

A Z Based Approach to Verifying Security Protocols

Benjamin W. Long, Colin J. Fidge, and Antonio Cerone

School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, Australia
{benl,cjf,antonio}@itee.uq.edu.au

Abstract. Security protocols preserve essential properties, such as confidentiality and authentication, of electronically transmitted data. However, such properties cannot be directly expressed or verified in contemporary formal methods. Via a detailed example, we describe the phases needed to formalise and verify the correctness of a security protocol in the state-oriented Z formalism.

1 Introduction

Security protocols enable confidential and authenticated electronic transmission of sensitive data. Unfortunately, several such protocols thought to be secure have later been found susceptible to attacks [8], so formal proofs of their correctness are essential.

Security protocols are usually described informally using a ‘standard notation’ [7]. However, this notation provides no semantics or reasoning principles, and can be ambiguous when interpreted in isolation. In contrast, ‘formal methods’ allow for rigorous specification and verification of computer systems in order to verify system correctness. Therefore, several ‘formal methods’ have been promoted for verifying the correctness of security protocols [18]. However, non-functional security concepts such as ‘confidentiality’ and ‘authentication’ [23,15,21] have been found difficult to specify and analyse. Furthermore, formalising security protocols can seem quite intimidating to security practitioners with little or no formal methods experience.

In this paper we present an intuitive Z [25] based approach for formalising and verifying the correctness of the Needham-Schroeder Public Key Protocol [20], from its informal description, thus demonstrating the essential phases required to achieve such a proof.

2 Previous Work

Many formal methods have been proposed for the verification of security protocols [18,16]. This section provides an overview of the methods closely related to our research.

Work already done in formal verification of security protocols is dominated by event-based methods, such as the spi calculus [1] and CSP [22], and logics [21,2,9], including specialised logics such as the BAN logic [5,26]. Other more popular methods include use of a new concept called *strand spaces* [27], and the purpose-built NRL Protocol Analyzer [17]. However, state-based methods such as B, Z, and VDM have seldom been used and we are interested in promoting the application of such methods since their rich data structures enable accurate modelling of message contents.

Previously, Kemmerer [13] used the Ina Jo formalism to analyse cryptographic protocols. The approach focussed on the assumptions and requirements pertaining to the operation of the cryptographic functions used for constructing cryptographic protocol messages.

Boyd [3] presented a formal design for describing secure communication architectures based on the concepts of ‘confidentiality’ and ‘authentication’ using Z. Here, the *communication channel* enabled the concept of message exchange between *users*. His approach was not for the purpose of any protocol in particular but a more general model that may be useful before particular protocols are considered.

Boyd and Kearney [4] explored protocol animation using Z for fair exchange protocols. For the purpose of modelling fair exchange protocols, each agent considers the other to be the intruder. Therefore, the possibility of an external attacker was ignored. Furthermore, the detailed structure of messages used in communication was not defined, thus making it difficult to model attacks made by clever manipulation of protocol messages.

Butler [6] used the B method to formally specify the Needham-Schroeder Protocol, incorporating event-based methodologies such as that of CSP into his model by defining possible event traces. Messages were modelled in some detail, however, an encrypted message was represented informally, merely by including the identity [6] of the agent to whom the key belonged as part of the message.

Inspired by this previous work, our goal is to show how the Z formalism can be used, without modification, to specify and verify security properties merely by careful design and application of appropriate data structures and operations.

3 The Needham-Schroeder Public Key Protocol

Needham and Schroeder [20] explain how public-key cryptography [11] can be used to distribute encryption keys to agents via a trusted third party. This protocol has become a classic example in the literature. Three of the seven steps from the protocol are concerned with authentication between two of the parties and are represented by the commonly used ‘standard notation’ [7] below.

1. $A \longrightarrow B : \{N_A, A\}_{K_B}$
2. $B \longrightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \longrightarrow B : \{N_B\}_{K_B}$

Firstly, agent Antonio sends a message (step 1) to Ben consisting of a *nonce* N_A (a unique datum used for only one protocol run) [12] and his identity A

encrypted with the public key of the receiver (in this case Ben's public key K_B). This message indicates that someone claiming to be Antonio wishes to establish communication with Ben. At this point Ben does not know that Antonio actually created the message because an intruder could have forged it.

Ben replies by sending a message (step 2) consisting of the nonce received, and also a new nonce N_B generated by him, and encrypts both with the public key of the agent whose identity was part of the received message (in this case Antonio's public key K_A).

If the nonce that Antonio sent as part of the initial message in step 1 is in the message sent in step 2, Antonio knows that Ben received and decrypted the initial message because Ben is the only agent that can decrypt a message that has been encrypted with key K_B , and we assume that it is impossible for any other agent to forge the nonce N_A . (An agent's public key is known by everyone — messages encrypted with it can be decrypted only by the agent's secret private key [11].) However, Antonio can authenticate Ben as the creator of the message only if Antonio trusts that Ben will not reveal Antonio's nonce to any other agent. This requirement forms part of an invariant on the system that we will discuss in Section 7.

Antonio sends Ben's nonce back to him encrypted with Ben's public key (step 3). Ben then knows that Antonio received and decrypted the message sent in step 2. Again, Ben can authenticate the creator of the message only if he trusts that Antonio will not reveal Ben's nonce to any other agent. By receiving this message Ben can also assume that the initial message was from Antonio.

4 Formalising the Protocol

Based on the informal description above, we now illustrate the phases needed to accurately model this protocol in the Z notation [25], via its rich set of mathematical operators.

4.1 Define a Suitable Set of Data Types

The first phase in formalising the protocol is to define a suitable set of data structures that capture the types of data used in the standard notation description. We note that the left-hand side of the standard notation displays the agents whom the messages are being sent between, and the right-hand side displays the messages.

We define *AGENT* to be the set of all agents in the network, including Antonio (*A*), Ben (*B*), Colin (*C*) and a special symbol ' \perp ' to represent 'no agent'.

$$AGENT ::= A \mid B \mid C \mid \perp$$

The protocol messages are constructed from several data items. Therefore we assume the set of all such items as a given type.

[*ITEM*]

Given the set of all items, the set of all messages *MSG* is the set of all possible sequences of items.

$$MSG == \text{seq } ITEM$$

In the protocol, there are nonce, key, and address items. These can be encrypted individually, or in combination with one or more other items, producing a single encrypted item. To allow for these different types of data, we declare the following four subsets of *ITEM* as an ‘axiomatic’, global Z definition.

$NON : \mathbb{P} \textit{ITEM}$ $KEY : \mathbb{P} \textit{ITEM}$ $ADR : \mathbb{P} \textit{ITEM}$ $ENC : \mathbb{P} \textit{ITEM}$	$\text{disjoint}\langle NON, KEY, ADR, ENC \rangle$ $NON \cup KEY \cup ADR \cup ENC = \textit{ITEM}$
--	---

Z’s ‘disjoint’ operator is used to ensure that the sets are pairwise disjoint, i.e., each element within a set is not an element of any other set. For completeness we also specify that each element in *ITEM* must be in one of the newly declared sets.

4.2 Define Supporting Functions

The second phase in formalising the protocol is to define operations on the components of messages. The method of encryption used in the protocol is public key encryption [11]. Therefore we introduce subsets of public and private keys from the set *KEY* of all keys.

$PUB : \mathbb{P} \textit{KEY}$ $PRV : \mathbb{P} \textit{KEY}$ $pair : \textit{KEY} \twoheadrightarrow \textit{KEY}$	$\text{disjoint}\langle PUB, PRV \rangle$ $PUB \cup PRV = \textit{KEY}$ $pair = pair^\sim$ $\forall k : \textit{KEY} \bullet k \in PRV \Leftrightarrow pair(k) \in PUB$
---	--

The set *PUB* is the set of public keys, and *PRV* is the set of private keys. The first two predicates in the schema above state that the sets are pairwise disjoint and that every key may only be a public, or a private key. The total bijective function *pair* is introduced to define a one-to-one symmetric correspondence between keys. Symmetry is ensured by the predicate $pair = pair^\sim$ (that is, *pair* is identical to its inverse). Thus if the pair (k_1, k_2) exists in the *pair* function, then the pair (k_2, k_1) also exists in the *pair* function. The last predicate specifies that

each private key corresponds to a public key and vice versa. Such a predicate, in combination with the symmetry of the *pair* function, ensures that each pair of keys in the *pair* function consists of one private key and one public key.

With these definitions, we now model important properties of the encrypt function *enc* and decrypt function *dec*. The encrypt function maps a key and a message to a unique encrypted item. The decrypt function maps a key *k* and a message *m* (that may contain encrypted items) to a message containing all items from *m* that are either non-encrypted or encrypted using a key other than *k*, plus items extracted from all encrypted items in *m* using *k*.

$enc : (KEY \times MSG) \mapsto ENC$ $dec : (KEY \times MSG) \mapsto MSG$	
$\forall k, \ell : KEY; m, m', m'' : MSG; s : ITEM \bullet$	
$dec(k, \langle \rangle) = \langle \rangle \wedge$	[1]
$((m = \langle s \rangle \wedge m' \wedge s \notin ENC) \Rightarrow$	[2]
$dec(k, m) = \langle s \rangle \wedge dec(k, m')) \wedge$	
$((m = \langle s \rangle \wedge m' \wedge s = enc(\ell, m'') \wedge \ell \neq pair(k)) \Rightarrow$	[3]
$dec(k, m) = \langle s \rangle \wedge dec(k, m')) \wedge$	
$(m = \langle enc(pair(k), m'') \rangle \wedge m' \Rightarrow$	[4]
$dec(k, m) = dec(k, m'') \wedge dec(k, m'))$	

The use of total injective functions ensure uniqueness and that all combinations of keys and messages have a mapping associated with them. Since the *enc* function maps an entire message (sequence of items) to a single encrypted item, arbitrary nesting of encrypted messages is allowed.

The predicates define the recursive nature of the decryption function. Given a message *m* and a key *k*, there are four possible cases depending on the structure of message *m*. Firstly, if the message is empty, then the result of the decryption is an empty message (conjunct 1). If the first item *s* in the given message is not an encrypted item (conjunct 2) or if the first item was encrypted using a key *ℓ* that does not correspond to the given key *k* (conjunct 3), then the item is unchanged and the result of the decryption is sequence $\langle s \rangle$ concatenated with the decryption of the remainder *m'* of the message. Lastly, if the first item of the given message is an encrypted item that was created using the key that corresponds (via function *pair*) to the given key (conjunct 4), then the result is the decryption of the secret message *m''* from the encrypted item, concatenated with the decryption of the remainder of the message.

The use of the encrypt and decrypt functions together with the *pair* function allow:

- encryption using a public key ($pair(k) \in PUB$) and decryption using the corresponding private key ($k \in PRV$); and
- encryption using a private key ($pair(k) \in PRV$) and decryption using the corresponding public key ($k \in PUB$).

In the protocol, Antonio makes use of his address, Antonio and Ben make use of their nonces, and all three agents make use of their keys. For simplification, we can thus assume that each agent possesses one of each of these items. The declarations below include four total functions that map each agent¹ to a unique public key, private key, nonce and address. The predicate ensures that the pair of keys associated with each agent G are a matching pair in the *pair* function.

$$\frac{\begin{array}{l} pub : AGENT \mapsto KEY \\ prv : AGENT \mapsto KEY \\ non : AGENT \mapsto NON \\ adr : AGENT \mapsto ADR \end{array}}{\forall G : AGENT \bullet pub(G) = pair(prv(G))}$$

4.3 Define the Global State

The third phase in formalising the protocol is to define its state space. The *InTransit* Z state schema below contains the content of the communications medium. It is often assumed in security analysis that protocol instances are independent, so modelling one instance is sufficient. Furthermore, we can also assume that one message only is in transit at a time for this protocol. The *to* and *from* variables represent whom the message is to and whom the message is from, respectively. (We use the ‘no agent’ value \perp as the *to* address to indicate that no message is in transit.) These variables hold the agent identifiers corresponding to those on the left-hand side of the standard notation steps. The *msg* variable represents the content of the message, if any, corresponding to the message value in the standard notation steps.

$$\frac{InTransit}{\begin{array}{l} to : AGENT \\ from : AGENT \\ msg : MSG \end{array}}$$

To initialise the protocol, we merely need to state that no message is in transit, by setting the destination address to ‘ \perp ’, in the following Z operation schema. An operation schema consists of a declaration part, above the line, and a predicate part, below. In this case the declaration part inherits the three global variable declarations from the *InTransit* state schema.

$$\frac{\begin{array}{l} Init \\ InTransit \end{array}}{to = \perp}$$

¹ To ensure that all predicates are well-defined, ‘no agent’ \perp has dummy keys and a nonce, but these are never used.

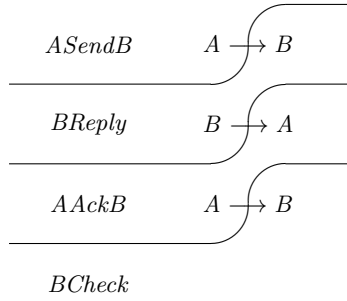


Fig. 1. Grouping of Needham-Schroeder protocol steps.

4.4 Specify the Agents’ Operations

The fourth phase in formalising the protocol is to model its dynamic behaviour as a set of Z operations. A direct interpretation of the standard notation description above would urge us to model this protocol as three atomic operations corresponding to the three steps modelled by the standard notation: one for Antonio sending the first message to Ben, another for Ben’s acknowledgement, and a final operation for Antonio’s reply. However, we want to allow for the modelling of intrusions made whilst a message is in transit, and this particular partitioning would make it difficult to interpose an intrusion between a message’s transmission and its subsequent receipt. Alternatively, the protocol could be broken into six operations — three sending operations and three receiving operations. However, assuming that an intruder cannot interfere with an agent’s internal operations, we can more concisely model an agent’s receipt and response to a message as part of a single ‘atomic’ operation [10,24]. Therefore we group the operations such that Ben receives and then sends his message within one operation, and also so that Antonio receives and acknowledges Ben’s reply within one operation (see Fig. 1). Consequently, we can adequately model the whole protocol as four operations in Z: *ASendB*, *BReply*, *AAckB*, and *BCheck*.

Antonio Sends the First Message to Ben. We have defined the first operation specifically for Antonio sending the initial message to Ben. The message consists of Antonio’s nonce *non(A)* and his address *adr(A)*, and is encrypted using Ben’s public key *pub(B)* ². This operation is modelled in Z as follows:

$$\begin{array}{|l}
 \hline
 \textit{ASendB} \\
 \hline
 \Delta \textit{InTransit} \\
 \hline
 to = \perp \wedge to' = B \wedge from' = A \\
 msg' = \langle enc(pub(B), \langle non(A), adr(A) \rangle) \rangle \\
 \hline
 \end{array}$$

² We assume that all public keys are known to all agents.

In the declaration part, Z 's ' Δ ' annotation is used to state that the variables in schema *InTransit* may change value by this operation. The undecorated (pre-state) variables specify the value of variables before the operation. The primed (post-state) variables specify the value of the variables after the operation. We want this operation to be applied only if there is no message in transit. This is checked by stating that the '*to*' address in the pre-state is no agent ' \perp '. The post-state variables *to'* and *from'* are set to indicate that after the operation the message in transit is to Ben and from Antonio. The post-state value *msg'* of the message contains an encrypted item made from the appropriate structure of the message sent in step 1 of the protocol. The encrypted item is inside Z 's sequence brackets ' $\langle \cdot \cdot \cdot \rangle$ ' because a message always consists of a sequence of items, even though it contains only one item in this case. After this operation, Antonio is waiting for a reply message.

Ben Replies to the Message. The operation *BReply* is a generalised operation where Ben receives a message *msg* from an unknown agent X and replies to this agent. The reply message *msg'* contains the nonce N received and Ben's nonce. As the recipient of the message, Ben cannot control who it will come from, and thus accepts a message from any valid agent X . Similarly, Ben does not know the value of the incoming nonce and so uses an arbitrary nonce N for this value.

$$\begin{array}{l}
 \text{---} \\
 \text{---} \text{BReply} \text{---} \\
 \text{---} \Delta \text{InTransit} \text{---} \\
 \text{---} \\
 to = B \wedge from' = B \\
 (\exists X : \text{AGENT}; N : \text{NON} \bullet to' = X \wedge \\
 \quad msg = \langle enc(pub(B), \langle N, adr(X) \rangle) \rangle \wedge \\
 \quad msg' = \langle enc(pub(X), \langle N, non(B) \rangle) \rangle) \\
 \text{---}
 \end{array}$$

The implicit precondition of the operation states that there is a message for Ben, the message is encrypted with Ben's public key (in other words Ben can decrypt it), and that the secret content of the message consists of a nonce N and an address *adr*(X). Ben uses the arbitrary identity X as his way of identifying the unknown sender. Hence, the reply message, consisting of the nonce N and Ben's nonce *non*(B), is encrypted using X 's public key *pub*(X).

Antonio Acknowledges the Message. In the next operation Antonio authenticates Ben's identity and sends the newly received nonce back to Ben in order to be authenticated by him.

$$\begin{array}{l}
 \text{---} \\
 \text{---} \text{AAckB} \text{---} \\
 \text{---} \Delta \text{InTransit} \text{---} \\
 \text{---} \\
 to = A \wedge to' = B \wedge from' = A \\
 (\exists N : \text{NON} \bullet msg = \langle enc(pub(A), \langle non(A), N \rangle) \rangle \wedge \\
 \quad msg' = \langle enc(pub(B), \langle N \rangle) \rangle) \\
 \text{---}
 \end{array}$$

After sending the initial message, Antonio expects a message of a particular form from Ben. He knows that the message should contain the nonce that he sent in the initial message and another nonce N , which Antonio assumes to belong to Ben. This is checked as part of the implicit precondition. Antonio replies to this message by sending a reply back to Ben with the nonce assumed to be Ben's, N , and encrypts the message using Ben's public key, $pub(B)$. The fact that Antonio assumes that the nonce is from Ben is the weakness of this protocol and is demonstrated when the intruder's operations are introduced in Section 5.

Ben Checks the Message. Ben is now expecting a message of a particular form. Again this is implicitly checked as part of the required pre-state of the following operation which ensures that Ben's nonce is part of the incoming message, therefore allowing Ben to authenticate the identity of the agent he believes he is communicating with.

$$\frac{\frac{BCheck}{\Delta InTransit}}{to = B \wedge to' = \perp \wedge msg = \langle enc(pub(B), \langle non(B) \rangle) \rangle}$$

Ben's receipt of the final message is modelled by setting the to variable in the post-state to 'no agent', allowing the protocol to start again.

5 An Attack on the Protocol

Lowe [14] identified an intrusion on this protocol whereby Antonio honestly communicates with Colin C (the intruder) not knowing that he has malicious intentions. Colin is able to masquerade as Antonio by sending modified messages to Ben.

1. $A \longrightarrow C : \{N_A, A\}_{K_C}$
2. $C_A \longrightarrow B : \{N_A, A\}_{K_B}$
3. $B \longrightarrow C_A : \{N_A, N_B\}_{K_A}$
4. $C \longrightarrow A : \{N_A, N_B\}_{K_A}$
5. $A \longrightarrow C : \{N_B\}_{K_C}$
6. $C_A \longrightarrow B : \{N_B\}_{K_B}$

When Ben receives the message from Colin in step 2, he believes that Antonio is initiating an instance of the protocol because Antonio's identity is in the message. He returns the message (step 3) to Antonio, following the protocol by encrypting the message with the key K_A of the agent whose identity was in the message. At this point Colin intercepts the message, but as he can't decrypt it, he merely forwards the message (step 4) to Antonio. Antonio believes that the nonce in the message belongs to Colin so he sends it back (step 5) to Colin for authentication. As Antonio sends Ben's nonce to Colin, Ben should not have

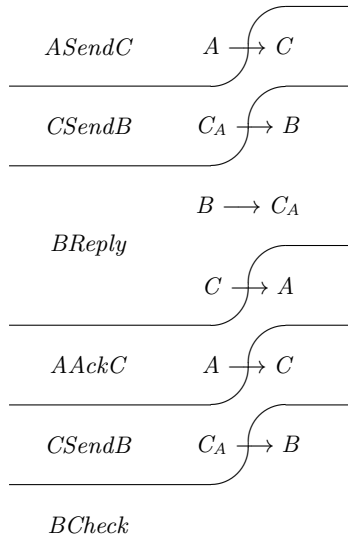


Fig. 2. Grouping of protocol steps with the intrusion.

trusted Antonio. Now Colin can decrypt the message to gain access to Ben’s nonce. Colin then sends Ben’s nonce (step 6) to Ben to complete the protocol and to hide the intrusion.

6 Formalising the Attack

The fifth phase in analysing the protocol is to model the attacker’s behaviour and any other operations required to enable the intruder’s involvement in the protocol.

We explained above that Colin intercepts the message in step 3 but is unable to gain anything from it in its encrypted form. As he forwards the message without modification to Antonio in step 4, there is no change to the state. Therefore, we choose to ignore this operation in the sequence of protocol steps for the intrusion in our Z model. With this in mind we group the standard notation steps to create six Z operation schemas for modelling the intrusion as shown in Figure 2. Fortunately, we do not need to redefine Ben’s operations as they are general enough for interaction with any agent.

The first operation needed to model the intrusion is $ASendC$ which is similar to the $ASendB$ operation, but where Antonio sends the initial message to Colin instead of Ben.

$ASendC$ $\Delta InTransit$
$to = \perp \wedge to' = C \wedge from' = A$ $msg' = \langle enc(pub(C), \langle non(A), adr(A) \rangle) \rangle$

The next operation is new and models Colin taking the message sent to him, encrypting it with Ben’s public key, and sending it to Ben. This generic operation is used in both steps 2 and 6 when Colin sends a message M to Ben.

$$\begin{array}{c}
 \frac{CSendB}{\Delta InTransit} \\
 \hline
 to = C \wedge to' = B \wedge from' = C \\
 (\exists M : MSG \bullet msg = \langle enc(pub(C), M) \rangle \wedge msg' = \langle enc(pub(B), M) \rangle)
 \end{array}$$

Finally, we also need to define the operation $AAckC$, which is similar to $AAckB$ except that Antonio is interacting with Colin instead of Ben.

$$\begin{array}{c}
 \frac{AAckC}{\Delta InTransit} \\
 \hline
 to = A \wedge to' = C \wedge from' = A \\
 (\exists N : NON \bullet msg = \langle enc(pub(A), \langle non(A), N \rangle) \rangle \wedge \\
 msg' = \langle enc(pub(C), \langle N \rangle) \rangle)
 \end{array}$$

7 Verification of the Protocol

7.1 Specify Desired Property

The sixth phase in analysing the protocol is to specify properties that must be preserved for the protocol to operate securely. Previously, Butler [6] incorporated a variable containing nonces that are *critical* into his model. He specifies a property (invariant) stating that any critical nonces are to remain secret. With a run of the original flawed Needham-Schroeder Protocol in the presence of an intruder, this clause no longer holds, because at the end of the protocol, the protocol’s critical nonce is no longer secret. Hence, the protocol is proven to be insecure.

We suggest that a simple property similar to Butler’s should apply to nonces contained within a message to ensure confidentiality in authentication protocols. Each honest agent must be able to trust other honest agents to use nonces securely. Given this fact, our property is that each honest agent must not reveal a nonce to another agent unless it belongs to either the sender or the receiver. If it belongs to the sender then we may assume that the sender wants it to become a secret between the sender and the other agent, and if it belongs to the receiver, there is no harm in sending it to him. We restrict the invariant to honest agents only, because we know that an intruder can always choose to violate security properties, but we want to ensure that each honest agent always aims to maintain such a property. (An intruder should not be able to force an honest agent to violate security properties either.) The advantage of our invariant is that no extra variable containing critical nonces such as in Butler’s

model is required. For the Needham-Schroeder Protocol, it can be expressed by the following invariant.

$$\begin{array}{|l}
 \hline
 \textit{Inv} \\
 \hline
 \textit{InTransit} \\
 \hline
 (to \neq \perp \wedge from \in \{A, B\}) \Rightarrow \\
 \{n : \textit{NON} \mid n \text{ in } dec(prv(to), msg)\} \subseteq \{non(from), non(to)\} \\
 \hline
 \end{array}$$

The invariant states that if there is a message in transit, which is sent from an honest agent (Antonio or Ben), and the recipient can decrypt it to reveal the secret content, then all ‘decryptable’ nonces in the message either belong to the sender or the recipient. *Z*’s ‘in’ operator checks that the value on its left is in the sequence on its right [25].

7.2 Formal Proof

The seventh and final phase is to verify that the invariant is preserved by the protocol. This could be done by separately analysing each operation and proving that, in isolation, it preserves the invariant. However, this is an unnecessarily strong goal because it requires us to show that the operations preserve the invariant even from unreachable states. Instead, we show that the operations preserve the invariant when sequentially composed in their intended ordering, thus proving the weaker, but sufficient, goal that they preserve the invariant in their intended context.

In the flawed version of the protocol described above, after the execution of *AAckC*, the invariant does not hold because there is a message from Antonio to Colin that is decrypted by Colin and contains Ben’s nonce, which does not belong to either the sender or the receiver. To formally prove that the invariant does not hold at this point, we use *Z*’s schema calculus [25]. We construct a schema from the composition of *ASendC*, *CSendB*, *BReply*, and *AAckC*, and prove that this sequence of operations contradicts the invariant.

Given two schemas *S* and *T*, the composition ‘*S* \circ *T*’ of these schemas is the conjunction of the two where there exists an intermediate state which satisfies both the post-state of *S* and the pre-state of *T* [25]. This angelic form of composition provides a suitable basis for exploring the security implications of *potential* sequences of protocol steps. (We are not attempting to prove that the sequence of steps *can* be performed successfully, in which case a demonic composition operator would be needed.)

After initialisation, only *ASendC* is enabled because $to = \perp$. In fact, it is easy to check that only one operation is enabled at any stage of the protocol sequence in our model. As *CSendB* is the enabled operation after *ASendC* has been performed, we construct this particular sequence. Using the schema composition operator, the composition of schemas *ASendC* and *CSendB* is as follows.

$$\begin{array}{c}
 \hline
 A\text{Send}C \ ; \ C\text{Send}B \\
 \hline
 \Delta\text{InTransit} \\
 \hline
 to = \perp \wedge to' = B \wedge from' = C \\
 (\exists \text{InTransit}'' \bullet to'' = C \wedge from'' = A \wedge \\
 \quad msg'' = \langle enc(pub(C), \langle non(A), adr(A) \rangle) \rangle \wedge \\
 \quad (\exists M : MSG \bullet msg'' = \langle enc(pub(C), M) \rangle \wedge \\
 \quad \quad msg' = \langle enc(pub(B), M) \rangle))
 \end{array}$$

To simplify this complicated schema, the nested quantifier can be removed by application of the one-point law [19] because we know that M must be $\langle non(A), A \rangle$.

$$\begin{array}{c}
 \hline
 A\text{Send}C \ ; \ C\text{Send}B \\
 \hline
 \Delta\text{InTransit} \\
 \hline
 to = \perp \wedge to' = B \wedge from' = C \\
 msg' = \langle enc(pub(B), \langle non(A), adr(A) \rangle) \rangle \\
 (\exists \text{InTransit}'' \bullet to'' = C \wedge from'' = A \wedge \\
 \quad msg'' = \langle enc(pub(C), \langle non(A), adr(A) \rangle) \rangle)
 \end{array}$$

Now that none of the pre or post-state variables depend on the doubly primed variables, we can also remove these variables and the remaining quantifier.

$$\begin{array}{c}
 \hline
 A\text{Send}C \ ; \ C\text{Send}B \\
 \hline
 \Delta\text{InTransit} \\
 \hline
 to = \perp \wedge to' = B \wedge from' = C \\
 msg' = \langle enc(pub(B), \langle non(A), adr(A) \rangle) \rangle
 \end{array}$$

Next we compose this schema with $B\text{Reply}$.

$$\begin{array}{c}
 \hline
 A\text{Send}C \ ; \ C\text{Send}B \ ; \ B\text{Reply} \\
 \hline
 \Delta\text{InTransit} \\
 \hline
 to = \perp \wedge from' = B \\
 (\exists \text{InTransit}'' \bullet from'' = C \wedge to'' = B \wedge \\
 \quad msg'' = \langle enc(pub(B), \langle non(A), adr(A) \rangle) \rangle \wedge \\
 \quad (\exists X : AGENT; N : NON \bullet to' = X \wedge \\
 \quad \quad msg'' = \langle enc(pub(B), \langle N, adr(X) \rangle) \rangle \wedge \\
 \quad \quad msg' = \langle enc(pub(X), \langle N, non(B) \rangle) \rangle))
 \end{array}$$

By application of the one-point law we can simplify the schema because we know that agent X must be A , and nonce N must be $non(A)$.

$$\begin{array}{c}
\frac{ASendC \circ CSendB \circ BReply}{\Delta InTransit} \\
\hline
to = \perp \wedge to' = A \wedge from' = B \\
msg' = \langle enc(pub(A), \langle non(A), non(B) \rangle) \rangle \\
(\exists InTransit'' \bullet from'' = C \wedge to'' = B \wedge \\
\quad msg'' = \langle enc(pub(B), \langle non(A), adr(A) \rangle) \rangle)
\end{array}$$

The remaining quantifier and doubly primed variables can be removed since the existence of such values is obvious.

$$\begin{array}{c}
\frac{ASendC \circ CSendB \circ BReply}{\Delta InTransit} \\
\hline
to = \perp \wedge to' = A \wedge from' = B \\
msg' = \langle enc(pub(A), \langle non(A), non(B) \rangle) \rangle
\end{array}$$

Finally we compose this schema with $AAckC$.

$$\begin{array}{c}
\frac{ASendC \circ CSendB \circ BReply \circ AAckC}{\Delta InTransit} \\
\hline
to = \perp \wedge to' = C \wedge from' = A \\
(\exists InTransit'' \bullet to'' = A \wedge from'' = B \wedge \\
\quad msg'' = \langle enc(pub(A), \langle non(A), non(B) \rangle) \rangle) \wedge \\
(\exists N : NON \bullet msg'' = \langle enc(pub(A), \langle non(A), N \rangle) \rangle) \wedge \\
\quad msg' = \langle enc(pub(C), \langle N \rangle) \rangle)
\end{array}$$

Once again we can simplify the schema because we know that nonce N is $non(B)$.

$$\begin{array}{c}
\frac{ASendC \circ CSendB \circ BReply \circ AAckC}{\Delta InTransit} \\
\hline
to = \perp \wedge to' = C \wedge from' = A \wedge msg' = \langle enc(pub(C), \langle non(B) \rangle) \rangle
\end{array}$$

Using this schema, we will prove that this sequence of operations does not maintain the invariant. To do this we show that, assuming the invariant holds before the operations, the invariant does not hold afterwards. This is expressed by the following schema.

$$\begin{array}{c}
\frac{Inv \wedge ASendC \circ CSendB \circ BReply \circ AAckC}{\Delta InTransit} \Rightarrow \neg Inv' \\
\hline
(((to \neq \perp \wedge from \in \{A, B\}) \Rightarrow \\
\quad \{n : NON \mid n \text{ in } dec(prv(to), msg)\} \subseteq \{non(from), non(to)\}) \wedge \\
to = \perp \wedge to' = C \wedge from' = A \wedge msg' = \langle enc(pub(C), \langle non(B) \rangle) \rangle) \\
\Rightarrow \\
\neg(((to' \neq \perp \wedge from' \in \{A, B\}) \Rightarrow \\
\quad \{n : NON \mid n \text{ in } dec(prv(to'), msg')\} \subseteq \{non(from'), non(to')\}))
\end{array}$$

We simplify this schema by eliminating the negation in the consequent of the schema.

$$\frac{
 \begin{array}{l}
 \text{Inv} \wedge \text{ASendC} \wp \text{CSendB} \wp \text{BReply} \wp \text{AAckC} \Rightarrow \neg \text{Inv}' \\
 \Delta \text{InTransit}
 \end{array}
 }{
 \begin{array}{l}
 (((to \neq \perp \wedge from \in \{A, B\}) \Rightarrow \\
 \{n : NON \mid n \text{ in } dec(prv(to), msg)\} \subseteq \{non(from), non(to)\}) \wedge \\
 to = \perp \wedge to' = C \wedge from' = A \wedge msg' = \langle enc(pub(C), non(B)) \rangle) \\
 \Rightarrow \\
 ((to' \neq \perp \wedge from' \in \{A, B\}) \wedge \\
 \{n : NON \mid n \text{ in } dec(prv(to'), msg')\} \not\subseteq \{non(from'), non(to')\})
 \end{array}
 }$$

We distinguish two cases.

- If $to \neq \perp$, then the overall antecedent (the first three lines of the predicate above) is false and the whole schema is trivially true.
- If $to = \perp$, then the overall antecedent is simplified as follows:

$$to = \perp \wedge to' = C \wedge from' = A \wedge msg' = \langle enc(pub(C), non(B)) \rangle \quad (1)$$

We distinguish two subcases.

- If predicate 1 is false, then the whole schema is trivially true.
- If predicate 1 is true, then both $to' \neq \perp$ and $from' \in \{A, B\}$ hold and the overall consequent (last two lines) can be simplified, in the context of predicate 1, as follows:

$$\{n : NON \mid n \text{ in } dec(prv(to'), msg')\} \not\subseteq \{non(from'), non(to')\} \quad (2)$$

Since we have assumed that predicate 1 is true, we can evaluate predicate 2 by replacing $from'$ with A and to' with C and applying the decrypt function to the private key $prv(C)$ and the message in transit $msg' = \langle enc(pub(C), non(B)) \rangle$.

Predicate 2 is then further simplified as follows:

$$\{non(B)\} \not\subseteq \{non(A), non(C)\} \quad (3)$$

Predicate 3 holds trivially and therefore the whole schema is true.

This completes the proof that the invariant is not maintained and the protocol is not secure. The proof relies on basic predicate logic only, and could be performed easily using a theorem prover.

8 The Fixed Needham-Schroeder Protocol

It is suggested by Lowe [14] that the protocol will operate securely if Ben's identity is included in the message he sends back to Antonio, i.e., if the message

in Ben's reply is $\{N_A, N_B, B\}_{K_A}$. Then Antonio will be able to check the identity of the agent that created the message, which should be the agent with whom he is communicating. There can be no deception by an intruder modifying a message and claiming that it belongs to him. Furthermore, since the message is encrypted with Antonio's key, the intruder cannot insert his own address into the message.

To confirm this, we can repeat the phases above using the new version of the protocol, starting with the fourth phase. The operations that change to incorporate the new message structure are $BReply$ and $AAckC$. Both play an important part in fixing the original protocol. The only difference in the new operation $BReply^*$ is that B adds his identity to msg' to conform to the new protocol.

$$\frac{BReply^* \quad \Delta InTransit}{\begin{array}{l} to = B \wedge from' = B \\ (\exists X : AGENT; N : NON \bullet to' = X \wedge \\ \quad msg = \langle enc(pub(B), \langle N, adr(X) \rangle) \rangle \wedge \\ \quad msg' = \langle enc(pub(X), \langle N, non(B), adr(B) \rangle) \rangle) \end{array}}$$

Now that the responding agent's identity is part of the message, Antonio has the opportunity to check that this identity corresponds to the agent he is communicating with. So part of the pre-state for $AAckC^*$ is that Colin's identity is in the message.

$$\frac{AAckC^* \quad \Delta InTransit}{\begin{array}{l} to = A \wedge to' = C \wedge from' = A \\ (\exists N : NON \bullet msg = \langle enc(pub(A), \langle non(A), N, adr(C) \rangle) \rangle \wedge \\ \quad msg' = \langle enc(pub(C), \langle N \rangle) \rangle) \end{array}}$$

Repeating the seventh phase in the analysis, we now show that each sequence of operations before $AAckC^*$ maintains the invariant and that the precondition of $AAckC^*$ is violated after $BReply^*$, therefore indicating that $AAckC^*$ must not be performed. Firstly we show that $ASendC$ maintains the invariant.

$$\frac{Inv \wedge ASendC \Rightarrow Inv' \quad \Delta InTransit}{\begin{array}{l} (((to \neq \perp \wedge from \in \{A, B\}) \Rightarrow \\ \quad \{n : NON \mid n \text{ in } dec(prv(to), msg)\} \subseteq \{non(from), non(to)\}) \wedge \\ to = \perp \wedge to' = C \wedge from' = A \wedge \\ msg' = \langle enc(pub(C), \langle non(A), adr(A) \rangle) \rangle) \\ \Rightarrow \\ ((to' \neq \perp \wedge from' \in \{A, B\}) \Rightarrow \\ \quad \{n : NON \mid n \text{ in } dec(prv(to'), msg')\} \subseteq \{non(from'), non(to')\}) \end{array}}$$

If $to \neq \perp$, then the overall antecedent is false and the whole schema is true. If $to = \perp$, the first conjunct is trivially true. The schema can be simplified by application of the one-point rule and the decrypt function.

$$\begin{array}{c}
 \frac{Inv \wedge ASendC \Rightarrow Inv'}{\Delta InTransit} \\
 \hline
 (to = \perp \wedge to' = C \wedge from' = A \wedge \\
 msg' = \langle enc(pub(C), \langle non(A), adr(A) \rangle) \rangle) \\
 \Rightarrow \\
 \{non(A)\} \subseteq \{non(A), non(C)\}
 \end{array}$$

As the consequent is true, the whole schema is trivially true thus confirming that the invariant is maintained. In other words, honest agent Antonio respects the invariant.

Using $ASendB \ ; \ CSendB$ calculated in Section 7, we prove that the sequence of these two operations maintains the invariant.

$$\begin{array}{c}
 \frac{Inv \wedge ASendC \ ; \ CSendB \Rightarrow Inv'}{\Delta InTransit} \\
 \hline
 (((to \neq \perp \wedge from \in \{A, B\}) \Rightarrow \\
 \{n : NON \mid n \text{ in } dec(prv(to), msg)\} \subseteq \{non(from), non(to)\}) \wedge \\
 to = \perp \wedge to' = B \wedge from' = C \wedge \\
 msg' = \langle enc(pub(B), \langle non(A), adr(A) \rangle) \rangle) \\
 \Rightarrow \\
 ((to' \neq \perp \wedge from' \in \{A, B\}) \Rightarrow \\
 \{n : NON \mid n \text{ in } dec(prv(to'), msg')\} \subseteq \{non(from'), non(to')\})
 \end{array}$$

When the overall antecedent is false, the whole schema is trivially true. When the overall antecedent (the first four lines of the predicate above) is true, then $from' = C$, and therefore the antecedent of the implication in the overall consequent is false which makes the whole schema true. More simply, this just means that the invariant is satisfied because it places no constraint on the behaviour of dishonest agent Colin.

For the next test we firstly calculate $ASendC \ ; \ CSendB \ ; \ BReply^*$.

$$\begin{array}{c}
 \frac{ASendC \ ; \ CSendB \ ; \ BReply^*}{\Delta InTransit} \\
 \hline
 to = \perp \wedge from' = B \\
 (\exists InTransit'' \bullet to'' = B \wedge from'' = C \wedge \\
 msg'' = \langle enc(pub(B), \langle non(A), adr(A) \rangle) \rangle) \wedge \\
 (\exists X : AGENT; N : NON \bullet to' = X \wedge \\
 msg'' = \langle enc(pub(B), \langle N, adr(X) \rangle) \rangle \wedge \\
 msg' = \langle enc(pub(X), \langle N, non(B), adr(B) \rangle) \rangle)
 \end{array}$$

Using the one-point rule we can simplify the schema because we know that X must be A and that N must be $\text{non}(A)$. Therefore, we can remove the remaining quantifier and doubly primed variables.

$$\frac{\frac{ASendC \ ; \ CSendB \ ; \ BReply^*}{\Delta InTransit}}{to = \perp \wedge to' = A \wedge from' = B \wedge msg' = \langle enc(pub(A), \langle non(A), non(B), adr(B) \rangle) \rangle}$$

Using this schema we can prove that this sequence of operations also maintains the invariant.

$$\frac{\frac{Inv \wedge ASendC \ ; \ CSendB \ ; \ BReply^* \Rightarrow Inv'}{\Delta InTransit}}{\begin{aligned} & ((to \neq \perp \wedge from \in \{A, B\}) \Rightarrow \\ & \quad \{n : NON \mid n \text{ in } dec(prv(to), msg)\} \subseteq \{non(from), non(to)\}) \wedge \\ & to = \perp \wedge to' = A \wedge from' = B \wedge \\ & msg' = \langle enc(pub(A), \langle non(A), non(B), adr(B) \rangle) \rangle \\ & \Rightarrow \\ & ((to' \neq \perp \wedge from' \in \{A, B\}) \Rightarrow \\ & \quad \{n : NON \mid n \text{ in } dec(prv(to'), msg')\} \subseteq \{non(from'), non(to')\}) \end{aligned}}$$

If $to \neq \perp$, then the overall antecedent is false and the whole schema is trivially true. Otherwise, if $to = \perp$, the first conjunct is trivially true and the remaining predicates can be simplified by application of the one-point rule and the decrypt function.

$$\frac{\frac{Inv \wedge ASendC \ ; \ CSendB \ ; \ BReply^* \Rightarrow Inv'}{\Delta InTransit}}{\begin{aligned} & (to = \perp \wedge to' = A \wedge from' = B \wedge \\ & msg' = \langle enc(pub(A), \langle non(A), non(B), adr(B) \rangle) \rangle) \\ & \Rightarrow \\ & \{non(A), non(B)\} \subseteq \{non(A), non(B)\} \end{aligned}}$$

As the consequent is true, we know that the schema is true. This is expected because honest agent Ben obeys the invariant.

It is at this point that $AAckC$ was enabled in the flawed protocol. We now show that the precondition of the new operation $AAckC^*$ is false at this point and hence not enabled. The precondition of an operation is calculated by assuming the existence of a final state in the operation [25].

$$\frac{\text{pre } AAckC^* \quad \text{InTransit}}{\exists \text{ InTransit}' \bullet}
\frac{}{to = A \wedge to' = C \wedge from' = A \wedge}
\frac{}{(\exists N : NON \bullet msg = \langle enc(pub(A), \langle non(A), N, adr(C) \rangle) \rangle) \wedge}
\frac{}{msg' = \langle enc(pub(C), \langle N \rangle) \rangle}$$

We know that such a post-state exists, so we can simplify the schema by removing the quantified post-state variables.

$$\frac{\text{pre } AAckC^* \quad \text{InTransit}}{to = A \wedge (\exists N : NON \bullet msg = \langle enc(pub(A), \langle non(A), N, adr(C) \rangle) \rangle)}$$

To prove that the precondition of $AAckC^*$ is not enabled, we prove that the sequence of operations leading up to $AAckC^*$ in the fixed protocol imply the negation of the precondition of $AAckC^*$. Z's 'pre' operator returns the implicit precondition of an operation schema by existentially quantifying the post-state variables.

$$\frac{ASendC \circ CSendB \circ BReply^* \Rightarrow \neg(\text{pre } AAckC^*)' \quad \Delta \text{InTransit}}{(to = \perp \wedge to' = A \wedge from' = B \wedge}
\frac{}{msg' = \langle enc(pub(A), \langle non(A), non(B), adr(B) \rangle) \rangle)}
\frac{}{\Rightarrow}
\frac{}{(to' \neq A \vee (\nexists N : NON \bullet msg' = \langle enc(pub(A), \langle non(A), N, adr(C) \rangle) \rangle))}$$

If the antecedent is false, the whole schema is trivially true. If the antecedent is true, then there does not exist a message of the form specified in the consequent, which requires address C to be in the message, and hence the whole schema is true. Therefore, operation $AAckC^*$ is not applicable at this point in the fixed protocol. Again, this formal proof matches our intuition. Operation $AAckC^*$ expects Colin's address to be in the message but the sequence of operations leading up to this point put Ben's address in the message instead.

Note that if $AAckC^*$ were performed, the invariant would be violated. We can therefore conclude that our invariant Inv is a desired property of the Needham-Schroeder Protocol, capable of formally distinguishing between successful and unsuccessful attacks.

9 Conclusion

Analysing security protocols is awkward because the security properties of interest are not directly expressible in typical formal methods. Although some researchers have attempted to remedy this by devising new, unfamiliar formalisms,

we instead prefer an approach which reuses widely-used methods. In this paper we demonstrated, via a worked example, that the Z notation is suitable, without change, for modelling and analysing a security protocol. This was done by systematically translating the protocol's informal 'standard notation' description into a Z model that accurately captured all essential features of the protocol. We then showed that Z's schema calculus provides a sound basis for formally reasoning about the protocol's correctness.

Acknowledgments

We wish to thank Luke Wildman for reviewing a draft of this paper and the anonymous reviewers for their helpful comments. This work was funded in part by Australian Research Council Discovery Grant DP0208046.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.
2. G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET registration protocols. *IEEE Journal On Selected Areas In Communications*, 21(5):77–87, January 2003.
3. C. Boyd. Security architectures using formal methods. *IEEE Journal On Selected Areas In Communications*, 11(5):694–701, June 1993.
4. C. Boyd and P. Kearney. Exploring fair exchange protocols using specification animation. In *Proceedings of the Information Security Workshop (ISW 2000)*, volume 1975 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 2000.
5. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report TR 39, Digital Equipment Corporation, February 1989.
6. M. Butler. On the use of data refinement in the development of secure communications systems. *Formal Aspects of Computing*, 14(1):2–34, 2002.
7. U. Carlsen. Generating formal cryptographic protocol specifications. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 137–146. IEEE Computer Society Press, 1994.
8. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0, 1997. <http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps>. Accessed May 2003.
9. E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 144–158. IEEE Computer Society Press, 2000.
10. G. Denker, J. K. Millen, A. Grau, and J. K. Filipe. Optimizing protocol rewrite rules of CIL specifications. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 52–63. IEEE Computer Society Press, 2000.
11. W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In *Proceedings of AFIPS 1976 National Computer Conference*, pages 109–112, Montvale, New Jersey, 1976.
12. L. Gong. Variations on the themes of message freshness and replay — or the difficulty of devising formal methods to analyze cryptographic protocols. In *Proceedings of the Computer Security Foundations Workshop VI*, pages 131–136. IEEE Computer Society Press, 1993.

13. R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal On Selected Areas In Communications*, 7(4):448–457, May 1989.
14. G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.
15. G. Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 31–43. IEEE Computer Society Press, 1997.
16. L. Ma and J. J. P. Tsai. *Formal Verification Techniques for Computer Communication Security Protocols*, volume 2. World Scientific Publishing Company, 2001. <http://www.cs.uic.edu/~lma/abstract1.pdf>. Accessed May 2003.
17. C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, February 1996.
18. C. A. Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology — ASIACRYPT '94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–149. Springer-Verlag, 1995.
19. C. Morgan. *Programming from Specifications*. Prentice Hall International Series In Computer Science. Prentice Hall International, 1989.
20. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
21. L. C. Paulson. Proving properties of security protocols by induction. In *Proceedings of 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 70–83. IEEE Computer Society Press, 1997.
22. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: The CSP Approach*. Addison-Wesley, 2000.
23. S. Schneider. Security properties and CSP. In *Proceedings of the 1996 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1996.
24. V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proceedings of 11th IEEE Computer Security Foundations Workshop (CSFW'98)*, pages 106–115. IEEE Computer Society Press, 1998.
25. J. M. Spivey. *The Z Notation : A Reference Manual*. Prentice Hall International Series In Computer Science. Prentice Hall, London, 1992.
26. S. G. Stubblebine and R. N. Wright. An authentication logic with formal semantics supporting synchronization, revocation, and recency. *IEEE Transactions on Software Engineering*, 28(3):256–285, March 2002.
27. F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society Press, May 1998.