COMPLEMENTING COMPUTATIONAL PROTOCOL ANALYSIS WITH FORMAL SPECIFICATIONS*

Kim-Kwang Raymond Choo, Colin Boyd, Yvonne Hitchcock, and Greg Maitland
Information Security Research Centre
Queensland University of Technology
GPO Box 2434, Brisbane, QLD 4001, Australia
{k.choo,c.boyd,y.hitchcock,g.maitland}@qut.edu.au

Abstract The computational proof model of Bellare and Rogaway for cryptographic protocol analysis is complemented by providing a formal specification of the actions of the adversary and the protocol entities. This allows a matching model to be used in both a machine-generated analysis and a human-generated computational proof. Using a protocol of Jakobsson and Pointcheval as a case study, it is demonstrated that flaws in the protocol could have been found with this approach, providing evidence that the combination of human and computer analysis can be more effective than either alone. As well as finding the known flaw, previously unknown flaws in the protocol are discovered by the automatic analysis.

1. Introduction

Cryptographic protocols are fundamental security tools for electronic communications and a high level of assurance is needed in the correctness of such protocols. Techniques to verify the correctness of security proofs for cryptographic protocols have been directed in two distinct directions, namely the formal methods approach [1, 4] and the computational complexity approach [5, 6, 10, 17].

In the formal methods approach, emphasis is placed on model checking and automatic theorem proving. Usually the abstract formal specification is in the tradition of the model of Dolev and Yao [12]. This means that a 'black box' model of cryptographic operations is used, which ignores different cryptographic properties and possible loss of partial information. Therefore it is quite possible to have flaws in protocols that were proven secure in the Dolev-

*This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.

Yao sense [3, 15] and we cannot be entirely confident that such a protocol can be implemented securely.

In the computational complexity approach, emphasis is placed on a proven reduction from the problem of breaking the protocols to another problem believed to be hard. Such proofs are invariably generated by humans. Application of the computational complexity approach to protocol analysis was initiated by Bellare and Rogaway in 1993, with a proof for a simple two party entity authentication and key exchange protocol [6]. They formally defined a model of adversary capabilities with an associated definition of security. Since then, the model has been further revised several times. In 1995, Bellare and Rogaway analysed a three-party server-based key distribution protocol [7] using an extension to the 1993 model. The most recent revision to the model was proposed in 2000 by Bellare, Pointcheval and Rogaway [5], hereafter referred to as the BPR2000 model.

A complete mathematical proof with respect to cryptographic definitions provides a strong assurance that a protocol is behaving as desired. However, the difficulty of obtaining correct computational proofs of security has been illustrated dramatically by the well-known problem with the OAEP mode for public key encryption [17]. Although OAEP was one of the most widely used and implemented algorithms, it was several years after the publication of the original proof that a problem was found (and subsequently fixed in the case of RSA). Problems with proofs of protocol security have occurred too. In this paper, we will use the original version of a protocol due to Jakobsson and Pointcheval [13] which carried a claimed proof in the Bellare-Rogaway model but was later found to be flawed by Wong and Chan [18].

In recent years a number of researchers have started to recognize the disparity in the two different approaches to protocol analysis. Previous efforts in unifying the two domains have been devoted towards providing abstract models of cryptographic primitives which are suitable for machine analysis and yet can be proven to be functionally equivalent (in some well-defined sense) to the real cryptographic primitives that they model. Abadi and Rogaway [2] started this trend, and more recently comprehensive efforts have been under way in two different but related projects by Canetti [9] and by Backes *et al.* [3].

In this work we take a different, more pragmatic, approach to the problem. We are motivated by the observation that so far no researchers have tried to utilize the communication and adversary model from computational proofs in a machine specification and analysis. Although we cannot capture the complexity-based definitions for security and cryptographic primitives, we can ensure that the same protocol and adversary capabilities are specified in both the human-generated proofs and the machine analysis. In other words, rather than trying to unify the two approaches, we treat them as complementary but ensure that, as far as possible, they are analysing the same objects.

Our thesis is that the human proof will take care of the cryptographic details lacking in the machine analysis, while the machine analysis will help to ensure that human error resulting in basic structural mistakes is avoided.

We provide a formal specification and machine analysis of the adversary model from the BPR2000 model as shown in Figure 1. The Bellare-Rogaway model is the most widely used model for human-generated security proofs of protocols. As a case study we analyse the protocol of Jakobsson and Pointcheval. The original version appeared in the unpublished pre-proceedings of Financial Cryptography 2001 with a claimed proof of security in the Bellare-Rogaway model. Nevertheless, a flaw in the protocol was discovered by Wong and Chan. In the published paper [13], the flaw in the protocol has been fixed.

Our choice of formalism for this work is Asynchronous Product Automata (APA), a universal state-based formal method [16]. APA is supported by the Simple Homomorphism Verification Tool (SHVT) [14] for analysis and verification of cooperating systems and communicating automata. Once the possible state transitions of each automaton have been specified, SHVT can be used to automatically search the state space of the model. SHVT provides a reachability graph of the explored states. In our APA specification, the abstract communication model captures the representation of the protocol, the message transmission, and the communication channels. The automated state space analyses performed with SHVT reveal the known attack on the Jakobsson-Pointcheval protocol and also two other previously unpublished attacks.

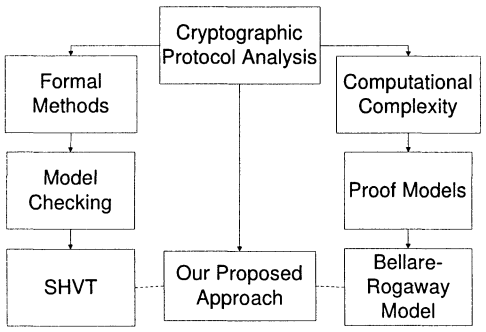

Figure 1. Our proposed approach

This work differs significantly from related earlier work of Boyd and Viswanathan [8], as their formal specification did not capture the entire Bellare- Rogaway model. In addition, no automatic searching was performed and no new attack was revealed in their earlier work.

We regard the main contributions of this paper to be confirmation of the feasibility of using formal specifications to identify problems in human-generated computational complexity proofs, demonstration of the use of SHVT in an automated manner to find unknown attacks in protocols, and a re-usable framework for automatic analysis of protocols proven secure in the BPR2000 model.

The remainder of this paper is structured as follows: Section 2 briefly explains the adversarial model used in our formal specification framework and the Bellare-Rogaway adversarial model. Section 3 describes the original version of the mutual authentication and key exchange protocol (MAKEP) due to Jakobsson and Pointcheval, and the hijacking attack first mentioned by Wong and Chan. Section 4 briefly outlines the state-based APA specification, followed by the results of the protocol analysis using SHVT. Section 5 presents the conclusions.

2. Overview of Our Formal Specification Framework

In this section, we present an overview of the BPR2000 model, followed by a definition of an adversary in our APA formal specification framework. We follow the general adversarial formalism of the BPR2000 model, except that the probabilistic characteristics of the BPR2000 adversary are not explicitly modelled in our formal specification due to the deterministic nature of SHVT. However, since our thesis is that the human proof will take care of the cryptographic details lacking in the machine analysis, this does not present an obstacle to our protocol analysis.

2.1 The BPR2000 Model

The BPR2000 model defines provable security for entity authentication and key distribution goals. In the model, the adversary $\mathcal{A}_{BR}$ is a probabilistic machine that has the capability to read, delay, replay, modify, delete, and fabricate messages between communicating principals and to start new instances of communicating principals. $\mathcal{A}_{BR}$ controls all the communications that take place between parties by interacting with a set of oracles at any time in any order. Each of the oracles represents an instance of a principal (Π_U^i denotes the i-th instance of a principal U) in a specific protocol run. The predefined oracle queries are described informally as follows.

- The SendClient and SendServer queries allow the adversary to simulate the actions of the principals according to the protocol by sending some message of her choice to any client or server oracle at will. The client or server oracle, upon receiving the query, will compute what the protocol specification demands and send back the response.

- The Reveal query allows the adversary to expose an old session key which has been previously accepted. Any oracle receiving this query, if it has accepted and holds some session key, will send this session key back to the adversary. This query enables the modelling of the requirement that loss of a session key should only affect the session that used the key, and not any other session. In addition, some session keys may not need to be kept secret after the completion of a session, e.g. keys used for message authentication.

- The Corrupt query allows the adversary to corrupt any principal at will, and thereby learn the complete internal state of the corrupted principal. The Corrupt query also gives the adversary the ability to overwrite the long-lived key of the corrupted principal with any value of her choice. This query can be used to model the real world scenarios of an insider cooperating with the adversary and an insider who has been completely compromised by the adversary.

- The Test query is the only oracle query that does not correspond to any of $\mathcal{A}_{BR}$'s abilities. If the oracle being asked a Test query has accepted with some session key, and depending on the randomly chosen bit b, $\mathcal{A}_{BR}$ is given either the actual session key or a session key drawn randomly from the session key distribution. $\mathcal{A}_{BR}$ may only make one Test query during a game simulation. The use of the Test query is explained in Section 2.1.3.

The definition of security depends on the notions of partnership of oracles and indistinguishability. The definition of partnership is used in the definition of security to restrict the adversary's Reveal and Corrupt queries to oracles that are not partners of the oracle whose key the adversary is trying to guess.

2.1.1 Definition of Partnership. Partnership is defined using the notion of session identifiers (*SIDs*). *SIDs* are defined as the concatenation of messages exchanged during the particular protocol run in question. An oracle who has accepted will hold the associated session key, a *SID* and a partner identifier (*PID*).

DEFINITION 1 *Two oracles, Π_A^i and Π_B^j, are partners if, and only if, both oracles have accepted the same session key with the same SID, have agreed on the same set of principals (i.e., the initiator and the responder of the protocol), and no other oracles besides Π_A^i and Π_B^j have accepted with the same SID.*

2.1.2 Definition of Freshness. Definition 2 describes the notion of freshness, which depends on the notion of partnership in Definition 1.

DEFINITION 2 *Oracle Π_A^i is fresh (or it holds a fresh session key) at the end of execution, if, and only if, oracle Π_A^i has accepted with or without a partner oracle Π_B^j, both oracle Π_A^i and its partner oracle Π_B^j (if such a partner oracle exists) are unopened (i.e., have not been sent a* Reveal *query), and none of the players are corrupted (i.e., no one has been sent a* Corrupt *query).*

2.1.3 Definition of Security. Security is defined using the game $\mathcal{G}$, played between a malicious adversary $\mathcal{A}_{BR}$ and a collection of oracles. $\mathcal{A}_{BR}$ runs $\mathcal{G}$ and is able to send any oracle queries at will. At some point during $\mathcal{G}$, $\mathcal{A}_{BR}$ will choose a fresh session on which to be tested and send a Test query to the fresh oracle associated with the test session. Depending on the randomly chosen bit b, $\mathcal{A}_{BR}$ is given either the actual session key or a session key drawn randomly from the session key distribution. $\mathcal{A}_{BR}$ continues making any oracle queries of its choice. Eventually, $\mathcal{A}_{BR}$ terminates the game and outputs a bit b', which is its guess of the value of b.

Success of $\mathcal{A}_{BR}$ is measured in terms of $\mathcal{A}_{BR}$'s advantage in distinguishing whether $\mathcal{A}_{BR}$ receives the real key or a random value. $\mathcal{A}_{BR}$ wins if, after asking a Test query, $\mathcal{A}_{BR}$'s guess bit b' equals the bit b selected during the Test query. If the advantage of $\mathcal{A}_{BR}$ is denoted by $\mathsf{Adv}^{\mathcal{A}_{\mathsf{BR}}}$, then $\mathsf{Adv}^{\mathcal{A}_{\mathsf{BR}}} = 2 \times \mathsf{Pr}[\mathsf{b} = \mathsf{b}'] - 1$.

DEFINITION 3 *A protocol is secure if both the following requirements are satisfied: (1) when the protocol is run between two oracles in the absence of a malicious adversary, the two oracles accept the same key, and (2) for all probabilistic, polynomial-time adversaries $\mathcal{A}_{BR}$, the advantage* $\mathsf{Adv}^{\mathcal{A}_{\mathsf{BR}}}$ *is negligible and the advantage that any adversary has in violating entity authentication is negligible. An adversary is said to violate entity authentication if some fresh oracle terminates (i.e., accepts a session key and completes the protocol) with no partner.*

2.2 Our Formal Specification Framework

In our formal framework using APA specification, protocol principals are modelled as a family of elementary automata. The various state spaces of the principals are modelled as a family of state sets. The channel through which the elementary automaton communicates is modelled by the addition and removal of messages from the shared state component Network, which is initially empty. Each of the elementary automata only has access to the particular state components to which it is connected. In addition to the regular protocol principals, we specify an adversary $\mathcal{A}$, which has access to the shared state component Network, but no access to the internal states of the principals.

Adopting the adversary formalism from the BPR2000 model, we consider an adversary $\mathcal{A}$ who is able to intercept messages in the Network, swap data

components in the intercepted messages to form new messages, remove messages from the Network, or fabricate new messages. $\mathcal{A}$ is then able to send these messages to the client or server oracles via the Network (corresponding to SendClient and SendServer queries in the BPR2000 model). Also, once an oracle, Π_U^i, has accepted and holds a session key, the (SID, PID) pair associated with that oracle becomes visible to the adversary $\mathcal{A}$ via the shared state component Transcript. If $\mathcal{A}$ so chooses, $\mathcal{A}$ is then able to obtain the session key of Π_U^i via a Reveal query or a Corrupt query. The shared state component Transcript also contains a log of all sent messages and is equivalent to a transcript in the Bellare-Rogaway model. The graphical illustration of MAKEP in APA specification is shown in Figure 2.

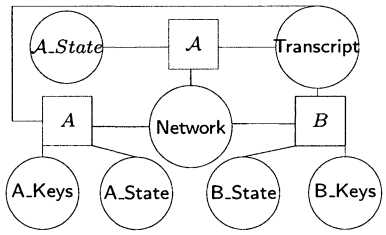

Figure 2. Graphical illustration of MAKEP in APA specification

The advantage of $\mathcal{A}_{BR}$ is not explicitly modelled in our specification due to its probabilistic nature. Instead of modelling the attack to distinguish between the real key and a random value, we simplify the game $\mathcal{G}$ defined in Section 2.1.3 by assuming that $\mathsf{Adv}^{\mathcal{A}_{\mathsf{BR}}} = 1$ if $\mathcal{A}_{BR}$ can obtain a fresh session key, otherwise $\mathsf{Adv}^{\mathcal{A}_{\mathsf{BR}}} = 0$. Consequently, some attacks might be left out while analysing the game $\mathcal{G}$. However, since our aim is to leave computational matters to the human-generated proof, this does not present an obstacle to our protocol analysis.

When using formal specification tools, insecurity is commonly specified in terms of the unreachability of the desired states or reachability of insecure states and a "secure" protocol in a formal specification does not necessarily imply that the protocol is secure. Hence, we find it more natural to define insecurity in our formal framework as given in Definition 4. Protocols proven insecure in our formal specification model will also be insecure in the Bellare-Rogaway model. Definition 4 depends on the notions of partnership in Definition 1 and freshness in Definition 2.

DEFINITION 4 *A protocol is insecure in our formal framework if:*

1 two fresh non-partner oracles accept the same key, or

2 *some fresh oracle accepts some key, which has been exposed (i.e., is known to $\mathcal{A}$), or*

3 *some fresh oracle accepts and terminates with no partner.*

3. Original Version of Jakobsson-Pointcheval MAKEP

Client A (a, g^a)		Server B (b, g^b)
$r_A, t_A \in_R \mathbb{Z}_q$, $K = y_B^{r_A} = g^{br_A}$ $k = h_0(g^b, g^{r_A}, K)$, $r = h_1(g^{t_A})$		
$k_2 = h_2(g^b, g^{r_A}, K)$	$\xrightarrow{g^{r_A}, r}$	$K = (g^{r_A})^b$, $k_2' = h_2(g^b, g^{r_A}, K)$
$k_2 \stackrel{?}{=} k_2'$	$\xleftarrow{k_2', e}$	$e \in_R \{0,1\}^k$
$d = t_A - ae \bmod q$	$\xrightarrow{d}$	$r \stackrel{?}{=} h_1(g^d(g^a)^e)$, $k = h_0(g^b, g^{r_A}, K)$
$k = h_0(g^b, g^{r_A}, K)$		$k = h_0(g^b, g^{r_A}, K)$

Figure 3. Original version of MAKEP

A (a, g^a)	$\mathcal{A}$		B (b, g^b)
$r_A, t_A \in_R \mathbb{Z}_q$ $K = y_B^{r_A} = g^{br_A}$ $k = h_0(g^b, g^{r_A}, K)$ $r = h_1(g^{t_A})$			
$k_2 = h_2(g^b, g^{r_A}, K)$	$\xrightarrow{g^{r_A}, r}$		$K_1 = (g^{r_A})^b$
	$\xrightarrow{g^{r_E}, r}$		$K_2 = (g^{r_E})^b$
		$\xleftarrow{k_{2,1}, e_1}$	$k_{2,1} = h_2(g^b, g^{r_A}, K_1)$ $e_1 \in_R \{0,1\}^k$
		$\xleftarrow{k_{2,2}, e_2}$	$k_{2,2} = h_2(g^b, g^{r_E}, K_2)$ $e_2 \in_R \{0,1\}^k$
$k_2 \stackrel{?}{=} k_{2,1}$	$\xleftarrow{k_{2,1}, e_2}$		
$d = t_A - ae_2 \bmod q$	$\xrightarrow{d}$	$\xrightarrow{d}$	
			$r \stackrel{?}{=} h_1(g^d(g^a)^{e_2})$
$k_{AB} = h_0(g^b, g^{r_A}, (g^b)^{r_A})$			$k_{BA} = h_0(g^b, g^{r_E}, (g^{r_E})^b)$

Figure 4. A hijacking attack on original version of MAKEP

There are two communicating principals in MAKEP shown in Figure 3, namely the server and the client of limited computing resources, A. The security goals of the protocol are mutual authentication and key agreement between the two communicating principals. A and B are each assumed to know the public key of the other party. Prior to the protocol run, A can pre-compute the session key k which is a hash of the shared secret with B using Diffie-Hellman

key exchange, the value r used for client authentication and B's public key (i.e., $k = h_0(g^b, g^{r_A}, K)$). In the protocol, the notation $r_A \in_R \mathbb{Z}_q$ denotes that r_A is randomly drawn from $\mathbb{Z}_q$.

Despite claims of the original version of MAKEP being proven secure in the Bellare-Rogaway model, a hijacking attack on the protocol was discovered by Wong and Chan [18] which breaks the client authentication as shown in Figure 4. The result of the attack is that B actually shares a key with a malicious adversary $\mathcal{A}$ when B believes the key is being shared with A. This attack is revealed by the SHVT analysis explained in Section 4.2.1.

4. Protocol Specification and Analysis

In this section, using the original version of MAKEP as a case study, we specify the protocol using APA. We demonstrate that SHVT can be used to find the hijacking attack first mentioned by Wong and Chan, and two previously unknown flaws in the protocol. For the remainder of this section, E denotes the adversary.

4.1 Protocol Specification

		Examples of some basic types
Agents	::=	set of all the principals (i.e., A, B) and $\mathcal{A}$ (i.e., E)
A_State	::=	A's internal state
A_Keys	::=	set of A's public and private keys
Accepted	::=	set of all oracles who had accepted (visible to $\mathcal{A}$)
		Examples of some functions
gFunction(g, m)	::=	denotes g^m, where m is some value (e.g., gFunction(g, rA) denotes g^{r_A} shown in Figure 3)
verifyGFun(m1, m2)	::=	the verification function used to verify if $g^{m1'} \stackrel{?}{=} g^{m2'}$ for some $m1'$ and $m2'$. (e.g., verifyGFun(gFunction(gFunction(g, a), b), gFunction(gFunction(g, b), a)) will return true)

Figure 5. Examples of basic types and functions

The first phase of our formal specification is to specify the basic types and the functions as shown in Figure 5. In order to increase run-time efficiency, and to overcome storage restrictions, we replace each unique data item in any message with a unique numeric message identifier (MID) in our specification. For example, the message in message flow 1 sent by A consists of two data items, g^{r_A} and r, whose message identifiers are $MID = 1$ and $MID = 2$ respectively. SID is then the concatenation of these unique $MIDs$ (e.g. $SID = [1, 2, \ldots]$) instead of the concatenation of messages from the BPR2000 model.

Initial State of the Original Version. The initial state of MAKEP is shown in Figure 6. The left-hand column shows the SHVT specification of the various initial states, and an explanation is given in the right-hand column.

A_State:=	{(B,server),(start,B), (publicK,gFunction(b),B)};	A knows that B is a server, can start a protocol session run with B (indicated by the keyword *start*), and knows the public key of B (i.e., g^b).
A_Keys:=	{(publicK,gFunction(a)),(privateK,a)};	A owns a key pair (a,g^a).
B_State:=	{(A,agent),(respond,A), (publicK,gFunction(a),A)};	B knows that A is an agent, can respond to a protocol run initiated by A (indicated by the keyword *respond*), and knows the public key of A (i.e., g^a).
B_Keys:=	{(publicK,gFunction(b)),(privateK,b)};	B owns a key pair (b,g^b).
E_State:=	{(publicK,gFunction(a),A),publicK, gFunction(b),B)};	$\mathcal{A}$(Eve) knows the public keys of A and B.
Network:=	$\emptyset$;	Network is initially empty.
Transcript:=	$\emptyset$;	Transcript is initially empty.

Figure 6. Initial state

Step 1 of the Original Version. Starting from the initial state shown in Figure 6, SHVT computes all reachable states. The first state transition of the initiator client A is explained in Figure 7. To ensure uniqueness of the values r_A, t_A and MID in the APA specification, once these values are assigned, they are removed from the pre-defined sets new_random_nonce and $MIDs$. We assume that (SID_A, PID_A) cannot be modified by the adversary. The (SID_A, PID_A) tuple is required to enable the SHVT analysis to define partnership.

A Malicious State Transistion. An active adversary $\mathcal{A}$ is able to intercept message (g^{r_A}, r) meant for B from A, to fabricate a new message (g^{r_E}, r) and to send the fabricated message (g^{r_E}, r) to B via the Network. This state transition as shown in Figure 8 is equivalent to a SendServer query in the BPR2000 model. Due to space contraints, details of other possible state transitions for the adversary and the protocol are omitted.

4.2 Protocol Analysis

Having formally specified the protocol in APA, we analyse the protocol specification using SHVT as shown in the sections below. The analyses were run on a Pentium IV 2.4 GHz computer with 512 Mb of RAM and the anal-

def_trans_pattern A step_1	Definition of a state transition
B,gb,rA,tA,rAA,r,k,k2,K,SIDA,PIDA,MID	Variables used in this step
['start',B] ? A_State,	Precondition: A can start protocol run with B.
[B,'server'] ? A_State,	Precondition: A knows B is the server.
['publicK',gb,B] ? A_State,	Precondition: A knows B's public key gb. (gb is a variable that takes the value $gFunction(b)$.)
rAA « new_random_nonce,	Random unique nonce values are drawn
rA := head(rAA),	from the pre-defined set
tA := head(tail(rAA)),	new_random_nonce and assigned to r_A
tail(tail(rAA)) » new_random_nonce,	and t_A respectively.
MID « SIDs,	Random unique $MIDs$ are drawn from the
tail(tail(MID)) » SIDs,	pre-defined set $SIDs$ are assigned to g^{r_A} and r respectively.
SIDA := [head(MID),head(tail(MID))],	SID is the concatenation of these unique $MIDs$.
PIDA := B,	PID of A is set to B.
K := ['KFunction',gb,rA],	A computes a new $K = (g^b)^{r_A}$.
k := ['hash0',gb,['gFunction',g,rA],K],	A computes the new shared secret k using the hash function h_0 (i.e., $k = h_0(g^b, g^{r_A}, K)$).
r := ['hash1',['gFunction',g,tA]],	A computes $r = h_1(g^{t_A})$.
k2 := ['hash2',gb,['gFunction',g,rA],K],	A computes $k_2 = h_2(g^b, g^{r_A}, K)$.
['start',B] « A_State,	A initiated one session with B, so one tuple enabling a session to start is removed from A's state.
[SIDA,PIDA,[tA,k,k2,K]] » A_State,	A stores the information that she shares with B for this protocol run.
(A,B, [head(SIDA1), head(tail(SIDA1))], [['gFunction',g, rA], r]) » Network;	A sends message g^{r_A}, r to the Network.

Figure 7. State transition - step 1

ysis statistics are shown in Figure 9. We set the break condition to terminate the SHVT analysis if any of the requirement(s) in Definition 4 are violated. The attack sequence and the internal states can be examined by viewing the reachability graph produced by SHVT.

For run-time efficiency, and to avoid enormous branching factors in the search space, we restrict the actions of the adversary so that certain actions are possible for only some message types. Running SHVT with adversaries having various restrictions and also restricting A to only two protocol runs results in SHVT finding the attacks shown in Figures 4, 10, and 11.

def trans pattern E SendServer	Definition of a state transition
(ga,gb,A,B,M,SIDE,S,rE)	Variables used in this step
['publicK',ga,A] ? E_State,	Precondition: $\mathcal{A}$ knows A's public key.
('publicK',gb,B) ? E_State,	Precondition: $\mathcal{A}$ knows B's public key.
[A,'agent'] ? E_State,	Precondition: $\mathcal{A}$ knows A exists.
[B,'server'] ? E_State,	Precondition: $\mathcal{A}$ knows B exists.
A $\neq$ B,	Precondition: A and B are two different principals.
(A,B,S,M) ? Network,	Precondition: Checks if there exists any message from A intended for B in the network.
rAA « new_random_nonce, rA := head(rAA), tA := head(tail(rAA)), tail(tail(rAA)) » new_random_nonce,	Random unique nonce values are drawn from the pre-defined set *new_random_nonce* and assigned to r_A and t_A respectively.
SIDE « SIDs, tail(tail(SIDE)) » SIDs,	Random unique *MIDs* are drawn from the pre-defined set *SIDs* and assigned to grE and rE respectively.
['fabricated','mf1',[head(SIDE),elem(2,S)], [['gFunction',g, head(rE)]]» E_State,	$\mathcal{A}$ stores information in her internal state.
(A,B,[head(SIDE),elem(2,S)], [['gFunction', g,head(rE)], elem(2,M)])» Network;	$\mathcal{A}$ sends a fabricated message to B via the Network.

Figure 8. A malicious state transition

Protocol Analysis	# Players	# Runs	# Nodes	Run-Time	Flaws?
Hijacking Attack	2	1	34	2 secs	Yes
New Attack 1	2	2	144	3 secs	Yes
New Attack 2	2	2	1538	79 secs	Yes

Figure 9. Analysis statistics

4.2.1 Hijacking Attack. State space analysis performed in the SHVT analysis reveals that both requirements 2 and 3 of Definition 4 can be violated. This attack was first mentioned by Wong and Chan [18] as shown in Figure 4.

4.2.2 New Attack 1. State space analysis in SHVT reveals that requirement 1 of Definition 4 is violated. The internal state of the final node in the reachability graph reveals that the following four oracles have accepted some session key: $\Pi_A^{[1,2,7,10,12])}$ belonging to A and having $SID = [1,2,7,10,12]$ accepted $h_0(g^{rA1}, g^{tA1}, (g^{rA1})^{tA1})$, $\Pi_A^{[3,4,9,8,11])}$ belonging to A and having $SID = [3,4,9,8,11]$ accepted $h_0(g^{rA1}, g^{tA2}, (g^{rA1})^{tA2})$, $\Pi_B^{[1,4,7,8,11])}$ belonging to B and having $SID = [1,4,7,8,11]$ accepted $h_0(g^{tA1}, g^{rA1}, (g^{tA1})^{rA1})$, and $\Pi_B^{[3,2,9,10,12])}$ belonging to B and having $SID = [3,2,9,10,12]$ accepted $h_0(g^{tA1}, g^{rA2}, (g^{tA2})^{rA1})$. By Definition 1, none of these oracles have

any partner oracles since their *SIDs* are different. However, we observe that the pairs ($\Pi_A^{[1,2,7,10,12]}$, $\Pi_B^{[1,4,7,8,11]}$) and ($\Pi_A^{[3,4,9,8,11]}$, $\Pi_B^{[3,2,9,10,12]}$) have accepted with the same session keys as shown in Figure 10.

$A\ (a, g^a)$		$\mathcal{A}$	$B\ (b, g^b)$
$r_{A,1}, t_{A,1} \in_R \mathbb{Z}_q$			
$K_{A,1} = y_B^{r_{A,1}} = g^{br_{A,1}}$			
$k_{A,1} = h_0(g^b, g^{r_{A,1}}, K_{A,1})$			
$r_1 = h_1(g^{t_{A,1}})$			
$k_{2(S1(A))} = h_2(g^b, g^{r_{A,1}}, K_{A,1})$	$g^{r_{A,1}}, r_1$		
$r_{A,2}, t_{A,2} \in_R \mathbb{Z}_q$			
$K_{A,2} = y_B^{r_{A,2}} = g^{br_{A,2}}$			
$k_{A,2} = h_0(g^b, g^{r_{A,2}}, K_{A,2})$			
$r_2 = h_2(g^{t_{A,2}})$			
$k_{2(S2(A))} = h_2(g^b, g^{r_{A,2}}, K_{A,2})$	$g^{r_{A,2}}, r_2$	$g^{r_{A,1}}, r_2$	$K_1 = (g^{r_{A(1)}})^b$
		$g^{r_{A,2}}, r_1$	$K_2 = (g^{r_{A(2)}})^b$
		$k_{2,1}, e_1$	$k_{2,1} = h_2(g^b, g^{r_{A,1}}, K_1)$
			$e_1 \in_R \{0,1\}^k$
		$k_{2,2}, e_2$	$k_{2,2} = h_2(g^b, g^{r_{A,2}}, K_2)$
$k_{2(S1(A))} \stackrel{?}{=} k_{2,1}$	$k_{2,1}, e_2$		$e_2 \in_R \{0,1\}^k$
$k_{2(S2(A))} \stackrel{?}{=} k_{2,2}$	$k_{2,2}, e_1$		
$d_1 = t_{A,1} - ae_2 \bmod q$	d_1	d_1	$r_2 \stackrel{?}{=} h_1(g^{d_1}(g^a)^{e_2})$
$d_2 = t_{A,2} - ae_1 \bmod q$	d_2	d_2	$r_1 \stackrel{?}{=} h_1(g^{d_2}(g^a)^{e_1})$
$k_{AB_{(1)}} = h_0(g^b, g^{r_{A,1}}, (g^b)^{r_{A,1}})$			$k_{BA_{(1)}} = h_0(g^b, g^{r_{A,1}}, (g^{r_{A,1}})^b)$
$k_{AB_{(2)}} = h_0(g^b, g^{r_{A,2}}, (g^b)^{r_{A,2}})$			$k_{BA_{(2)}} = h_0(g^b, g^{r_{A,2}}, (g^{r_{A,2}})^b)$

Figure 10. New attack 1

This implies that by revealing one oracle in any pair, the adversary $\mathcal{A}$ is able to distinguish the session key held by the other oracle in the same pair. Hence, the protocol state is not secure since the adversary $\mathcal{A}$ can find a fresh session key. In addition, mutual authentication is violated since both the client and server oracles terminate without a partner.

The attack sequence is shown in Figure 10, and is revealed by following the reachability graph to the insecure state. The attack sequence is as follows: the adversary $\mathcal{A}$ intercepts and removes the two original messages from the Network, swaps the components in these two messages to form two new messages, and sends these two modified messages to B impersonating A via the Network. B, upon receiving these two messages, will respond as per the protocol specification. $\mathcal{A}$ intercepts the messages in protocol flow 2 sent by B to A, and swaps the components in these two messages to form new messages and again sends these two modified messages back to the Network, imperson-

ating B. If A authenticates the server, she will respond with some value d as per the protocol specification. B receives the messages d_1 and d_2 in protocol flow 3. Once some oracle has accepted and holds some session key, the particular (SID, PID) pair will be made visible to the adversary via the shared state component Transcript. $\mathcal{A}$ is then able to send Reveal queries to the oracles of B, and receive the session keys held by the associated fresh oracles of A.

$A\ (a, g^a)$		$\mathcal{A}$	$B\ (b, g^b)$
$r_{A,1}, t_{A,1} \in_R \mathbb{Z}_q$ $K_{A,1} = y_B^{r_{A,1}} = g^{br_{A,1}}$ $k_{A,1} = h_0(g^b, g^{r_{A,1}}, K_{A,1})$ $r_1 = h_1(g^{t_{A,1}})$			
$k_{2(S1(A))} = h_2(g^b, g^{r_{A,1}}, K_{A,1})$	$g^{r_{A,1}}, r_1 \rightarrow$		
$r_{A,2}, t_{A,2} \in_R \mathbb{Z}_q$ $K_{A,2} = y_B^{r_{A,2}} = g^{br_{A,2}}$ $k_{A,2} = h_0(g^b, g^{r_{A,2}}, K_{A,2})$ $r_2 = h_2(g^{t_{A,2}})$			
$k_{2(S2(A))} = h_2(g^b, g^{r_{A,2}}, K_{A,2})$	$g^{r_{A,2}}, r_2 \rightarrow$	$g^{r_{A,1}}, r_E \rightarrow$	$K_1 = (g^{r_{A,1}})^b$
		$g^{r_E}, r_1 \rightarrow$	$K_2 = (g^{r_E})^b$
		$g^{r_E}, r_2 \rightarrow$	$K_3 = (g^{r_E})^b$
		$g^{r_{A,2}}, r_E \rightarrow$	$K_4 = (g^{r_{A,2}})^b$
		$k_{2,1}, e_1 \leftarrow$	$k_{2,1} = h_2(g^b, g^{r_{A,1}}, K_1)$ $e_1 \in_R \{0,1\}^k$
		$\vdots$	$\vdots$
		$k_{2,4}, e_4 \leftarrow$	$k_{2,4} = h_2(g^b, g^{r_{A,2}}, K_4)$
$k_{2(S1(A))} \stackrel{?}{=} k_{2,1}$	$k_{2,1}, e_2 \leftarrow$		$e_4 \in_R \{0,1\}^k$
$k_{2(S2(A))} \stackrel{?}{=} k_{2,4}$	$k_{2,4}, e_3 \leftarrow$		
$d_1 = t_{A,1} - ae_2 \bmod q$	$d_1 \rightarrow$	$d_1 \rightarrow$	$r^E \stackrel{?}{=} h_1(g^{d_1}(g^a)^{e_2})$
$d_2 = t_{A_2} - ae_3 \bmod q$	$d_2 \rightarrow$	$d_2 \rightarrow$	$r^E \stackrel{?}{=} h_1(g^{d_2}(g^a)^{e_3})$
$k_{AB_{(1)}} = h_0(g^b, g^{r_{A,1}}, (g^b)^{r_{A,1}})$			$k_{BA_{(1)}} = h_0(g^b, g^{r_E}, (g^{r_E})^b)$
$k_{AB_{(2)}} = h_0(g^b, g^{r_{A,2}}, (g^b)^{r_{A,2}})$			$k_{BA_{(2)}} = h_0(g^b, g^{r_E}, (g^{r_E})^b)$

Figure 11. New attack 2

4.2.3 New Attack 2. State space analysis in SHVT reveals that requirements 2 and 3 of Definition 4 are violated and the internal state of the final node in the reachability graph reveals that fresh oracles of B, $\Pi_B^{[3,2,5,6,9]}$ and $\Pi_B^{[5,2,7,8,16]}$, have accepted with no partner. In addition, the adversary $\mathcal{A}$ is able to compute both the session keys accepted by B since both session keys are computed based on the random number g^{rE} chosen by the adversary $\mathcal{A}$.

Hence $\mathcal{A}$ is able to decrypt all messages sent by B to A encrypted with these session keys. The attack sequence is shown in Figure 11, and is revealed by following the reachability graph to the insecure state.

5. Conclusion and Future Work

We have described a formal model which can complement computational complexity proofs in the Bellare-Rogaway model. In our model the adversary capabilities match those in the Bellare-Rogaway model. Through a detailed study of the Jakobsson-Pointcheval protocol we have demonstrated that this approach can capture structural flaws in protocols. We were able to find both existing and previously unknown flaws in the protocol using SHVT. Such a tool is useful in checking the hand-generated Bellare-Rogaway proofs. We may speculate that if Jakobsson and Pointcheval had access to such a tool when constructing their original proof of security they could have spotted the flaw in the protocol.

Further directions for this work include extending it to other cryptographic protocols with proofs of security in order to gain better confidence in their correctness. In so doing we should be able to re-use the basic adversary model already developed. We would also like to explore other computational complexity models, in particular the Canetti-Krawczyk modular approach [10], to gain a better understanding of the uses of a complementary model. Finally, we would like to make use of the recent work of Canetti *et al.* [9, 11] and/or Backes *et al.* [4] in order to incorporate abstract cryptographic properties with a sound computational basis.

Acknowledgments

Thanks are due to Dr Carsten Rudolph of Fraunhofer Institute for Secure Telecooperation for his invaluable feedback on earlier drafts of this paper and also the anonymous referees for their critical feedback.

Notes

1. This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.
2. This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.
3. This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.
4. This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.
5. This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.
6. This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.

References

[1] M. Abadi and A.D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *4th ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In *IFIP International Conference on Theoretical Computer Science - IFIP TCS2000*, pages 3–22. Springer-Verlag, 2000.

[3] M. Backes and C. Jacobi. Cryptographically Sound and Machine-Assisted Verification of Security Protocols. In *20th International Symposium on Theoretical Aspects of Computer Science - STACS 2003*, pages 310–329. Springer-Verlag, 2003.

[4] M. Backes, C. Jacobi, and B. Pfitzmann. Deriving Cryptographically Sound Implementations Using Composition and Formally Verified Bisimulation. In *Formal Methods - Getting IT Right*, pages 310–329. Springer-Verlag, 2002.

[5] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Advances in Cryptology – Eurocrypt*, pages 139 – 155. Springer-Verlag, 2000. LNCS Volume 1807/2000.

[6] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology*, pages 110–125. Springer-Verlag, 1993. LNCS Volume 773/1993.

[7] M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *27th ACM Symposium on the Theory of Computing*, pages 57–66. ACM Press, 1995.

[8] C. Boyd and K. Viswanathan. Towards a Formal Specification of the Bellare-Rogaway Model for Protocol Analysis. In *Formal Aspects of Security - FASec 2002*, pages 209–223. British Computer Society Press, Dec 2002.

[9] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, 2000.

[10] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology - Eurocrypt 2001:*, pages 453–474. Springer-Verlag, May 2001. LNCS Volume 2045/2001.

[11] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels (Extended Version). Cryptology ePrint Archive, Report 2002/059, 2002.

[12] D. Dolev and A.C. Yao. On the Security of Public Key Protocols. *IEEE Transaction of Information Technology*, 29(2):198–208, 1983.

[13] M. Jakobsson and D. Pointcheval. Mutual Authentication and Key Exchange Protocol for Low Power Devices. In *Financial Cryptography*, pages 178–195. Springer-Verlag, 2001.

[14] P. Ochsenschlager, J. Repp, and R. Rieke. Abstraction and a Verification Method for Cooperating Systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459, Jun 2000.

[15] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic Security of Reactive Systems. In Steve Schneider and Peter Ryan, editors, *Electronic Notes in Theoretical Computer Science*, volume 32. Reed Elsevier, 2000.

[16] R. Rieke. Implementing the APA Model for the Symmetric Needham-Schroeder Protocol in State Transition Pattern Notation in the SH Verification Tool. Technical report, Fraunhofer Institute for Secure Telecooperation SIT, 26 July 2002.

[17] V. Shoup. OAEP Reconsidered. In *Advances in Cryptology - CRYPTO 2001*, pages 239–259. Springer-Verlag, 2001. LNCS Volume 2139/2001.

[18] D.S. Wong and A.H. Chan. Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices. In *Advances in Cryptology - Asiacrypt 2001*. Springer-Verlag, 2001.