it.

$P \ni X$**:** P holds X, and therefore, may freely combine X with other messages and send the resulting combinations out to other principals. Unlike $P \triangleleft X$ requires that X is obtained by P during a round of protocol, $P \ni X$ can be obtained as an assumption before a round of protocol.

$P \mapsto X$**:** P generates X. This formula represents the origin of message X. The message X is used to represent constants or the atomic messages that are generated by the principal. Examples include nonces and principal identifiers.

Protocol steps, goals and assumptions are each expressed as formulae within the logic in the usual way. The logic also uses the conventional logic operators $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication) from propositional logic and some basic notation from set theory, such as $=$ (equality), and $\cup$ (union).

2.2 Inference Rules

To verify a security protocol, an idealised protocol and initial principal assumptions are initially given. The inference rules are used to guide the search from initial assumptions to further beliefs in the order of protocol execution. Belief logics verify whether expected protocol goals are covered by initial assumptions and these further derived beliefs.

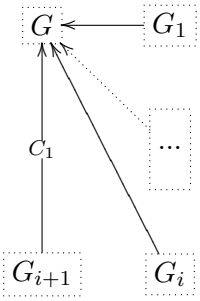
2.3 Heuristic Rules

To synthesise (design) a security protocol, only protocol goals and initial principal assumptions are given beforehand. A number of well-ordered protocol steps are derived from given protocol goals and given assumptions using a heuristic as template. Intuitively, the heuristic rules of the BSW-ZF logic are a more prescriptive form of the synthesis rules of the BSW logic, for the purposes of automation in ASPB.

While the inference rules of the BSW-ZF logic are used to verify security protocols, the BSW-ZF logic can also be used for synthesising security protocols. The heuristic rules can be used as synthesis rules to guide the *backward construction* for candidate subprotocols from their goals. In this case, the backward construction means that the protocol steps are constructed by the backward order of protocol execution. The reason for this is that the last required protocol step for a given protocol goal is naturally derived from applying heuristic rules to the protocol goal.

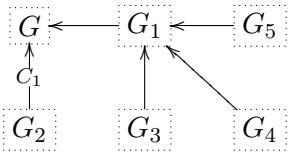
Our heuristic rules take the general form of a *rooted tree* of temporally ordered goals. A rooted tree is a directed acyclic graph (DAG) with a vertex

that is singled out as the root.



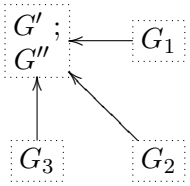
This is interpreted to mean that in order to reach the goal G, all subgoals G_1, ..., G_i, have to be established without any conditions or in any particular temporal order. Subgoal G_{i+1} is an optional conditional subgoal under condition C_1, with the interpretation that if condition C_1 is satisfied, then G_{i+1} has to be established before the goal G can be established. Otherwise, G_{i+1} does not need to be considered. Arcs indicate the temporal relationship between a goal and its subgoals.

In some scenarios, subgoals have to be established in a particular temporal order. For example, before a principal receives a message via a channel, there must be another principal who first sent the message in that channel. Therefore, we use the following form to illustrate the ordered subgoals.

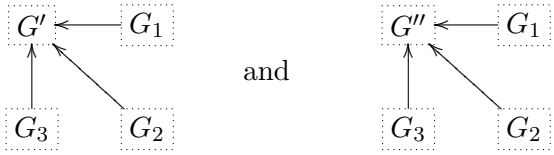


This is interpreted to mean that in order to establish the goal G, all subgoals G_1, ..., G_5, have to be reached in a particular temporal order. Here, G_3, G_4, and G_5 must be established before G_1. However, G_3, G_4, and G_5 do not need to be reached in any temporal order relative to each other.

The goals are expressed in the heuristic rules using a composed form $G'; G''$, as follows,

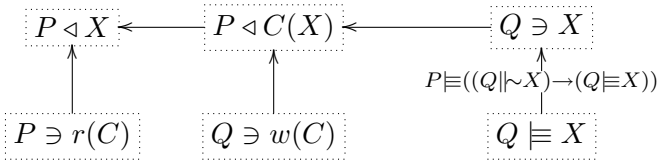


This is an abbreviated form of two heuristic rules and means that these rules have the same subgoals but different goals. They can be rewritten as



Rules **Heur1** and **Heur11** illustrate the heuristic rules of the BSW-ZF logic. Appendix A provides the full set of heuristic rules derived from the inference rules of the BSW-ZF logic.

Heur1 To see message X, P must receive X on channel C and be able to read C. Before P receives X from C, some principal Q should hold X and may write in channel C. In addition, if the honesty assumption is present, then Q is required to believe X before it can hold X.



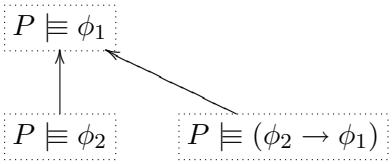
That $P \triangleleft X$ can be synthesised from $P \ni r(C)$ and $P \triangleleft C(X)$ is justified by the Seeing Axiom **S1** of the BSW-ZF logic [38]:

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

For the purposes of this paper we do not consider the BSW-ZF Axioms, other than to note that they were used to inform our selection of the heuristic rules **Heur1** through **Heur16** (Appendix A).

Note that a heuristic (more general than **Heur1**) that is directly based on the S1 rule—whereby $P \triangleleft X$ is synthesized from $P \ni r(C)$ and $P \triangleleft C(X)$— is inadequate in practice. In particular, when there are no given protocols, the origin of the message $C(X)$ is unknown. Therefore, we require that some principal Q holds X and that Q may write in C *before* P may receive X from channel C, as reflected in Heuristic **Heur1**. This is similar in intent to the 'required precondition' that is used in [5].

Heur11 To believe ϕ_1, P must believe ϕ_2 and $\phi_2 \rightarrow \phi_1$.



3 ASPB Architecture

Figure 1 outlines the architecture of the Automatic Security Protocol Builder (ASBP). A protocol specification is parsed and decomposed into a series of single goal protocol requirements using the Specification Parser. From these single goal protocol requirements, a collection of subprotocols are synthesised to satisfy the individual goals using the Single Goal Synthesiser (Section 4). The Protocol Composer (Section 5) merges these subprotocols to form complete candidate protocols. The Protocol Selector (Section 6.3) selects what are considered the most suitable protocols from the candidate protocols.

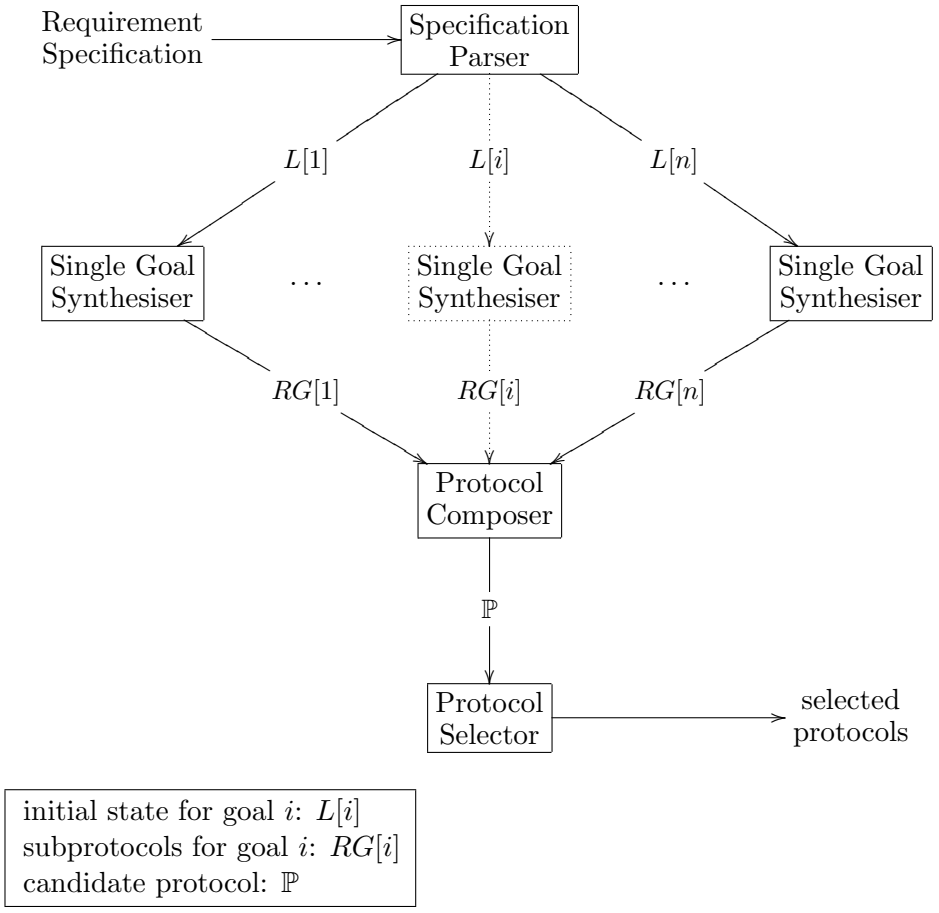


Figure 1: Overview of Automatic Security Protocol Builder

When designing a protocol, the designer should understand the security

context of the principals. It reflects the keys that principals know, the trust relationships between principals, and any other assumptions that they may hold. Our protocol requirement specification defines the terms that are used in the specification, initial known assumptions, and goals for the protocol to be designed. These are represented using the BSW-ZF logic.

Example 1 Figure 2 gives a complete requirement specification for a mutual authentication protocol using symmetric keys with a Trusted Third Party (TTP). Note that *unreachable assumptions* may also be included in the requirement specification. These do not affect the logic, but are used to direct the synthesis during pruning, and will be explained in Section 4.1. △

The Requirement Specification Parser parses the specification and checks whether all the formulae are valid based on the logic syntax and the types defined in the 'declarations' section of the specification. For each goal to be proven, the parser generates a single-node (root) formula tree (of that goal) as an initial synthesis state. This single-node rooted tree is regarded as a *initial formula tree* for synthesis, such as Figure 3(a). These provide the inputs to the Single Goal Synthesiser.

4 The Single Goal Synthesiser

The Single Goal Synthesiser accepts an initial synthesis state from the Requirement Specification Parser, automatically synthesises its goal using the heuristic rules of the BSW-ZF logic, and builds complete formula trees from the goal. The BSW-ZF heuristic rules define how this tree may be expanded, and a snapshot of this tree (rooted at that goal), represents a synthesis state. The single (root) node of an initial formula tree is the goal that any synthesised protocol must uphold.

Since the heuristic rules of the BSW-ZF logic are also defined in terms of rooted trees, the synthesis of a one-way authentication protocol can be illustrated by *tree grafting*, whereby instantiations of suitable heuristic rules are grafted into *incomplete formula trees* that are extended from the initial formula tree until a *complete formula tree* is held. An incomplete formula tree has one or more leaves that are not assumptions from the requirement specification, such as (a), (b) and (c) in Figure 3. All the leaves of a complete formula tree correspond to assumptions from the requirement specification, such as Figure 3 (d).

If a protocol goal is not an assumption from the requirement specification, then the corresponding initial formula tree is an incomplete formula

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

Figure 2: A complete requirement specification for mutual authentication protocols using symmetric keys with TTP

tree. If a protocol goal is an assumption from the requirement specification, then the corresponding initial formula tree is a complete formula tree. The fact that an initial formula tree can be a complete formula tree, reflects that complete formula trees may contain no protocol steps, since the corresponding protocol goals do not require the cooperation of principals.

By distinguishing complete formula trees from incomplete complete formula trees, the end-point of a synthesis process is determinable. More specifically, the synthesis for the root of the complete formula tree is finished, when a complete formula tree is generated. Figure 3(b) and (c) describe two grafting steps from initial formula trees. Algorithm 1 describes this process.

Algorithm 1 StateSet syn(State *initState*)

```
State s;
StateSet S', R = φ;
StateSet L = {initState};
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 synthesis state s from the set of current states L.

If the given state s is an incomplete formula tree, then operation *subsyn*(s) selects a leaf node that is not an assumption, and applies the currently applicable heuristic rules to this leaf node. A number of new formula trees are generated by appending the subgoals of the applicable heuristic rules to the formula tree. Operation *subsyn*(s) returns these new formula trees as a collection of synthesis states. For example, the different formula trees in Figures 3(b) and (c) could be generated and returned by *subsyn*(s) for the formula tree in Figure 3(a).

If no heuristic rule can be applied to a leaf node (terminal subgoal) then the Single Goal Synthesiser tests whether this leaf matches an assumption.

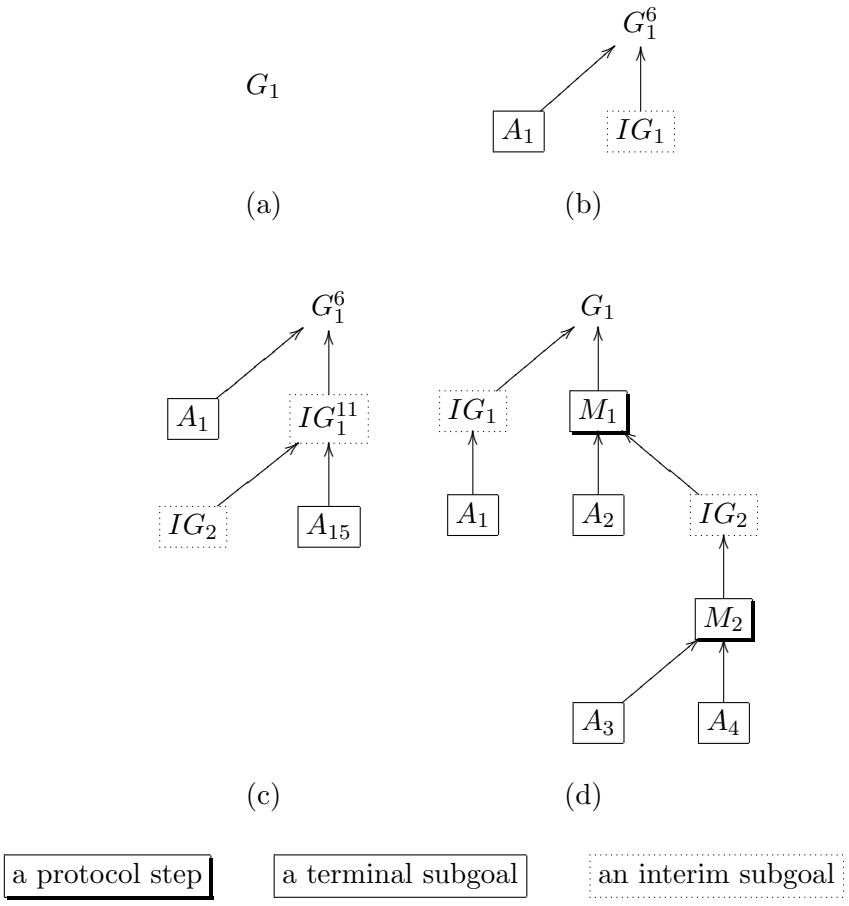


Figure 3: Incomplete and Complete Formula Trees

If so, then it is considered reachable. If all leaves of a tree are reachable, then the tree is a complete formula tree, and represents a candidate protocol for the goal. It is added to the set of complete formula trees R for the given goal.

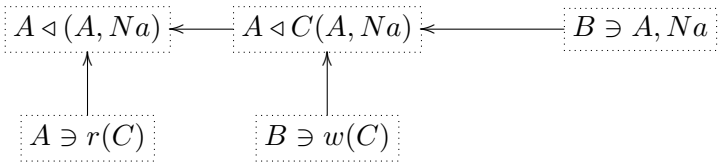
An automatic verification tool [26] for the original BSW Logic has been implemented using Theory Generation [18]. This tool [26] also supports (manually guided) synthesis of protocols using the synthesis rules described in [8]. The Single Goal Synthesiser described in this section builds on and extends this manual tool in [26] by automatically carrying out the synthesis process for the BSW-ZF logic.

4.1 Improving Performance

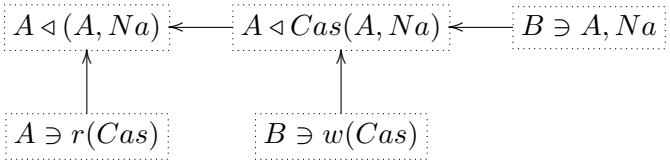
To improve the performance of the Single Goal Synthesiser, a number of ad-hoc strategies are used as part of the operation *subsyn*(s), including early pruning, variable instantiation, and tree pruning.

Early Pruning. Having applied a heuristic rule, consider generated formulae that are of the form of $P \ni r(C)$, $P \ni w(C)$, or $P \models\!\!\sharp(Y)$. To speed up the judgement process, the operation $subsyn(s)$ checks whether leaves of this form in generated formula trees match the initial assumptions or specify unreachable assumptions of the requirement specification. If a leaf matches an assumption, then the leaf is reachable and the search terminates at this point. For example, $A \ni r(Cp)$ and $B \ni w(Cp)$ in Figure 4(d) match specified assumptions, then the search terminates at this point. Only $B \ni (A, Na)$ remains for further synthesis. If any leaf matches a specified unreachable assumption, then the corresponding formula tree is also unreachable, and the formula tree is simply discarded. For example, if $B \ni w(Cas)$ in Figure 4(b) or $A \ni r(Cbs)$ in Figure 4(c) match specified unreachable assumptions, then the corresponding formula trees are simply discarded. Otherwise, the formula tree is kept for further synthesis. The rationale for early pruning is that formulae of this form are frequently generated using the heuristic rules, such as **Heur1**, **Heur5**–**10**, and **Heur12**–**14**, and it is straightforward to immediately determine whether they are reachable by this approach.

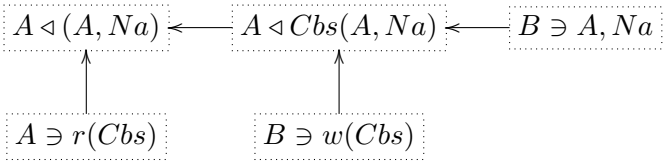
Variable Instantiation. Some heuristic rules introduce new variables that do not appear in the goal, but appear in the subgoals. For example, **Heur1**, and **Heur6**–**10**. If the type of the variable is known (for example, variable Q is a principal, and variable C is a channel in **Heur1**),



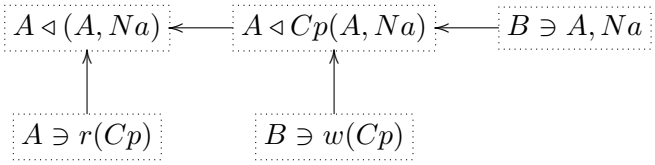
(a) Applying **Heur1** on $A \triangleleft (A, Na)$ with variable Q instantiated as B



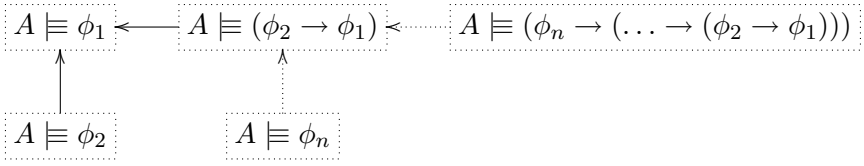
(b) Instantiated variable C by Cas



(c) Instantiated variable C by Cbs



(e) Instantiated variable C by Cp



(d) A Formula Tree for Recursively Applying **Heur11** on $A \models \phi_1$

Figure 4: Examples for Early Pruning and Variable Instantiation

then the heuristic rule is repeatedly applied for all possible instantiations of the variable from the 'declarations' section of the requirement specification. As a result of this, a number of independent states that have no variables are generated. For example, Figure 4(a) introduces channel variable C. Assuming channel declarations Cas, Cbc and Cp and applying all possible instantiations of variable C in the example in Figure 4 then variable-free independent states Figure 4(b), (c), and (d) are generated. Operation $subsyn(s)$ then applies the 'early pruning' strategy to these instantiated states.

Some heuristic rules introduce variables of unknown type. For example, in theory, variable ϕ_2 in **Heur11** could be bound to any logical formula, resulting in an infinite state space. In practice, we consider that in order to reach **Heur11**'s goal, both of its two subgoals must be reached beforehand. In order to determine whether **Heur11** is applicable to $P \models \phi_1$, where ϕ_1 has no unbound variables, a search is made for a pattern $P \models (\phi_2 \rightarrow \phi_1)$ for some arbitrary ϕ_2 in the assumptions of the requirement specification. If matched, the heuristic rule is applied. If no match occurs, then a search is done to test if it is possible to synthesise a formula of the form $P \models (\phi_2 \rightarrow \phi_1)$, from existing assumptions in the requirement specification. In practice, this search is relatively straightforward as only **Heur11** can be applied to $P \models (\phi_2 \rightarrow \phi_1)$, whereupon it recursively searches for formulae of the form $P \models (...(... \rightarrow \phi_1))$ in assumptions, such as the formula tree in Figure 4(d).

To further minimise the search space, **Heur11** disregards higher order belief formulae [7] of the form $P \models (Q \mid\!\sim (Q \mid\!\sim X))$ and $P \models (Q \models (Q \models X))$, etc., on the basis that they do not provide any additional information than the first order beliefs $P \models (Q \mid\!\sim X)$ and $P \models (Q \models X)$, etc.. With these strategies, we restrict the state search space, and generate a wide range of useful protocols.

Tree Pruning. The Single Goal Synthesiser does not allow any formula as its own direct or indirect parent in a formula tree. The reason for this is that the Single Goal Synthesiser terminates the extension for any branch of a formula tree as early as possible. If a goal can be achieved, then the simplest subtree does not involve itself as a subgoal. Otherwise it involves an infinite search for this goal, since any branch for this goal in a formula tree requires itself as a subgoal, such as the formula tree in Figure 5(c). For a goal to have itself as its own

direct or indirect parent in a formula tree, the high level subtree for this goal can be safely replaced by the low level subtree for the same goal. The replacing formula tree is already in the state set since the Single Goal Synthesiser extends a formula to all possible formula trees without any formula as its own direct or indirect parent. For example, formula trees in Figure 5(a), (b) can be simplified to the formula tree in Figure 5(d).

Example 2 Two subprotocols are automatically synthesised from Goal G_1 in the requirement specification of Figure 2. Subprotocol 1.1 corresponds to the protocol messages from the search tree in Figure 4. Further synthesis generates an additional search tree with corresponding Subprotocol 1.2.

Subprotocol 1.1	**Subprotocol 1.2**
A,	A,
$B \triangleleft C_p(A, Na)$,	$S \triangleleft C_{as}(A, Na)$,
$S \triangleleft C_{bs}(A, Na)$,	$B \triangleleft C_{bs}(A, Na)$,
$A \triangleleft C_{as}(B \mid\sim (A, Na))$.	$A \triangleleft C_p(A, Na)$.

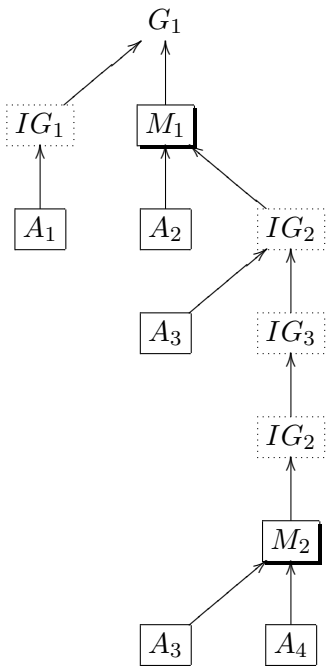
Note that the first line of a subprotocol indicates the subprotocol initiating principal (initiator) that generates the first message of the subprotocol. The synthesis of the symmetrically similar goal G_2 generates two similar search trees from which symmetrically similar subprotocols 2.1 and 2.2 are obtained.

Subprotocol 2.1	**Subprotocol 2.2**
B,	B,
$A \triangleleft C_p(B, Nb)$,	$S \triangleleft C_{bs}(B, Nb)$,
$S \triangleleft C_{as}(B, Nb)$,	$A \triangleleft C_{as}(B, Nb)$,
$B \triangleleft C_{bs}(A \mid\sim (B, Nb))$.	$B \triangleleft C_p(B, Nb)$.

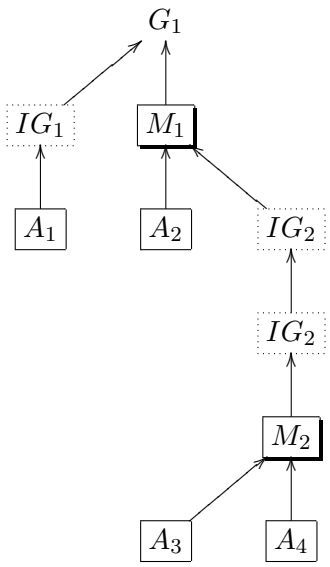
△

5 The Protocol Composer

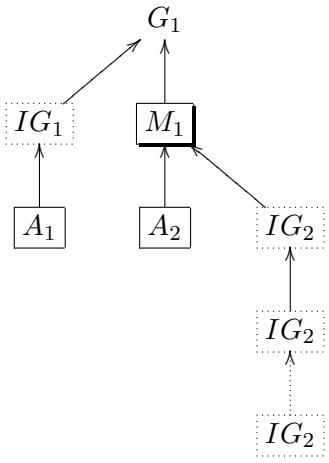
The Single Goal Synthesiser is used to generate subprotocols from a single goal. While not considered in the original paper [8], it is possible, in theory, to use the original BSW synthesis rules to synthesise multiple goals. In the same way, the heuristic rules of the BSW-ZF logic could be used to synthesize a protocol from multiple goals. However, in practice, building a search



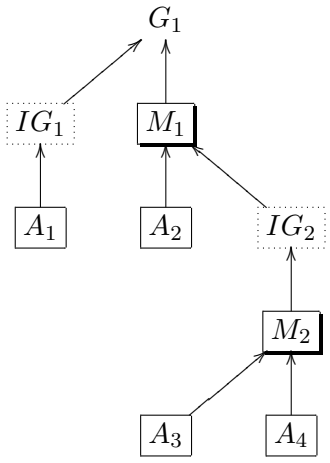
(a) Subgoal IG_2 as Its Own Indirect Parent



(b) Subgoal IG_2 as Its Own Direct Parent



(c) Infinite Search for Subgoal IG_2



(d) No Subgoal as Its Own Parent

a protocol step a terminal subgoal an interim subgoal

Figure 5: Self-Parent Formula Trees and Infinite 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 first synthesising only single goal protocols and it then uses the Protocol Composer to, in turn, merge these single goal protocols into a single candidate protocol that meets the composition of goals.

A feasible candidate protocol for a requirement specification must achieve all of the goals described in the requirement specification, otherwise the candidate protocol does not satisfy the requirement specification (within the logic). For this reason, the Protocol Composer generates a feasible candidate protocol by composing a number of subprotocols. Each of these subprotocols is selected from the generated subprotocol set for each goal in the requirement specification, and the composition of these subprotocols satisfies all the goals. There can be many possible ways to merge subprotocols. The easiest way to merge two subprotocols is to append one to the end of the other. However, this may lead to lengthy and inefficient protocols. In addition, such protocols are certainly not fail-stop [15], since an attacker may run part of the protocol (achieving one goal) without completing the other subprotocol.

5.1 Merging Principal sequences

A security protocol is a sequence of messages exchanged between principals in order to achieve a number of security goals. At an abstract level, these message exchanges can be described just in terms of a *principal sequence*. A principal sequence is a sequence of principal identities based on the order of message exchanges between the principals in a protocol. For example, Subprotocol 1.1 implementing goal G_1 (Example 2) has principal sequence $A \rightarrow B \rightarrow S \rightarrow A$; Subprotocol 2.2 implementing goal G_2 has principal sequence $B \rightarrow S \rightarrow A \rightarrow B$.

Given principal sequences X and Y, then we say that sequence X *covers* sequence Y if Y appears as a fragmented subsequence of X. For example, Figure 6(a) illustrates that $A \rightarrow B \rightarrow S \rightarrow A \rightarrow B$ covers $A \rightarrow S \rightarrow A$. The Protocol Composer uses principal sequences to guide the construction of new protocols when merging subprotocol messages.

Given a collection of subprotocols $\mathbb{P}_1 \ldots \mathbb{P}_n$ that meet goals $G_1 \ldots G_n$ of a protocol specification, respectively, the Protocol Composer tests whether a given principal sequence covers the principal sequences of the subprotocols $\mathbb{P}_1 \ldots \mathbb{P}_n$. This is done using a variation of the shortest subsequence algorithm [19]. If the given principal sequence covers the principal se-

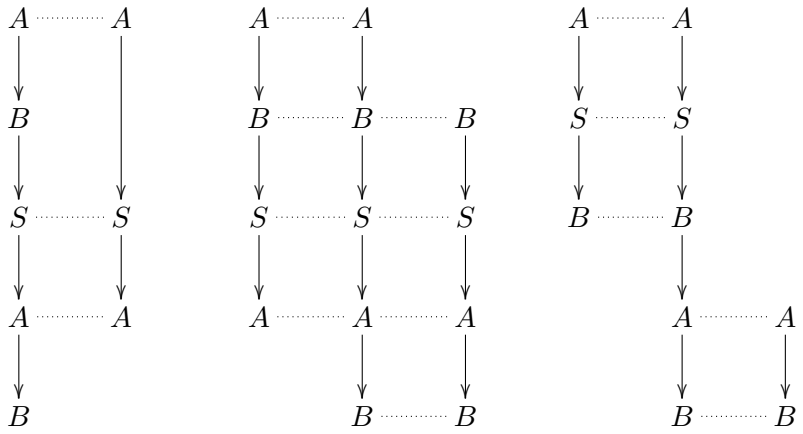


Figure 6: Sequence Covering Examples

quences of the subprotocols $\mathbb{P}_1 \ldots \mathbb{P}_n$, then it is a possible principal sequence of a candidate protocol. For example, in Figure 6(b), principal sequence $A \rightarrow B \rightarrow S \rightarrow A \rightarrow B$ covers both principal sequence $A \rightarrow B \rightarrow S \rightarrow A$ for Subprotocol 1.1 and principal sequence $B \rightarrow S \rightarrow A \rightarrow B$ for Subprotocol 2.2, then principal sequence $A \rightarrow B \rightarrow S \rightarrow A \rightarrow B$ is a possible principal sequence of a candidate protocol that satisfies both G_1 and G_2. The candidate protocol is generated by merging the messages of the individual subprotocols according to the generated (covering) principal sequence. The behaviour of the Protocol Composer is specified by Algorithm 2.

The Protocol Composer accepts two parameters: the protocol initiator *prin* and an array RG. Each element $RG[i]$ of RG is the state set for the ith goal of the given requirement specification. The state sets are generated by the Single Goal Synthesiser and correspond to the possible subprotocols for a given goal. The Protocol Composer generates candidate protocols based on the rule of shorter first, so that principal sequences (protocol patterns) are generated from the shortest to the longest.

Operation $sPattern(RG)$ returns the shortest length of all the possible candidate protocols in RG. If a candidate protocol is composed of a number of subprotocols, then the principal sequencing of the candidate protocol must cover all the principal sequences of these subprotocols. Thus, the candidate protocol is not shorter than the longest of these subprotocols. Note

Algorithm 2 ProtocolSet compose(Principal *prin*, Array *RG*)

```
Array RS = φ;
ProtocolSet ℙ′, ℙ = φ;
PatternSet T; Pattern t;
for i = sPattern(RG) to lPattern(RG) do
  T = genPatterns(prin, i);
  while ¬ empty(T) do
    t = choose(T);
    T = T \ t;
    RS = match(t, RG);
    ℙ′ = subCompose(t, RS);
    ℙ = add(ℙ, ℙ′);
  end while
end for
return P;
```

that the shortest pattern only guarantees that possible candidate protocols are not shorter than the shortest pattern. It does not means that a candidate protocol as the length of given shortest pattern can be generated. The reason for this is that the principal sequence of a candidate protocol is a common principal sequence that covers the principal sequences of its composing subprotocols. If the principal sequence of any subprotocol may not cover all other subprotocols, then a common principal sequence should be longer than all of its composing subsequences. For example, the length of all subprotocol principal sequences in Example 2 is 4, then the principal sequence length of a candidate protocol may not be shorter than 4. Otherwise, the principal sequence of candidate protocols may not cover any subprotocol. It follows that no protocol can be composed by given principal sequence length and subprotocols. On the other hand, the length of the shortest principal sequence that may cover subprotocols for all goals is 5. Figure 6(b) presents a possible covering example. This principal sequence length is longer than the length returned by operation $sPattern(RG)$.

Operation $lPattern(RG)$ returns the longest possible pattern length of all the possible candidate protocols in RG. Given n goals, where l_i is the length of the longest subprotocol for the ith goal, then the longest pattern length is $\sum_{i=0}^{n}(l_i - 1) + 1$. The motivation here is to generate a category of protocols such that the message receiver of a protocol step is the message sender of the next protocol step. Since the first line of a subprotocol only indicates the subprotocol initiator that generates the first message of

the subprotocol, when merging subprotocols into a candidate protocol, all of the subprotocol initiators should be a receiver of the previous protocol step, except for one initiator that is the candidate protocol initiator. The length of the longest pattern of a candidate protocol should be the sum of the number of steps of all the subprotocols plus the candidate protocol initiator. Note that the number of steps in a subprotocol is the length of the associated principal sequence minus one (which is used to indicate the subprotocol initiator). For example, principal sequence of the given candidate protocol in Figure 6(c) is $A \rightarrow S \rightarrow B \rightarrow A \rightarrow B$. It covers its subprotocol principal sequences $A \rightarrow S \rightarrow B$ and $A \rightarrow B$. If a candidate protocol is composed as Figure 6(c) by the subprotocols corresponding to the given subprotocol principal sequences. No message is sent at the step $B \rightarrow A$ of the generated candidate protocol. Therefore the last step $A \rightarrow B$ of the generated candidate protocol can be executed without executing the previous protocol steps. In this case, it is possible for an intruder to generate and send a message to B in this protocol step without participating previous protocol steps of the same round.

Operation $genPatterns(prin, i)$ generates all of the principal sequences, that are initiated by principal $prin$ and with length i, as possible protocol patterns.

Operation $match(t, RG)$ returns an array RS such that each element $RS[i]$ is a subset of the corresponding state set $RG[i]$ that is covered by the principal sequences of all $RS[i]$'s states that are, in turn, covered by the given principal sequence t. This is done by using the longest common subsequence algorithm [19].

Operation $subCompose(t,RS)$ returns all possible candidate protocols that follow principal sequence t, composed from the subprotocols of array RS. A candidate protocol is generated by composing combinations of subprotocols from $R[i]$ ($i \in [0, \ldots, n]$).

If a goal G_1 is a subgoal of another goal G_2 in the same requirement specification then G_1 is considered *relevant* to G_2. In this case, it is not necessary to further synthesise G_1 as doing so will simply generate subprotocols that will also be generated as parts of the subprotocols for G_2.

5.2 Merging Subprotocols

The Protocol Composer merges subprotocols according to the following rules.

Merg1 (Early appearing rule) A message from a subprotocol $\mathbb{P}_i$ should

appear in the candidate protocol $\mathbb{P}$ as early as possible, constrained only by the principal sequencings.

Merg2 By the inference rule **S2** of the BSW-ZF logic

$$\frac{P \lhd (X, Y)}{P \lhd X, P \lhd Y}$$

we get the following rule

$$\frac{P \lhd X, P \lhd Y}{P \lhd (X, Y)}$$

which means that the messages, that are received by P from the different subprotocols, may be composed by one message in the final protocol.

Merg3 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 message $C_1(X_3), C_2(X_2, Y_2)$ from two subprotocols merge into a resulting message $C_1(X_1, X_3), C_2(X_2, X_2, Y_1, Y_2)$.

Merg4 (Reducing rule) Any redundant message components should be reduced. For example, message $C(X_2, X_2)$ should be reduced to $C(X_2)$.

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 (Example 2). Thus there are 2×2 possible combinations of the subprotocols 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 that corresponds to the merge of subprotocols 1.1 and 2.2 from Example 2 (in 3.1 seconds):

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

$\triangle$

6 Further Refining the Candidate Protocols

6.1 Realising Idealised protocols

The Single Goal Synthesiser generates messages expressed as formulae within the BSW-ZF logic. The final implementation of these protocols is given in terms of conventional protocol message steps. For example, both message $Cas(A \mid\sim Na)$ and $Cas(A \triangleleft Na)$ are expressed by the same notation $Cas(A, Na)$. To minimise 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 [6]. For example, $Cas(A, Na, 0)$ and $Cas(A, Na, 1)$ or $Cas(A, Na)$ and $Cas(Na, A)$. In ASPB, we distinguish 'similar' messages by the latter approach, that is, exchanging message component order.

Since the message receiver of a protocol step is the message sender of the next protocol step, the above protocol in Example 3 can be rewritten in the following format:

$$\begin{aligned} A \rightarrow B: &\quad C_p(A, Na), \\ B \rightarrow S: &\quad C_{bs}(A, B, Na, Nb), \\ S \rightarrow A: &\quad C_{as}(B, Na, Nb, A), \\ A \rightarrow B: &\quad C_p(Nb, B) \end{aligned}$$

Each protocol step means that a principal sends a message and another principal receives this message. For example, the first step of the above protocol $A \rightarrow B : C_p(A, Na)$ means that $A \mid\sim C_p(A, Na)$ and $B \triangleleft C_p(A, Na)$.

6.2 Removing Redundant Components

The Protocol Composer may use a redundancy removing strategy to further remove redundant components of protocol messages. This is inspired by Mao's protocol idealisation process [22], that is used to transform protocol messages into BAN-like formulae via a context-sensitive syntactic analysis of the protocol syntax. Our redundancy removing process is used to transform BSW-ZF formulae into protocol messages using a similar analysis of the protocol syntax as follows.

Let the *relevant principal set* $\mathcal{N}_P(N)$ represent a set of principals to which principal P currently believes that the fresh nonce N is uttered as a reference. P should remember this reference wherever P sees N. At the beginning of a protocol, the relevant principal sets for all principals are initialised to empty.

The Protocol Composer uses the following relevancy rules to calculate the relevant principal set for every principal at each protocol step.

RR1 If P believes that only P and Q may write in channel C, and P sees nonce N together with principal R from C, then P believes that N is uttered as a reference to the principals P, Q, and R.

$$\frac{P \models (s(w(C)) = \{P, Q\}), P \ni r(C), P \triangleleft C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{P, Q, R\}}$$

RR2 If P believes that only Q may write in channel C, and P sees nonce N with principal R from C, then P believes that N is uttered as a reference to the principals Q and R.

$$\frac{P \models (s(w(C)) = \{Q\}), P \ni r(C), P \triangleleft C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{Q, R\}}$$

RR3 If P believes that only P and Q may read from channel C, and P writes nonce N with principal R in C, then then P believes that N is uttered as a reference to the principals P, Q, and R.

$$\frac{P \models (s(r(C)) = \{P, Q\}), P \ni w(C), P \mid\sim C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{P, Q, R\}}$$

RR4 If P believes that only Q may read from channel C, and P writes nonce N with principal R in C, then then P believes that N is uttered as a reference to the principals Q and R.

$$\frac{P \models (s(r(C)) = \{Q\}), P \ni w(C), P \mid\sim C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{Q, R\}}$$

Generally, if $\mathcal{N}_P(N) \cap \mathcal{N}_Q(N) = \psi_1$ before a protocol step $P \rightarrow Q : (N, \psi_2)$, then the protocol step is rewritten by $P \rightarrow Q : C(N, \psi_2/\psi_1)$ This is done in reverse order of protocol execution. If a message may not be distinguished from other messages after applying a relevancy rule, then further principal identifiers from ψ_2 are kept in that message, until the message can be distinguished from other messages. In addition, if $\psi_2/\psi_1 = \{\}$ in the first encrypted message of a protocol, then at least one principal identity of that message is kept to stop principals misusing that message. If we remove all principal identities from the first encrypted message, the protocol may subject to reflection/oracle attacks. The reason for this is that the message

receiver may not judge who generates this message (another principal in the current protocol run, or itself in a previous run). This is because it is not practical for principals to keep messages from previous protocol runs. An example that illustrates this problem is given in Appendix B.

Example 4 The messages in the generated protocol from Example 3 are reduced and then simplified using above rules to give:

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow S: & \{A, Na, Nb\}_{K_{bs}}, \\ Message\ 3 & S \rightarrow A: & \{Na, Nb, B\}_{K_{as}}, \\ Message\ 4 & A \rightarrow B: & Nb. \end{array}$$

△

6.3 The Protocol Selector

In general, given subprotocols $\mathbb{P}_1$ and $\mathbb{P}_2$ that uphold goals G_1 and G_2, respectively, then the monotonicity of the BSW-ZF logic ensures that the resulting merged candidate protocol as outlined above also upholds the goals G_1 and G_2 within the logic. Therefore, all the protocols generated by the Protocol Composer are valid within our logic.

However, belief logics do have weaknesses. Regardless of whether we deal with type flaw attacks by assuming that component types can be recognised by principals, the logic is still vulnerable to other classes of attacks. For example, the Protocol Composer generates the following simple mutual authentication protocol.

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow A: & B, Nb, \{A, Na\}_{Kab} \\ Message\ 3 & A \rightarrow B: & \{B, Nb\}_{Kab} \end{array}$$

While secure within the BSW-ZF and many other belief logics, this protocol is subject to a reflection/oracle attack:

$$\begin{array}{lll} Message\ 1 & A \rightarrow B & : A, Na, \\ Message\ 2 & B \rightarrow I(A) & : B, Nb, \{A, Na\}_{Kab}, \\ Message\ 2' & I(B) \rightarrow A & : B, Nb', \{A, Na\}_{Kab}, \\ Message\ 3 & A \rightarrow B & : \{B, Nb'\}_{Kab}. \end{array}$$

Here, Nb' is generated by the intruder. There are many examples of secure protocols, which when composed are vulnerable to attack [37]. If the unsuitable protocol is identifiable then it can be discarded following verification by the Protocol Selector. For example, when a subprotocol $\mathbb{P}_1$ does not merge any message with any other subprotocols in the merging process, we consider that the resulting candidate protocol is an unsuitable protocol and discarded by the Protocol Selector. The reason for this is that the subprotocol $\mathbb{P}_1$ could be executed independently. An intruder is possible to participate the subprotocol without participating the rest protocol of the same round.

It is useful to further consider verification of additional ad-hoc properties. For example, the non-injective agreement property [21]: *"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."* The generated protocols could be re-analysed using more sophisticated protocol analysis tools, such as the NRL Analyzer [23], the Interrogator model [24], FDR [20], Murφ [25], Athena [33]. In this case, the Protocol Selector of ASPB would be used to narrow down the set of candidate protocols to be verified. However, we point out that many existing protocol checkers are also limited and also include ad-hoc strategies, they do not guarantee the correctness of verified protocols.

We also suggest that practical techniques such as [6] 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.

7 Discussion and Evaluation

In this section, we evaluate ASPB and compare it with existing approaches, including, the Automatic Protocol Generator (APG) [30], and the Evolutionary approach [10].

Table 1 provides a time performance comparison between ASPB and APG [30]. The first three rows give the performance results for generating the mutual authentication protocol without TTP described in Sec B (and specified in Figures 7, 8, and 9). The fourth row gives the performance results for generating the mutual authentication protocol with TTP described in Figure 2. The other experiment was for mutual authentication and key agreement with TTP described in Figure 10. The fifth row gives the result

Table 1: The time performance comparison between ASPB and APG

protocol purpose		ASPB		APG[a]
		Stage 1[b]	Stage 2[c]	
mutual authentication without TTP	signature keys	1.2 sec.	1.3 sec.	N/A
	public keys	1.2 sec.	1.3 sec.	23 sec.
	symmetric keys	1.2 sec.	1.3 sec.	10 sec.
mutual authentication with TTP	symmetric keys	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 a 400MHz Intel Pentium III [30].

[b] Time to synthesise, compose, and generate all candidate protocols running on a 1.8GHz Intel Pentium IV.

[c] Estimated time that the Athena [33] checker would take to further validate the ASPB generated candidate protocols.

for four message protocols. The last row gives the result for five message protocols. The results of similar experiments were reported in [30]. While ASPB was tested on a faster computer than that used to test APG, given the marked difference in speed, it is nevertheless reasonable to conclude that ASPB runs significantly faster than APG.

ASPB generates approximately 500 valid five-message candidate protocols in 25 seconds. However, on manual inspection, many of these protocols are similar, containing minor textual and redundant variations. On manual inspection, we estimate that in this set there are 24 reasonably distinct four-message candidate protocols and 76 reasonably distinct five-message protocols. We selected several protocols from each category to demonstrate our results in Section B.

APG has not been tested for five message protocols; in this case we conjecture that direct application of the forward search approach of APG would result in a very large and potentially infeasible search space. Perrig and Song's estimate [30] is based on the average number of messages that a principal can generate in a given round of the protocol and is explained as follows. In a given principal sequence four-message three-party authentication and key agreement protocol (as the requirement described in Figure 10 and, for example, the given principal sequence is $A \rightarrow B \rightarrow S \rightarrow B \rightarrow A$), A generates over 100 different messages to B, then B in turn generates about

500 different messages and sends them to S. S can generate 30,000 messages and sends to B. B can generate around 500 messages in the final round. The combination of this number of messages is on the order of 10^{12}. If these four-message protocols extend to five-message versions by one message that is sent from A to B, there are around 500 different messages generated by A when using Perrig and Song's estimation approach in [30]. We conjecture that the search space for a given principal sequence five-message three-party authentication and key agreement protocol is over 500×10^{12} The search space for all possible principal sequences is $500 \times 10^{12} \times 2^5$ (based on the limitation that one does not send messages to itself, and therefore there are 2 possible message receivers at each protocol step).

APG [30] generates the 'best' protocol for each protocol requirement in the entire protocol search space with respect to its metric functions. The metric function of APG [30] gives each message operation a cost value. For example, the cost value of a message decryption/encryption is 3; the cost value of generating a nonce is 1. The cost value of a protocol is the total cost of all the protocol operations. The minimal cost protocol that meets all of the protocol goals is APG's 'best' protocol. Compared with APG, the evolutionary approach [10] only searches a part of the entire protocol space. Its goal is to find a 'good' solution for each protocol requirement in a given protocol search space with respect to its fitness function. Unlike APG, the evolutionary approach does not guarantee that the 'best' protocol in the entire search space can be found in the selected part of the search space. On the other hand, because the evolutionary approach searches a smaller protocol search space than APG, it may find a 'good' protocol in a shorter time than APG. For example, APG generates the 'best' protocol for mutual authentication and key exchange protocol in two hours; Chen, Clark, and Jacob generate a 'good' protocol for the same set of protocol goals only in 3 minutes using an evolutionary approach [10] based on the SVO logic.

Since variable instantiation in ASPB's single goal synthesiser disregards higher order belief formulae, and it is possible to use these higher order belief formulae to generate some protocols that are different to protocols generated using the first order beliefs, then ASPB does not search the entire protocol space to generate all possible protocols that satisfy requirement specifications. When compared to the above approaches, ASPB generates a number of 'good' protocols, including the 'best' protocol, in a shorter period. For example, ASPB generates a number of four message mutual authentication and key exchange protocols for the same set of protocol goals and assumptions within 20 seconds. The reason for this is that the Single Goal Synthesiser uses the heuristics to direct its backwards search for valid

protocols from a protocol goal. Unlike [12] and [30] which compose random messages, our 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.

Athena [33] is used to guarantee that APG generates correct protocols. However, fine grained distinctions between honesty and dishonesty cannot be made, since the underlying analysis model of APG, the strand space model, does not consider the concepts of honesty and dishonesty. Sometimes, an improper trust relationship may lead to some principals being cheated by others. With the BSW-ZF Logic, it is possible to distinguish honest (and competent) from dishonest (and incompetent) principals. This fine-grain distinction is useful for a protocol designer.

Since the Single Goal Synthesiser is completely independent of the Protocol Composer, our approach does not depend on the BSW-ZF logic. In future research, we could extend the logic (or change the basic logic to another) to suit more complex requirements.

Finally, increased performance could be achieved by parallelising the single goal synthesis and composition steps across separate processors.

8 Conclusion

In this paper we describe how the synthesis process of the BSW-ZF 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 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 synthesised 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 [28, 31]. Nevertheless, ASPB could be used narrow down the set of candidate protocols to be verified by a more sophisticated checker.

9 Acknowledgements

Acknowledgements. This research has been supported, in part, by Science Foundation Ireland grant 08/SRC/11403.

References

[1] ISO/IEC 9798-2. Information technology - security techniques – entity authentication part 2: Mechanisms using symmetric encipherment algorithems (second edition), 1999.

[2] ISO/IEC 9798-3. Information technology - security techniques – entity authentication part 3: Mechanisms entity authentication using digital signature techniques (second edition), 1998.

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

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

[5] J. Alves-Foss and T. Soule. A weakest precondition calculus for analysis of cryptographic protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[6] T. Aura. Strategies against replay attacks. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 59–68, Washington, DC, USA, 1997. IEEE Computer Society Press.

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

[8] L. Buttyán, S. Staamann, and U. Wilhelm. A simple logic for authentication protocol design. In *Proceedings of the 11th IEEE Computer*

Security Foundations Workshop, pages 153–162. IEEE Computer Society Press, 1998.

[9] U. Carlsen. Optimal privacy and authentication on a portable communications system. *SIGOPS Operating Systems Review*, 28(3):16–23, 1994.

[10] H. Chen, J. A. Clark, and J. L. Jacob. Synthesising efficient and effective security protocols. *Electronic Notes in Theoretical Computer Science*, 125(1):25–41, 2005.

[11] J. A Clark and J. L Jacob. A survey of authentication protocol literature, version 1.0., 1997. URL `http://www.cs.york.ac.uk/~jac/`.

[12] J. A Clark and J. L Jacob. Searching for a solution: Engineering tradeoffs and the evolution of provable secure protocols. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 82–95. IEEE Computer Society Press, May 2000.

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

[14] L. Gong, R. Needham, and R. 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.

[15] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 44–55, 1995.

[16] J. D. Guttman. Security protocol design via authentication tests. In *Proceedings of 15th IEEE Computer Security Foundations Workshop*, pages 92–103. IEEE Computer Society Press, April 2002.

[17] J. D. Guttman and F. J. Thayer. Authentication tests. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 96–109. IEEE Computer Society Press, 2000.

[18] D. Kindred. *Theory Generation for Security Protocols*. PhD thesis, Carnegie Mellon University, 1999. Also available as Carnegie Mellon University, Computer Science Report No. CMU-CS-99-130.

[19] D. E. Knuth. *The art of computer programming*, volume 3. Addison-Wesley, Reading, 2nd edition, 1998. sorting and searching.

[20] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes In Computer Science*, pages 147–166. Springer-Verlag, Berlin Germany, 1996.

[21] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 31–43, Los Alamitos, 1997. IEEE Computer Society Press.

[22] W. Mao. An augmentation of BAN-like logics. In *Proceedings of the Eighth IEEE Computer Security Foundations Workshop (CSFW'95)*, pages 44–57. IEEE Computer Society Press, 1995.

[23] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[24] J. K. Millen. The interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy (SP'95)*, pages 251–260, Washington, DC, USA, 1995. IEEE Computer Society.

[25] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society Press, May 1997.

[26] D. O'Crualaoich. Theory generation for the Simple Logic, Bachelor Thesis, University College Cork, 2002.

[27] L. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.

[28] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1, 2):85–128, 1998.

[29] A. Perrig and D.X. Song. A first step towards the automatic generation of security protocols. In *Proceedings of Network and Distributed System Security Symposium*, February 2000.

[30] 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*, pages 64–76. IEEE Computer Society Press, July 2000.

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

[32] H. Saidi. Towards automatic synthesis of security protocols. In *Proceedings of the Logic-Based Program Synthesis Workshop, AAAI 2002 Spring Symposium*, Stanford University, California, March 2002.

[33] D. X. Song. Athena: a new efficient automated checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE Computer Society Press, June 1999.

[34] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 187–191. IEEE Press, 1994.

[35] P. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of IEEE Computer Security Foudations Workshop*, pages 14–28. IEEE Computer Society Press, 1994.

[36] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 20th IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society Press, 1998.

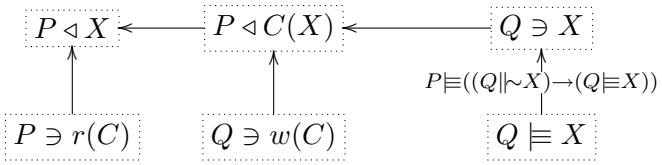
[37] W. Tzeng and C. Hu. Inter-protocol interleaving attacks on some authentication and key distribution protocols. *Information Processing Letters*, 69(6):297–302, 1999.

[38] H. Zhou and S. N. Foley. Fast automatic synthesis of security protocols using backward search. In *proceedings of the ACM Workshop on Formal Methods for Security Engineering (FMSE)*, pages 1–10, Washington DC, October 2003.

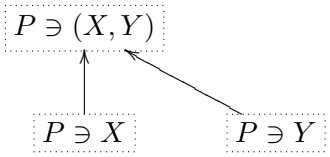
A The BSW-ZF Heuristic Rules

The following heuristic rules have their origins in the synthesis rules of the BSW logic.

Heur1 (explained in main paper)

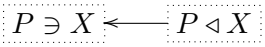


Heur2 To hold a compound message (X, Y), P may hold the message components X and Y separately.

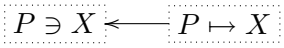


Note that this rule does not require that P must hold the message components X and Y separately. The reason for this is that P may obtain the compound message (X, Y) from the same message that he received. For example, when principal B sees a compound message (A, Na), it is not necessary for B to hold A and Na separately in order to hold the compound message.

Heur3 To hold a message X, P may see X.

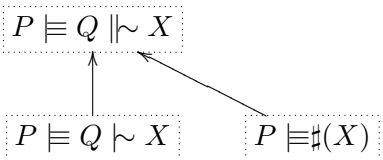


Heur4 To hold a message X, P may generate X.



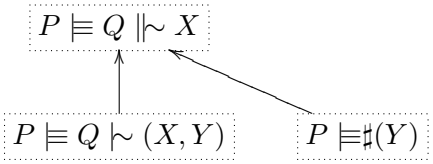
This rule is used to track the origin of all message components in a protocol on the basis that even if a principal holds all components of a message, he may not be willing to send out the message, unless he receives some request to do so or as the protocol initiator.

Heur5 To believe Q says X, P may believe that Q said X and X is fresh.



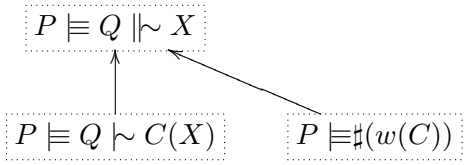
Sometimes a principal may believe another principal says something when it was once said with something fresh. This is reflected by **Heur6**.

Heur6 To believe Q says X, P may believe that Q said (X, Y) and Y is fresh.

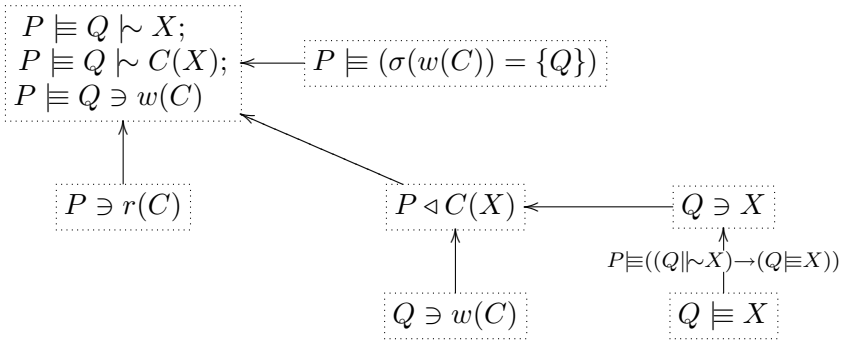


When the subgoal $P \models \sharp(Y)$ is reached, by **F2** then $P \models \sharp(X, Y)$ is also reached. Thus, together with $P \models Q \mid\sim (X, Y)$, and by rules **F1** and **F4**, $P \models Q \mid\sim (X, Y)$ and $P \models Q \mid\sim X$ are reached.

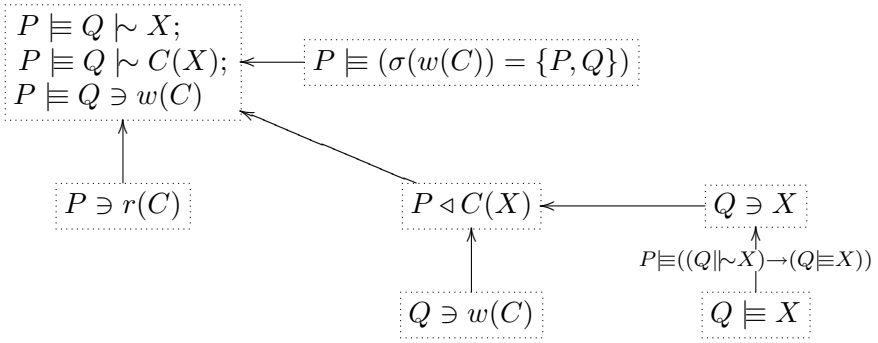
Heur7 To believe Q says X, P may believe that Q said X on C and $w(C)$ is fresh.



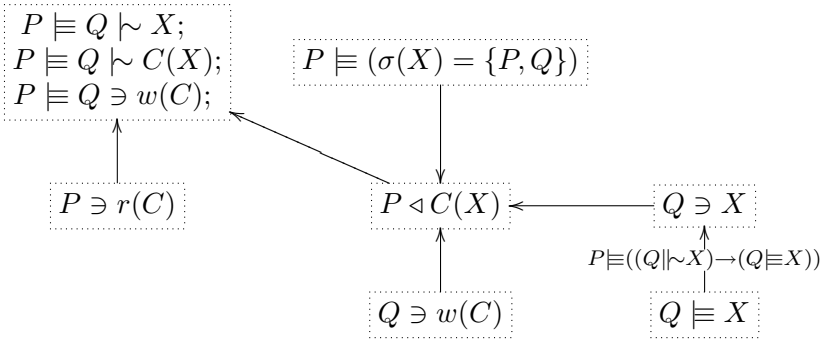
Heur8 To believe that Q said X, Q said X on channel C, and Q holds $w(C)$, P has to receive X via a channel C that he can read and that he believes that it can be written to only by Q. Furthermore, Q must hold X and $w(C)$. If X is a formula and P believes that Q is honest, then Q must also believe X.



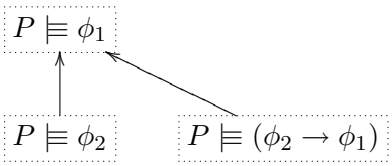
Heur9 To believe that Q said X, Q said X on channel C, and Q holds $w(C)$, P has to receive X via a channel C that he can read and that he believes it can be written to only by P and Q. Furthermore, Q must hold X and $w(C)$. If X is a formula and P believes that Q is honest, then Q must also believe X.



Heur10 To believe that Q said X, P has to receive X via a channel C that he can read and that he believes X is a secret between Q and himself before he sees it. Furthermore, P must believe that Q is able to write to C, and that Q must believe that he sees X.



Heur11 (explained in main paper)



Heur12 To believe a compound message X is fresh, P must believe some part X' of X is fresh.

$$P \models \sharp(X) \longleftarrow P \models \sharp(X')$$

Heur13 To believe $w(C)$ is fresh, P must believe $r(C)$ is fresh.

$$P \models \sharp(w(C)) \longleftarrow P \models \sharp(r(C))$$

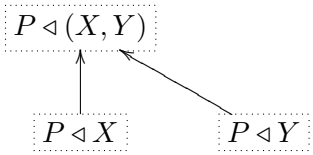
Heur14 To believe $r(C)$ is fresh, P must believe $w(C)$ is fresh.

$$P \models \sharp(r(C)) \longleftarrow P \models \sharp(w(C))$$

Heur15 To believe a message X is a shared secret only between P and Q, P must believe some part of the message is a shared secret only between P and Q.

$$P \models (\sigma(X) = \{P, Q\}) \longleftarrow P \models (\sigma(X') = \{P, Q\})$$

Heur16 To see a compound message (X, Y), P may see the message components X and Y separately.



Proposition (Provable Synthesis) A protocol synthesised using the BSW-ZF heuristic rules can be proven to uphold its protocol goal using the BSW-ZF inference rules.

Proof Outline The validity of the heuristic rules is justified by their corresponding inference rules, except for **Heur1, 8, 9, 10**. The heuristic rules **Heur1, 8, 9, 10** require additional subgoals to ensure valid temporal ordering of the subgoals of generated protocols, as discussed in **Heur1**.

These heuristic rules require that necessary subgoals are satisfied before a protocol goal is achieved. Any protocol synthesised using these heuristic rules guarantees that all of the subgoals for the protocol goals are satisfied. To verify a protocol, its subgoals serve as the premises of the inference rules. It is sufficient to deduce protocol goals from these subgoals using the corresponding inference rules.

B Protocol Examples

ASPB generates a large number of protocols for different requirements. For the sake of illustration, only a small number of generated protocols are explored in this section. Note that we do not consider type-flaw attacks [11] in our prototype. However, this could be done by using other protocol checkers, such as the NRL Analyzer [23], the Interrogator model [24], FDR [20], Murφ [25], Athena [33], during protocol selection.

B.1. Mutual Authentication with TTP

Figure 2 illustrates a protocol requirement specification of a simple mutual authentication protocol that uses a trusted third party. ASPB generates Protocol 1.1 that was described in Example 4.

Protocol 1.1. (Perrig and Song's real optimal protocol [30])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow S: & \{A, Na, Nb\}_{K_{bs}}, \\ Message\ 3 & S \rightarrow A: & \{Na, Nb, B\}_{K_{as}}, \\ Message\ 4 & A \rightarrow B: & Nb. \end{array}$$

Protocol 1(a). (Perrig and Song's optimal protocol 1 [30])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow S: & \{Na, Nb, A\}_{K_{bs}}, \\ Message\ 3 & S \rightarrow A: & \{Na, Nb, A, B\}_{K_{as}}, \\ Message\ 4 & A \rightarrow B: & Nb. \end{array}$$

Protocol 1(b). (Perrig and Song's optimal protocol 2 [30])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow S: & \{A, B, Na, Nb\}_{K_{bs}}, \\ Message\ 3 & S \rightarrow A: & \{Na, Nb, B\}_{K_{as}}, \\ Message\ 4 & A \rightarrow B: & Nb. \end{array}$$

Protocol 1(a) and Protocol 1(b) were generated by Perrig and Song's APG [30]. While ASPB also generates these two protocols, it considers them as interim protocols with redundant components. For example, the redundant components in Protocol 1(a) is A in $Message\ 3$. Before S sends out $Message\ 3$, $\mathcal{N}_A(Na) = \mathcal{N}_S(Na) = \{A, S\}$ is obtained by using **RR1** and

RR3. Therefore, $\mathcal{N}_A(Na) \cap \mathcal{N}_S(Na) = \{A, S\}$. Since $\{A, B\}/\{A, S\} = \{B\}$, A is safely removed from $Message$ 3. The protocol properties do not change after the redundant components are removed. The redundant components in Protocol 1(b) is B in $Message$ 2. Similar redundant removing process can be applied on Protocol 1(b).

B.2. Mutual Authentication without TTP

Figure 2 illustrates a requirement specification of a simple mutual authentication protocol that uses a trusted third party. If the assumptions are changed to reflect the absence of this third party (all of the assumptions using S, Cas, and Cab are replaced by proper assumptions) then the synthesised protocols include the following.

B.2.1. Using symmetric keys

In this case, principals A and B share a symmetric key K_{ab}. The following assumptions about channel Cab are used in the modified requirement specification that is illustrated in Figure 7. For this modified requirement specification, ASPB generates following six correct protocols, Protocol 2.1–2.6.

Protocol 2.1. (Perrig and Song's Optimal Minimum Cost Protocol [29])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & \{A, Na\}_{K_{ab}}, \\ Message\ 2 & B \rightarrow A: & \{Na, Nb\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow B: & Nb. \end{array}$$

Protocol 2.2. (Perrig and Song's Minimum Cost Protocol 2 [29])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow A: & \{B, Na, Nb\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow B: & Nb. \end{array}$$

Protocol 2.3. (Perrig and Song's Minimum Cost Protocol 1 [29])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow A: & \{A, Na, Nb\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow B: & Nb. \end{array}$$

The above three protocols were first generated by APG [29]. Compare with Protocol 2.2 and Protocol 2.3, Protocol 2.1 has an encrypted first

declarations {
Channel Cab, Cp;
Principal A, B;
Nonce Na, Nb;
}
assumptions {
$A \models (\sigma(w(Cab)) = \{A, B\})$;
$B \models (\sigma(w(Cab)) = \{A, B\})$;
$A \models (\sigma(r(Cab)) = \{A, B\})$;
$B \models (\sigma(r(Cab)) = \{A, B\})$;
$A \ni r(Cab)$; $A \ni w(Cab)$;
$A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cab)$; $B \ni w(Cab)$;
$B \ni r(Cp)$; $B \ni w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \mapsto Na$; $B \mapsto Nb$;
$A \ni A$; $B \ni B$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
}

Figure 7: A Requirement Specification for Mutual Authentication without TTP using Symmetric Keys

message, and a shorter second message. Perrig and Song consider that Protocol 2.1 can be used in environments that perform fast encryption and decryption operations, but using slow links for data transitions. Protocol 2.2 and Protocol 2.3 may be used in regular environments, that the cost of data transitions is lower than encryption and decryption operations.

Note that Protocol 2.1 also satisfies the secrecy requirement that is described as goal $G_{a1} : A \models (\sigma(Na) = \{A, B\})$ and goal $G_{a2} : B \models (\sigma(Na) = \{A, B\})$. By satisfying G_{a1} and G_{a2} after a round of these protocols, it is possible for A and B to use Na as a secret shared only between them for further communication. Protocol 2.2 and Protocol 2.3 do not satisfy goal G_{a1} and G_{a2}.

Before obtaining Protocol 2.2 and Protocol 2.3, ASPB generates the fol-

lowing interim Protocol 2(a) that has a redundant component in the second message.

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow A: & \{A, B, Na, Nb\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow B: & Nb. \end{array}$$

In the redundancy removing process, either protocol initiator's identifier A or protocol responder's identifier B can be removed from $Message\ 2$. This does not change the properties of Protocol 2(a). After removing one of the message components A and B, Protocol 2.2 and Protocol 2.3 are obtained individually. On the other hand, according to the redundancy removing rules, message components A and B may not be removed at the same time to obtain Protocol 2(b).

Protocol 2(b).

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow A: & \{Na, Nb\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow B: & Nb. \end{array}$$

The reason for this is that Protocol 2(b) is subject to the following reflection/oracle attack. When E_B intercepts message 1, E_B may start another round of Protocol 2(b) by forwarding all of A's messages to A. After message 3, A believes that A finished two rounds of the protocol with B, but B does not participate in any round of the protocol.

$$\begin{array}{lll} Message\ 1 & A \rightarrow E_B: & A, Na, \\ Message\ 1' & E_B \rightarrow A: & B, Na, \\ Message\ 2' & A \rightarrow E_B: & \{Na, Na'\}_{K_{ab}}, \\ Message\ 2 & E_B \rightarrow A: & \{Na, Na'\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow E_B: & Na'. \\ Message\ 3' & E_B \rightarrow A: & Na'. \end{array}$$

Protocol 2.4.

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & \{A, Na\}_{K_{ab}}, \\ Message\ 2 & B \rightarrow A: & \{Na, Nb\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow B: & \{Nb\}_{K_{ab}}. \end{array}$$

Protocol 2.5.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{B, Na, Nb\}_{K_{ab}}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb\}_{K_{ab}}. \end{aligned}$$

Protocol 2.6.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na, Nb\}_{K_{ab}}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb\}_{K_{ab}}. \end{aligned}$$

Protocol 2.4–2.6 are similar to Protocol 2.1–2.3, but each of them has an encrypted third message. Comparing with Protocol 2.1–2.3, we consider that Protocol 2.4–2.6 also satisfy the secrecy requirement, that are described as additional protocol goals, $G_{a3} : B \models (\sigma(Nb) = \{A, B\})$ and $G_{a4} : A \models (\sigma(Nb) = \{A, B\})$. By satisfying G_{a3} and G_{a4} after a round of these protocols, A and B can use Nb as a secret shared only between them for further communication.

ISO/IEC 9798-2 [1] proposes the ISO/IEC Symmetric-Key Three-Pass Mutual Authentication Protocol, as follows:

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb, A\}_{K_{ab}}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Na, Nb\}_{K_{ab}}. \end{aligned}$$

Clearly, Protocols 2.1–2.6, generated by ASPB, are simpler than the ISO standard protocol, yet achieve the same mutual authentication goals. Therefore, it is reasonable to consider that component Na in $Message\ 3$ of the standard ISO protocol is redundant component for the purposes of mutual authentication.

B.2.2. Using signature keys

In this case, both principals can verify each others signature key SK_a and SK_b. Assumptions are adapted to represent the corresponding authentic channel Ca and Cb. The modified requirement specification is represented in Figure 8. For this modified requirement specification, ASPB generated two correct protocols.

declarations {
Channel Cab, Cp;
Principal A, B;
Nonce Na, Nb;
}
assumptions {
$A \models (\sigma(w(Ca)) = \{A\})$;
$B \models (\sigma(w(Ca)) = \{A\})$;
$A \models (\sigma(w(Cb)) = \{B\})$;
$B \models (\sigma(w(Cb)) = \{B\})$;
$A \ni r(Ca)$; $A \ni w(Ca)$; $A \ni r(Cb)$;
$A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cb)$; $B \ni w(Cb)$; $B \ni r(Ca)$;
$B \ni r(Cp)$; $B \ni w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \mapsto Na$; $B \mapsto Nb$;
$A \ni A$; $B \ni B$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
}

Figure 8: A requirement specification for Mutual Authentication without TTP using signature keys

Protocol 2.7.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na, Nb\}_{SK_b}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb, B\}_{SK_a}. \end{aligned}$$

Protocol 2.8.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad \{A, Na\}_{SK_a}, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na, Nb\}_{SK_b}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb, B\}_{SK_a}. \end{aligned}$$

The only difference between Protocol 2.7 and Protocol 2.8 is whether the first message is signed by A's signature key. ASPB generates these

two different protocols because ASPB's Single Goal Synthesiser instantiates typed variable with all possible instantiations that are defined in requirement specification. Since both Protocol 2.7 and Protocol 2.8 satisfy given mutual authentication goals, the above difference does not influence the authentication properties of these protocols. Therefore, it is reasonable to consider that the signature in $Message\ 1$ of Protocol 2.8 is redundant component for the purposes of mutual authentication.

ISO/IEC 9798-3 [2] proposes the ISO/IEC Signature-Key Three-Pass Mutual Authentication Protocol, as follows:

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na, Nb\}_{SK_b}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Na, Nb, B\}_{SK_a}. \end{aligned}$$

Clearly, Protocol 2.7, a protocol generated by ASPB, is simpler than the ISO standard protocol, yet achieves the same mutual authentication goals. Therefore, it is reasonable to consider that component Na in $Message\ 3$ of the standard ISO protocol is redundant component for the purposes of mutual authentication.

B.2.3. Using Public Keys

In this case, both principals know each others' public keys. The following assumptions about channel Ca and Cb are used in the modified requirement specification. The modified requirement specification is represented in Figure 9. For this modified requirement specification, the following two protocols were generated by ASPB.

Protocol 2.9. (Corresponds to Needham-Schroeder-Lowe protocol [20])

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad \{A, Na\}_{K_b}, \\ Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb, B\}_{K_a}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb\}_{K_b}. \end{aligned}$$

Protocol 2.10. (Perrig and Song's Minimum Cost Protocol 3 [29])

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad \{A, Na\}_{K_b}, \\ Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb, B\}_{K_a}, \\ Message\ 3 \quad & A \rightarrow B: \quad Nb. \end{aligned}$$

The difference of Protocol 2.9 and Protocol 2.10 is whether the last message is encrypted. Consequently, Protocol 2.9 achieves the secrecy requirement, that is described as additional protocol goals $G_{a3} : B \models (\sigma(Nb) =$

declarations {
Channel Cab, Cp;
Principal A, B;
Nonce Na, Nb;
}
assumptions {
$A \models (\sigma(r(Ca)) = \{A\})$;
$B \models (\sigma(r(Ca)) = \{A\})$;
$A \models (\sigma(r(Cb)) = \{B\})$;
$B \models (\sigma(r(Cb)) = \{B\})$;
$A \ni r(Ca)$; $A \ni w(Ca)$; $A \ni w(Cb)$;
$A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cb)$; $B \ni w(Cb)$; $B \ni w(Ca)$;
$B \ni r(Cp)$; $B \ni w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \mapsto Na$; $B \mapsto Nb$;
$A \ni A$; $B \ni B$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
}

Figure 9: A requirement specification for Mutual Authentication without TTP using Public Keys

$\{A, B\}$) and $G_{a4} : A \models (\sigma(Nb) = \{A, B\})$, while Protocol 2.10 does not satisfy these secrecy requirements.

B.3. Mutual Authentication and Key Agreement Protocol

Figure 10 gives a complete requirement specification for a mutual authentication and key agreement protocol that involves a Trusted Third Party.

B.3.1. Four Message Protocols

ASPB synthesises a number of four-message mutual authentication protocols. For the purpose of illustration, we describe and discuss the following

declarations {
Channel Cas, Cbs, Cab, Cp;
Principal A, B, S;
Nonce Na, Nb;
Message X;
Formula ϕ;
}
assumptions {
$A \models (\sigma(w(Cas)) = \{A, S\})$; $A \models (\sigma(r(Cas)) = \{A, S\})$;
$S \models (\sigma(w(Cas)) = \{A, S\})$; $S \models (\sigma(r(Cas)) = \{A, S\})$;
$B \models (\sigma(w(Cbs)) = \{B, S\})$; $B \models (\sigma(r(Cbs)) = \{B, S\})$;
$S \models (\sigma(w(Cbs)) = \{B, S\})$; $S \models (\sigma(r(Cbs)) = \{B, S\})$;
$S \models (\sigma(w(Cab)) = \{A, B\})$; $S \models (\sigma(r(Cab)) = \{A, B\})$;
$A \ni r(Cas)$; $A \ni w(Cas)$; $A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cbs)$; $B \ni w(Cbs)$; $B \ni r(Cp)$; $B \ni w(Cp)$;
$S \ni r(Cas)$; $S \ni w(Cas)$; $S \ni r(Cp)$; $S \ni w(Cp)$;
$S \ni r(Cbs)$; $S \ni w(Cbs)$;
$A \models\sharp(Na)$; $B \models\sharp(Nb)$; $S \models\sharp(w(Cab))$;
$A \mapsto Na$; $B \mapsto Nb$; $S \mapsto r(Cab)$; $S \mapsto w(Cab)$;
$A \ni A$; $B \ni B$; $S \ni S$;
$A \models ((S \mid\sim \phi) \rightarrow (S \models \phi))$;
$B \models ((S \mid\sim \phi) \rightarrow (S \models \phi))$;
$A \models ((S \models (B \mid\sim X)) \rightarrow (B \mid\sim X))$;
$B \models ((S \models (A \mid\sim X)) \rightarrow (A \mid\sim X))$;
$A \models ((S \models (\sigma(w(Cab)) = \{A, B\}) \rightarrow (\sigma(w(Cab)) = \{A, B\})))$;
$B \models ((S \models (\sigma(w(Cab)) = \{A, B\}) \rightarrow (\sigma(w(Cab)) = \{A, B\})))$;
$A \models (S \mid\sim w(Cab) \rightarrow\sharp(w(Cab)))$;
$B \models (S \mid\sim w(Cab) \rightarrow\sharp(w(Cab)))$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
$A \models (\sigma(w(Cab)) = \{A, B\})$; /* G_3 */
$B \models (\sigma(w(Cab)) = \{A, B\})$; /* G_4 */
}

Figure 10: The requirement specification for the mutual authentication and key agreement protocol using Trusted Third Party.

generated protocols, Protocols 3.1–3.7, that satisfy the requirement specification in Figure 10.

Protocol 3.1. (Perrig and Song's protocol 1 in Protocol-Set S1 [30])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Protocol 3.2. (Perrig and Song's protocol in Protocol-Set S3 [30])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & Nb, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

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). Protocols 3.1 and 3.2 also appear in Protocol-Sets S1 and S3 from [30]. In addition, Protocol 3(a) was generated by APG in Protocol-Set S1 [30], as follows:

Protocol 3(a). (Perrig and Song's protocol 2 in Protocol-Set S1 [30])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{\{Nb, K_{ab}\}_{K_{bs}}\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

ASPB does not generate Protocol 3(a). The reason for this is that ASPB does not allow a message to be sent into multiple channels at the same protocol step (as $\{\{Nb, K_{ab}\}_{K_{bs}}\}_{K_{ab}}$ in Message 4). Instead, ASPB generates Protocol 3.3 that is similar to Protocol 3(a). Compare to Protocol 3(a), Protocol 3.3 has a simpler message component $\{B\}_{K_{ab}}$ in Message 4. Since both of message components $\{\{Nb, K_{ab}\}_{K_{bs}}\}_{K_{ab}}$ and $\{B\}_{K_{ab}}$ are used to prove that principal A holds session key K_{ab}, and the use of session key K_{ab} may prove the freshness of the current message component, the content encrypted by K_{ab} is only required to be recognisable by B, but is not required to be fresh.

Protocol 3.3.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{B\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

However, for the requirement specification in Figure 9, ASPB does not generate protocols in the Protocol-Set S2 for authentication and key agreement from [30] (this includes the original Yahalom protocol). The reason is that the protocols in Protocol-Set S2 have an assumption that B believes that A is honest. If B believes A accepts a session key then B will accept it. Otherwise, A is possible to make B to accept an old session key in the current round of protocol. For example, Protocol 3(b) is Protocol 1 in Protocol-Set S2 [30].

Protocol 3(b).

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{K_{ab}\}_{K_{bs}}.
\end{array}
$$

When assumption, that B believes that A is honest, is not made, A is able to generate and send message

$$
Message\ 4' \quad A \rightarrow B: \quad \{Nb\}_{K'_{ab}}, \{K'_{ab}\}_{K_{bs}}
$$

where K'_{ab} is an old (expired) session key between A and B.

For the sake of illustration, this assumption was not made in the requirement specification in Figure 10 which was used to conduct our experiments.

Protocol 3.4.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \{B, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{B, Na\}_{K_{as}}, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

In the new Protocol 3.4, when the TTP S receives $Message\ 2$, S may check whether two principals know who the other party is in the current round. If S finds that one principal attempts to cheat the other, then it

can stop the current round as early as possible. While Protocol 3.4 has a higher cost (in terms of message size) than the other protocols, it provides a more powerful TTP. Two new protocols, that are similar to Protocol 3.4, are generated by ASPB as Protocol 3.5 and 3.6.

Protocol 3.5.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Protocol 3.6.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & Nb, \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Protocol 3.7.(BAN optimized Yahalom protocol [7])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & Nb, \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, K_{ab}, B\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Syverson [34] describes a flaw in Protocol 3.7 (when B is not able to distinguish the format of different components) as follows.

$$
\begin{array}{lll}
Message\ 1(\alpha) & A \rightarrow B: & A, Na, \\
Message\ 2(\alpha) & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 1(\beta) & I(A) \rightarrow B: & A, (Na, Nb) \\
Message\ 2(\beta) & B \rightarrow I(S): & B, Nb', \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3(\alpha) & omitted, & \\
Message\ 4(\alpha) & I(A) \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Na(= K_{ab}), Nb\}_{K_{bs}}.
\end{array}
$$

However, ASPB uses the following assumptions.

- A principal can recognise 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 3.7 is also a correct protocol.

ASPB generates Paulson amended Yahalom Protocol [27], but considers it as an interim protocol. After removing the redundant message components, Protocol 3.7 is obtained.

Protocol 3(c).(Paulson amended Yahalom Protocol [27])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & Nb, \{A, B, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, B, Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

B.3.2. Five Message Protocols

The ISO/IEC Symmetric-Key Five-Pass Mutual Authentication Protocol proposed in ISO/IEC 9798-2 [1].

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{Na, K_{ab}, B\}_{K_{as}}, \{Nb, K_{ab}, A\}_{K_{bs}}, \\
Message\ 4 & B \rightarrow A: & \{Na, K_{ab}, B\}_{K_{as}}, \{Na, Nb'\}_{K_{ab}}, \\
Message\ 5 & A \rightarrow B: & \{Nb', Na\}_{K_{ab}}.
\end{array}
$$

Carlsen also describes a five message protocol, the Secret Key Initiator Protocol [9], as follows:

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{Na, B, K_{ab}\}_{K_{as}}, \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & B \rightarrow A: & \{Na, B, K_{ab}\}_{K_{as}}, \{Na\}_{K_{ab}}, Nb', \\
Message\ 5 & A \rightarrow B: & \{Nb'\}_{K_{ab}}.
\end{array}
$$

In these protocols, principal B uses two nonces. The protocol requirement specification in Figure 10 that formed the basis of our experiments specified that principal B uses one nonce. A consequence of this is that the exact ISO/IEC Symmetric-Key Five-Pass Mutual Authentication Protocol and

Carlsen protocol are not generated for our given requirement specification in Figure 10. However, ASPB generates the Carlson protocol when the protocol specification is extended to include B's use of two nonces.

The remainder of this section discuss a number of five-message mutual authentication protocols generated by ASPB, that satisfy the requirement specification in Figure 10. For the purpose of illustration, we describe and discuss the following generated protocols, Protocol 4.1–4.8.

Protocol 4.1.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 4 & B \rightarrow A: & \{Na, Nb, K_{ab}, B\}_{K_{as}}, \{Na\}_{K_{ab}}, \\
Message\ 5 & A \rightarrow B: & \{Nb\}_{K_{ab}}.
\end{array}
$$

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 $Message$ 3, 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 $Message$ 4, she may believe that Nb is generated by B (otherwise, B may not generate $\{Na\}_{K_{ab}}$).

Protocol 4.2.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$

The first four messages of Protocol 4.2 are the same as Protocol 3.1. From the additional message $Message$ 5, Protocol 4.2 meets an additional goal that A believes B has received the session key Kab.

Protocol 4.3.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow A: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & A \rightarrow S: & A, \{Nb, B\}_{K_{as}}, \{A, Na\}_{K_{bs}}, \\
Message\ 4 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 5 & A \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{Nb\}_{K_{ab}}.
\end{array}
$$

Protocol 4.3 is a novel protocol. When S receives $Message\ 3$, he may check whether the components have been generated by A and B. If this is the case then S sends $Message\ 4$. Further five-message protocols that were generated by ASPB include the following.

Protocol 4.4.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \{A, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \{A, Na\}_{K_{as}}, \\
Message\ 3 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$

Protocol 4.5.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, \{B, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{B, Na\}_{K_{as}}, \{Nb, A\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow B: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & B \rightarrow A: & Na, \{Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 5 & A \rightarrow B: & Nb.
\end{array}
$$

Protocol 4.6.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, \{B, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{B, Na\}_{K_{as}}, \{Nb, A\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$

Protocol 4.7.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, \{A, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{B, Na, Nb, K_{ab}\}_{K_{as}}, \{Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & A \rightarrow B: & \{Nb, K_{ab}\}_{K_{bs}}, \{Nb\}_{K_{ab}}, \\
Message\ 5 & B \rightarrow A: & \{A\}_{K_{ab}}.
\end{array}
$$

Protocol 4.8.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow A: & \{A, Na, Nb, Kab\}_{K_{bs}}, \{B, Na, Kab\}_{K_{as}}, \\
Message\ 4 & A \rightarrow B: & \{Nb, Na\}_{K_{ab}}, \{A, Na, Nb, Kab\}_{K_{bs}} \\
Message\ 5 & B \rightarrow A: & \{Nb\}_{K_{ab}}.
\end{array}
$$

B.3.3. Six Message Protocols

ASPB synthesises a number of six-message mutual authentication protocols. For the purpose of illustration, we only describe the following protocol, Protocol 5.1, that satisfies the requirement specification in Figure 10.

Protocol 5.1.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 4 & B \rightarrow A: & \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 5 & A \rightarrow B: & \{A, Na, Nb\}_{K_{ab}}, \\
Message\ 6 & B \rightarrow A: & \{B, Na\}_{K_{ab}}.
\end{array}
$$

ASPB does not generate the six-message protocols in [10] that are listed as Protocol 5(a) and 5(b). The reason for this is that ASPB generates protocols according to corresponding principal sequences, whereby the receiver of a message is the sender of the next message. Protocol 5(a) and 5(b) can not be described in terms of principal sequences. For example, in Protocol 5(a), after A receives message 2, B (not A) sends Message 3. We observe that without the participation of previous protocol steps, B is unable to determine when Message 3 should be sent. Therefore, we argue that if a protocol is not described in terms of a principal sequence, then it is difficult to properly execute the protocol in practice.

Protocol 5(a).

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow S: & A, B, Na, \\
Message\ 2 & S \rightarrow A: & \{B, Na, Kab\}_{K_{as}}, \\
Message\ 3 & B \rightarrow S: & B, A, Nb, \\
Message\ 4 & S \rightarrow B: & \{A, Na, Nb, Kab\}_{K_{bs}}, \\
Message\ 5 & A \rightarrow B: & \{B, Nb, Na\}_{K_{ab}}, \\
Message\ 6 & B \rightarrow A: & \{Nb, A\}_{K_{ab}}.
\end{array}
$$

Protocol 5(b).

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & S \rightarrow A: & \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 5 & A \rightarrow B: & \{A, Na, Nb\}_{K_{ab}}, \\
Message\ 6 & B \rightarrow A: & \{B, Na\}_{K_{ab}}.
\end{array}
$$

Protocol 5(a) can be refined as Protocol 4.8 by reordering the message to satisfy the ASPB principal sequence and by refining messages by using the ASPB redundancy removing rules. Clearly, our 5-message protocols can be considered to be more compact and efficient than the 6-message versions in [10]. Similarly, Protocol 5(b) can be refined as Protocol 5.1, that is generated by ASPB.