SOFTWARE VERIFICATION RESEARCH CENTRE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 02-02

A Z Based Approach to Verifying Security Protocols

Benjamin W. Long
Colin J. Fidge
Antonio Cerone

August 2002

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533
http://www.svrc.uq.edu.au/
A Z Based Approach to Verifying Security Protocols

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

Software Verification Research Centre
The University of Queensland, Brisbane, Australia
e-mail: {benl,cj,antonio}@svrc.uq.edu.au

Abstract  We investigate the applicability and effectiveness of the Z method for specification and verification of public key infrastructure security protocols and show that careful consideration of data structures and models leads to an effective and elegant approach.

Key words  Specification and verification – Z – Security protocols

1 Introduction
Over the last decade, security has become an increasingly important issue in the networking world. Secure messaging is crucial in defence and e-commerce systems where the result of sensitive information getting into the hands of a malicious agent can be catastrophic [23]. Security protocols have been introduced to provide appropriate guidelines and techniques to ensure secure communications. We have seen the introduction of many security protocols and have witnessed many failures [2–4]. ‘Formal methods’ have often been promoted as a way of improving the correctness of security protocols, but the results to date have been disappointingly complex [12].

Formal methods are usually used to describe the functionality of a system and the restrictions that are placed upon its behaviour. However, to model security protocols we require the ability to express ill-defined concepts such as ‘confidentiality’ and ‘authentication’, which are not easily specified through existing formal methods. Previous research has emphasised the extension of existing methods, and the introduction of new methods designed specifically for the purpose of specification and verification of security protocols [12].

Research already done in verification of security protocols is dominated by event-based methods such as the spi calculus [1], and CSP [21], and specialised logics such as the BAN logic [7,25]. However, state-based methods such as B and particularly Z have seldom been used [8,5]. This forms part of the motivation for our research.

We investigate the use of Z [26] for specifying and verifying public key infrastructure security protocols. In doing this we devise a suitable set of data structures that capture the essential functionality of security protocols such as encryption, decryption, and the use of complex messages. To demonstrate the capability of our generic Z structures, we model the benchmark Needham-Schroeder Public Key Authentication Protocol [19], specifying the exact behaviour of operations internal to each agent in the system and the interactions between them. A suitable invariant ensuring the security of the protocol is devised and verified through the use of Z’s schema calculus [20].

2 Related Work
Many formal methods have been proposed for the verification of security protocols [12,17]. This section provides an overview of some of the methods closely related to our research.

Boyd [5] presented a formal design for a variety of security services based on the concepts of ‘confidentiality’ and ‘authentication’ using Z. His design is not for the purpose of any protocol in particular but a more general model that may be useful before particular protocols are considered. The model concentrates on users and keys, describing optimal behaviour of secure channels and secure services such as those used for key distribution, rather than modelling secure message transmissions.

Boyd and Kearney [6] explored protocol animation using Z for fair exchange protocols. Such protocols are used when two agents wish to exchange information, guaranteeing that each agent receives the other’s information. Boyd and Kearney were not concerned with the possibility of external attackers. The state of the system is modelled by the number of messages sent, what the agents have received, and the state of each agent, i.e.,
whether they are active or finished. The actual structure of the messages used in communication is not defined, making it difficult to apply this approach to confidentiality and authentication in the presence of intruders.

Butler [8] uses 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 of a system. The changing state of the system is modelled through the use of a history variable that contains a sequence of all messages transferred between agents. The model explicitly encodes each message used in the specification. Moreover, encryption is represented informally, merely by including keys as part of the messages.

Lowe [16] uses the Failures Divergences Refinement Checker (FDR), a model checker for CSP, to analyse the Needham-Schroeder Protocol. The agents of the system are modelled as CSP processes. CSP has little data structure support, so the message structure is captured in the event name. For example, \texttt{Msg1.Encrypt.k.n_a.a'} represents a message consisting of A's nonce \(n_a\), and A's identity \(a'\) encrypted with a key \(Encrypt.k\). These compound names make it difficult to express the complex recursive structures often needed for the specification of security protocols. Again the encrypt function is not formally modelled.

Focardi et al. [13] analyse the Needham-Schroeder Protocol using non-interference. Their model is specified using the Value-Passing Security Process Algebra, an extension of the Security Process Algebra (SPA), which itself is an extension of Milner's CCS. Basing their reasoning on non-interference, they aim to prove the absence of information flow between protocol partners and potential attackers. The agent's actions are defined as SPA processes making the specifications complex and 'code-like'.

While successful for their specific context, all of the above methods appear to be inflexible, or encode security concepts awkwardly. 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 Background

This section reviews basic security concepts and Z constructs.

3.1 The Standard Notation

The semi-formal 'standard notation' [9] is a widely used notation for describing the behaviour of security protocols. Interactions between agents are modelled by listing the transmissions of messages between them. The following is a simple example of the standard notation, modelling the transmission of a message from agent A (Antonio) to agent B (Ben).

\[ A \rightarrow B : \{ M \}_K, A \]

The message consists of two segments: an encrypted segment \( \{ M \}_K \) which is the result of encrypting message \( M \) with key \( K \); and a segment containing Antonio's address \( A \). Although this notation is a concise method of representing the structure of messages and the pattern of communications, it does not allow the modelling of operations internal to the agents or attacks made by intruders [9]. Furthermore, no semantics or reasoning principles are provided for this notation. Standard notation descriptions can be ambiguous or unclear, reinforcing the need for a formal method with the ability to provide more complete specifications and rigorous proofs.

3.2 Security Protocols

Security protocols are used to enable 'secure' transmission of sensitive information between agents of a system. The two most important properties of a secure message are:

- any message parts that are meant to be confidential within a select group of agents can be read only by the desired agents, and
- there is evidence that the message has not been forged.

These properties are referred to as confidentiality and authentication respectively. The use of encryption to achieve such properties for setting up secure communication between two agents was first described by Feistel [19].

Confidentiality is preserved by encrypting any secrets that are transmitted between agents in a way such that only desired viewers are able to decrypt the message\footnote{We assume that keys and encryption techniques are not compromised.}. The two general forms of key based encryption algorithms are symmetric and public-key algorithms [22, p. 3].

Symmetric algorithms involve an encryption key and a decryption key that are easily derived from each other. In fact, in many systems they are usually the same [22, p. 3]. Agents wishing to communicate in confidence will agree on a key for the encryption and decryption of messages to be transferred. Often an encryption key is only shared between a pair of agents. However, a third (trusted) party such as an authentication server could possibly know the shared key also [19].

Public-key encryption [11] is where all agents have a pair of corresponding keys associated with them: a public key that is distributed to everyone, and a private key that is only known to themselves. If a public key is used to encrypt a secret message, only the corresponding private key can be used to decrypt the message. Therefore,
in order to send a secret message, the sender uses the recipient’s public key to encrypt the message. Because the recipient is the only agent with the corresponding private key (the only agent that can decrypt the message), confidentiality is maintained. Similarly, if a private key is used to encrypt a message, only the corresponding public key can be used to decrypt it. An agent would encrypt a message with his private key in order to prove his identity (part of authentication) to the recipient. If the recipient successfully decrypts the message using the corresponding public key, he knows the identity of the agent that encrypted the message because assumably only one agent knows the private key.

Authentication is achieved by ensuring that intruders are not able to forge messages [21]. An intruder can forge a message by copying the known format of messages used in protocols (possibly by tampering with a message in transit), or simply by replaying messages from a previous instance of a protocol.

### 3.3 The Z Specification Language

Z [26] is a mathematical language based on set theory and logic. It can be used to specify the exact behaviour of systems without having to describe every detail such as implementable data structures and algorithms. Z models the state of the system being specified and the set of operations that are made upon the state of the system. Its mathematical base helps prevent ambiguity and allows us to conduct rigorous arguments to establish desired properties of the specification [20].

The main construct used in Z is the schema. In a schema we use mathematical predicates to define certain behaviours or properties of the system being modelled. A typical Z schema has a name Name, a list Decs of declarations, and a list Preds of predicate conjuncts describing behaviours and constraints on the items listed in the declarations:

```
Name

Decs

Preds
```

Unnamed schemas introduce global ‘axiomatic’ definitions.

Z has been used to design the functionality of many systems [14] such as the formal specification of a block structured symbol table, a telephone network, the UNIX filing system, and an assembler.

The advantage of using Z for the specification of security protocols is that it is a formal method which enables rigorous proof to be applied and it has powerful data structures that enable concise specifications to be written.

### 4 Data Structures

To model security protocols in Z, we have to define a suitable set of data types and structures that capture the appropriate aspects of cryptography common to most security protocols. To do this we first consider the basic operation of such protocols.

All protocols involve messages containing segments of data that are sent from one agent to another. Therefore we assume the set of all such segments as a given type.

\[ \text{[SEGMENT]} \]

Among these segments may be confidential data, keys, and addresses. These can be encrypted individually, or in combination with one or more other segments, producing a single encrypted segment. Many messages include ‘nonces’ (unique data values) for authentication purposes. To allow for these different types of segments, we declare the following five subsets of SEGMENT. Although two segments could have the same value (for example a key and an address could have the same bit pattern), by declaring the sets in the following way we are assuming that the role of each segment in the protocol is clear, either by its context or through an additional type ‘tag’. Then each segment is unique.

\[
\begin{align*}
\text{DAT} : & \mathbb{P} \text{SEGMENT} \\
\text{KEY} : & \mathbb{P} \text{SEGMENT} \\
\text{ADR} : & \mathbb{P} \text{SEGMENT} \\
\text{ENC} : & \mathbb{P} \text{SEGMENT} \\
\text{NON} : & \mathbb{P} \text{SEGMENT} \\
\end{align*}
\]

\[ \text{disjoint} (\text{DAT}, \text{KEY}, \text{ADR}, \text{ENC}, \text{NON}) \]

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. Note that we do not specify that each element in SEGMENT must be in one of the newly declared sets. This allows the definition of more types of segments as needed (see Section 4.2). Given these sets, a message is merely a sequence of segments.

\[ \text{MSG} == \text{seq SEGMENT} \]

Although this structure seems ‘flat’, we shall see in the next section how nested sequences of encrypted segments are handled.

#### 4.1 Encryption and Decryption

In order to cater for both symmetric and public-key algorithms, we will introduce subsets of private, public, and symmetric keys from the set of keys in the system. Again we will assume that each segment’s purpose is identified accordingly, allowing the three sets of keys to be disjoint.
The set $PRV$ is the set of private keys, $PUB$ is the set of public keys, and $SYM$ is the set of symmetric keys. The first two predicate conjuncts of the schema above state that the sets are pairwise disjoint and that every key may only be a private, or a public or a symmetric key. The total bijective function $pair$ is introduced to define a one-to-one symmetric correspondence between keys. Symmetry is ensured by the predicate $pair = pair^{-1}$ (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 either one private key and one public key or two symmetric keys.

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

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 segments) to a single encrypted segment, 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 (conject 1), then the result of the decryption is an empty message. If the first segment $s$ in the given message is not an encrypted segment (conject 2) or if the first segment was encrypted using a key that does not correspond to the given key (conject 3), then the segment is unchanged and the result of the decryption is message $(s)$ concatenated with the decryption of the remainder $m'$ of the message. Lastly, if the first segment of the given message is an encrypted segment that was created using the key that corresponds (via function $pair$) to the given key (conject 4), then the result is the decryption of the secret message $m''$ from the encrypted segment, 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 ($k \in PRV$);
- encryption using a private key ($pair(k) \in PRV$) and decryption using the corresponding public key ($k \in PUB$);
- encryption using a symmetric key ($pair(k) \in SYM$) and decryption using the corresponding symmetric key ($k \in SYM$), where the corresponding keys are possibly the same ($k = pair(k)$).

Therefore, we have defined data structures sufficient to support both symmetric and public-key cryptography.

### 4.2 Extending the Framework

The above definitions enable the modelling of protocols using symmetric and public-key cryptography. However, if they are lacking some additional features required for a particular protocol, extending the definitions is a simple task. To demonstrate the flexibility of our approach, we consider the addition of a hash function.

Hash functions are used to create digests of messages in the process of creating digital signatures [10, Ch. 7]. The set of hashed segments is defined from the set of all segments. Given a message, there is only one corresponding hashed segment. The function is not injective because, although we would like a unique value for every message that is hashed, this is not feasible.

$$HSH : \mathbb{P}SEGMENT$$

$$hash : MSG \rightarrow HSH$$

$$\text{disjoint} \{DAT, KEY, ADR, ENC, NON, HSH\}$$

In this way we can extend the communications data structure to introduce new concepts as desired.
5 The Needham-Schroeder Protocol

Needham and Schroeder [19] explain how public-key cryptography can be used to distribute keys to agents while at the same time authenticating each agent to one another. The original protocol, commonly referred to as the Needham-Schroeder Public Key Authentication Protocol, involves interaction with a third party to obtain keys for secure communication. This protocol has become a classic example in the literature. In standard notation, the three steps that allow agents Antonio (A) and Ben (B) to authenticate the origin of messages they have received are:

1. \[ A \rightarrow B : \{ N_A, A \}_K_B \]
2. \[ B \rightarrow A : \{ N_A, NB \}_K_A \]
3. \[ A \rightarrow B : \{ N_B \}_K_B \]

Firstly Antonio sends a message (step 1) to Ben consisting of a nonce \( N_A \) (a unique datum used for only one protocol run) 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 \( N_A \)). 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 property 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.

5.1 Modelling the Protocol

To model this protocol in Z, there are three agents that need to be introduced into the system, Antonio (A), Ben (B), and also Colin (C), an intruder who will be used to show a known attack on the protocol. We define \( AGENT \) to be the set of all agents in the network including a special symbol ‘⊥’ to represent ‘no agent’.

\[ AGENT ::= A \mid B \mid C \mid ⊥ \]

The state schema \( System \) below includes three total functions that map each agent\(^2\) to a unique public key, private key and nonce. Each agent has a nonce that is used for authenticating the agent with whom he is communicating. This is represented by an injective function ‘non’ that maps an agent to a nonce. This nonce is usually generated dynamically by the agent. However, for simplification of the model, we have statically associated nonces with agents since we are modelling only one protocol instance at a time. (A more advanced model could include the set of nonces already used by each agent.)

The predicate in the schema below ensures that the pair of keys associated with each agent \( G \) are a matching pair in the \( pair \) function.

\[
\begin{align*}
\text{System} & \quad \exists \ G : AGENT \rightarrow PUB \\
& \quad \exists \ prv : AGENT \rightarrow PRV \\
& \quad \exists \ non : AGENT \rightarrow NON \\
& \quad \forall \ G : AGENT, pub(G) = pair(prv(G))
\end{align*}
\]

The \( InTransit \) state schema contains the content of the communications medium. It is often assumed in security analysis that protocol instances are independent, so modelling one message at a time is adequate. Likewise, we also assume that one message only is in transit at a time. The \( msg \) variable represents the content of the message, if any, and the \( to \) and \( from \) variables represent whom the message is to and whom the message is from, respectively. We use the ‘no agent’ value \( ⊥ \) as the \( to \) address to indicate that no message is in transit.

\[
\begin{align*}
\text{InTransit} & \quad \exists \ msg : MSG \\
& \quad \exists \ to : AGENT \\
& \quad \exists \ from : AGENT
\end{align*}
\]

As the functions in \( System \) are static for the purpose of this paper, the \( InTransit \) state is sometimes modified while the \( System \) state is not. By declaring two state schemas, \( System \) and \( InTransit \), specifications are more concise when changing one part of the state but not the other. The complete state of the Needham-Schroeder system \( NS \) can be modelled by the conjunction of the two state schemas:

\[ NS \equiv System \land InTransit \]

To initialise the protocol, we merely need to state that no message is in transit, by setting the destination address to `⊥`.

\(^2\) To ensure that all predicates are well-defined, ‘no agent’ \( ⊥ \) has dummy keys and a nonce, but these are never used.
5.2 The 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 partitioning would make it difficult to interpose an intrusion between a message’s transmission and its subsequent receipt. Alternatively, the protocol could be naively broken into six operations — three sending operations and three receiving operations. However, assuming an intruder can not 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. 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.

\[
\begin{align*}
\text{InitNS} & \quad \text{NS} \\
\text{to} & = \bot
\end{align*}
\]

5.3 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 \(\text{non}(A)\) and his identity \(A\), and is encrypted using Ben’s public key \(\text{pub}(B)\)\(^3\). This operation is modelled in Z as follows:

\[^{\text{3}}\text{We assume that all public keys are known to all agents.}\]

\[
\begin{align*}
\text{ASendB} & \quad \text{A} \rightarrow \text{B} \\
\text{BReply} & \quad \text{B} \rightarrow \text{A} \\
\text{AAckB} & \quad \text{A} \rightarrow \text{B} \\
\text{BCheck} &
\end{align*}
\]

In the declaration, Z’s `Δ’ annotation is used to state that the variables in schema \(\text{InTransit}\) may change value, whereas the `\(\Xi\)’ annotation says that those in schema \(\text{System}\) are not changed 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 implicitly checked in the precondition by the absence of a destination. The post-state variables \(\text{to}’\) and \(\text{from}’\) are set to indicate that after the operation the message in transit is to Ben and from Antonio. The post-state value \(\text{msg}’\) of the message contains an encrypted segment made from the appropriate structure of the message sent in step 1 of the protocol. The encrypted segment is inside Z’s sequence brackets \(\langle \cdots \rangle\) because a message always consists of a sequence of segments, even though it contains only one segment in this case. After this operation, Antonio is waiting for a reply message.

5.4 Ben Replies to the Message

The operation BReply is a generalised operation where Ben receives a message \(\text{msg}’\) from an unknown agent X and replies to the agent whose identity is present in the received message. The reply message \(\text{msg}’\) contains the nonce received and Ben’s nonce.

\[
\begin{align*}
\text{BReply} & \quad \text{A} \rightarrow \text{B} \\
\text{ΔInTransit} & \quad \text{to} = \bot \land \text{to}’ = \text{B} \land \text{from}’ = \text{A} \\
\text{msg} & = \langle \text{enc}(\text{pub}(B), \{\text{non}(A), A\}) \rangle \\
\text{msg}’ & = \langle \text{enc}(\text{pub}(X), \{\text{non}(A), A\}) \rangle
\end{align*}
\]

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 identity \(X\). 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 \(\text{non}(B)\), is encrypted using X’s public key \(\text{pub}(X)\).
5.5 Antonio Acknowledges the Message

In the following operation Antonio authenticates Ben and sends the newly received nonce back to Ben in order to be authenticated by him.

\[
\begin{align*}
A & \text{Ack} \ B \\
\Delta \text{InTransit} \\
\Xi \text{System} \\
to = A \land to' = B \land from' = A \\
(\exists N : NON) \\
\text{msg} &= \langle \text{enc}(\text{pub}(A), \langle \text{non}(A), N \rangle) \rangle \\
\text{msg'} &= \langle \text{enc}(\text{pub}(B), \langle N \rangle) \rangle
\end{align*}
\]

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, \(\text{pub}(B)\). 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 6.

5.6 Ben Checks the Message

Ben is now expecting a message of a particular form. Again this is implicitly checked as part of the precondition of the following operation which ensures that Ben’s nonce is part of the received message, therefore allowing Ben to authenticate the identity of the agent he believes he is communicating with.

\[
\begin{align*}
B & \text{Check} \\
\Delta \text{InTransit} \\
\Xi \text{System} \\
to = B \land to' = \bot \\
\text{msg} &= \langle \text{enc}(\text{pub}(B), \langle \text{non}(B) \rangle) \rangle
\end{align*}
\]

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.

6 An Attack on the Protocol

Lowe [15] identified an intrusion on this protocol whereby Antonio honestly communicates with Colin \(C\) (the intruder) not knowing that he is malicious. Colin is able to masquerade as Antonio by sending modified messages to Ben.

1. \(A \rightarrow C : \{N_A, A\}_{K_C}\)
2. \(C_A \rightarrow B : \{N_A, A\}_{K_A}\)
3. \(B \rightarrow C_A : \{N_A, N_B\}_{K_A}\)
4. \(C \rightarrow A : \{N_A, N_B\}_{K_A}\)
5. \(A \rightarrow C : \{N_B\}_{K_C}\)
6. \(C_A \rightarrow B : \{N_B\}_{K_A}\)

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 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.

6.1 The Operations

We explained above that Colin intercepts the message in step 3 but realises that he 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 as described in Section 5.2, leaving us with six \(Z\) operation schemas for modelling the intrusion (see Fig. 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 \(A_{SendC}\) which is similar to the \(A_{SendB}\) operation, but where Antonio sends the initial message to Colin instead of Ben.

\[
\begin{align*}
A & \text{SendC} \\
\Delta \text{InTransit} \\
\Xi \text{System} \\
to = \bot \land to' = C \land from' = A \\
\text{msg'} &= \langle \text{enc}(\text{pub}(C), \langle \text{non}(A), A \rangle) \rangle
\end{align*}
\]

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. The operation is applicable whenever there is a message for Colin in transit.
longer holds, because at the end of the protocol, the secret nonce is no longer secret. Hence, the protocol is not secure.

Lowe’s model [16] introduces a property stating that an agent B can commit to a session with an agent A only if A was originally communicating with B. In the flawed Needham-Schroeder Protocol, B commits to a session with A even though A was not originally communicating with B. As this property is violated, Lowe concludes that the protocol is not secure.

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 safely. 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 wishes for 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 safety properties, but we want to ensure that each honest agent always aims to maintain such a safety 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 secret nonces such as in Butler’s model is required. Our property can be modelled as an invariant for any such protocol. For the Needham-Schroeder Protocol, it can be expressed by the following invariant.

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 the left is in the sequence on the right [24].

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 will use Z’s schema calculus [20]. We will construct a schema from the composition of ASendC, CSendB, BReply, and AAckC, and prove that this sequence of operations contradicts the invariant. Given

\[
\text{CSendB} \quad \Delta \text{lnTransit} \\
\Xi \text{System} \\
\text{to} = C \land \text{to}' = B \land \text{from}' = C \land \exists M : MSG \bullet \text{msg} = \langle \text{enc}(\text{pub}(C), M) \rangle \land \text{msg}' = \langle \text{enc}(\text{pub}(B), M) \rangle
\]

\[
\text{AAckC} \quad \Delta \text{lnTransit} \\
\Xi \text{System} \\
\text{to} = A \land \text{to}' = C \land \text{from}' = A \land \exists N : NON \bullet \text{msg} = \langle \text{enc}(\text{pub}(A), \langle \text{non}(A), N \rangle) \rangle \land \text{msg}' = \langle \text{enc}(\text{pub}(C), \langle N \rangle) \rangle
\]

\[
\text{Inv} \\
\text{lnTransit} \\
\text{(to} \neq \bot \land \text{from} \in \{A, B\} \Rightarrow \{n \in \text{NON} | n \in \text{dec}(%20\text{prov(to)}, \text{msg})\} \subseteq \{\text{non(from)}, \text{non(to)}\}\]

7 Verification of the Protocol

The Needham-Schroeder Protocol has been analysed previously using different approaches [8,16] which not only are based on different modeling frameworks, but also utilise different characterisations of the security of the protocol.

Butler’s model [8] incorporates a variable containing nonces that are secret. He specifies a property (invariant) stating that ‘once a nonce is secret, it remains secret.’ With a run of the original flawed Needham-Schroeder Protocol in the presence of an intruder, this clause no
two schemas \( S \) and \( T \), the composition ‘\( S \uplus T \)’ 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 \( T \) [20, p.147].

After initialisation, only \( ASendC \) is enabled because \( to = \bot \). 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.

\[
\begin{align*}
& \text{ASendC} \uplus \text{CSendB} \\
\Delta \text{In Transit} & \\
\Xi \text{System} & \\
\end{align*}
\]

To simplify this complicated schema, the nested quantifier can be removed by application of the one-point law [18] because we know that \( M \) must be \( \langle \text{non}(A), A \rangle \).

\[
\begin{align*}
& \text{ASendC} \uplus \text{CSendB} \\
\Delta \text{In Transit} & \\
\Xi \text{System} & \\
\end{align*}
\]

Next we will compose this schema with \( BReply \).

\[
\begin{align*}
& \text{ASendC} \uplus \text{CSendB} \uplus \text{BReply} \\
\Delta \text{In Transit} & \\
\Xi \text{System} & \\
\end{align*}
\]

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 \( \text{non}(A) \).

\[
\begin{align*}
& \text{ASendC} \uplus \text{CSendB} \uplus \text{BReply} \\
\Delta \text{In Transit} & \\
\Xi \text{System} & \\
\end{align*}
\]

Finally we compose this schema with \( AAckC \).

\[
\begin{align*}
& \text{ASendC} \uplus \text{CSendB} \uplus \text{BReply} \uplus \text{AAckC} \\
\Delta \text{In Transit} & \\
\Xi \text{System} & \\
\end{align*}
\]

Once again we can simplify the schema because we know that nonce \( N \) is \( \text{non}(B) \).
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:

\[
\begin{align*}
\Delta \text{InTransit} \\
\equiv \text{System}
\end{align*}
\]
\[
to = \perp \land \text{to}' = C \land \text{from}' = A \\
\text{msg}' = \langle \text{enc} (\text{pub}(C), \text{non}(B)) \rangle
\]

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

\[
\begin{align*}
\Delta \text{InTransit} \\
\equiv \text{System}
\end{align*}
\]
\[
( (\text{to} \neq \perp \land \text{from} \in \{A, B\}) \Rightarrow \\
\{ n : \text{NON} \mid n \in \text{dec} (\text{prev}(\text{to}), \text{msg}) \} \\
\subseteq \{\text{non}(\text{from}), \text{non}(\text{to})\} \land \\
\text{to} = \perp \land \text{to}' = C \land \text{from}' = A \land \\
\text{msg}' = \langle \text{enc} (\text{pub}(C), \text{non}(B)) \rangle \rangle \\
\Rightarrow \\
( (\text{to}' \neq \perp \land \text{from}' \in \{A, B\}) \land \\
\{ n : \text{NON} \mid n \in \text{dec} (\text{prev}(\text{to'}), \text{msg}') \} \\
\subseteq \{\text{non}(\text{from'}), \text{non}(\text{to'})\}) \]

We distinguish two cases.

- If \(\text{to} \neq \perp\), then the overall antecedant (the first five lines of the predicate above) is false and the whole schema is trivially true.

- If \(\text{to} = \perp\), then the overall antecedant is simplified as follows:

\[
\begin{align*}
\text{to} = \perp \land \text{to}' = C \land \text{from}' = A \land \\
\text{msg}' = \langle \text{enc} (\text{pub}(C), \text{non}(B)) \rangle
\]

We distinguish two subcases.

- If predicate 1 is false, then the whole schema is trivially true.

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

\[
\{ n : \text{NON} \mid n \in \text{dec} (\text{prev}(\text{to'}), \text{msg}') \} \\
\subseteq \{\text{non}(\text{from'}), \text{non}(\text{to'})\} \]

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

Predicate 2 is then further simplified as follows:

\[
\{\text{non}(B)\} \subseteq \{\text{non}(A), \text{non}(C)\}
\]

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

This proves that the invariant is not maintained and the protocol is not secure.

8 The Fixed Needham-Schroeder Protocol

It is suggested by Lowe [15] 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 \(\{N_A, N_B, B\}_{K_s}\). 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.

The operations that change to incorporate this new message structure are \(B\text{Reply} \) and \(A\text{AckC}\). Both play an important part in fixing the original protocol. The only difference in the new operation \(B\text{Reply}^*\) is that B adds his identity to msg’ to conform to the new protocol.

\[
\Delta \text{InTransit} \\
\equiv \text{System}
\]
\[
\text{to} = B \land \text{from}' = B \\
\langle \exists X : \text{AGENT}; \text{N : NON} \bullet \\
\text{to}' = X \land \\
\text{msg} = \langle \text{enc}(\text{pub}(B), \langle N, X \rangle) \rangle \land \\
\text{msg}' = \langle \text{enc}(\text{pub}(X), \langle N, \text{non}(B), B \rangle) \rangle
\]

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 precondition for \(A\text{AckC}^*\) is that Colin’s identity is in the message.
We now show that each sequence of operations before $A\text{Ack}C^*$ maintains the invariant and that the precondition of $A\text{Ack}C^*$ is violated after $B\text{Reply}^*$, therefore indicating that $A\text{Ack}C^*$ must not be performed. Firstly we show that $A\text{Send}C$ maintains the invariant.

$$\text{Inv} \wedge A\text{Send}C \Rightarrow \text{Inv}'$$

$$\text{ΔInTransit}$$

$$\text{System}$$

$$((\exists to \neq \perp \land \text{from} \in \{A,B\}) \Rightarrow \{ n : \text{NON} \mid n \in \text{dec}(\text{prv}(to), \text{msg}) \} \subseteq \{ \text{non}(\text{from}), \text{non}(to) \}) \land$$

$$\text{to} = \perp \land \text{to}' = C \land \text{from}' = A \land$$

$$\text{msg}' = \langle \text{enc}(\text{pub}(C), \langle \text{non}(A), A \rangle) \rangle$$

$$\Rightarrow$$

$$((\perp \neq \perp \land \text{from} \in \{A,B\}) \Rightarrow \{ n : \text{NON} \mid n \in \text{dec}(\text{prv}(to'), \text{msg}') \} \subseteq \{ \text{non}(\text{from}'), \text{non}(\text{to}') \})$$

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

$$\text{Inv} \wedge A\text{Send}C \Rightarrow \text{Inv}'$$

$$\text{ΔInTransit}$$

$$\text{System}$$

$$(\exists \text{to} = \perp \land \text{to}' = C \land \text{from}' = A \land$$

$$\text{msg}' = \langle \text{enc}(\text{pub}(C), \langle \text{non}(A), A \rangle) \rangle)$$

$$\Rightarrow$$

$$\{ \text{non}(A) \} \subseteq \{ \text{non}(A), \text{non}(C) \}$$

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 $A\text{Send}B \Rightarrow C\text{Send}B$ calculated in Section 7, we prove that the sequence of these two operations maintains the invariant.

$$\text{Inv} \wedge A\text{Send}C \Rightarrow C\text{Send}B \Rightarrow \text{Inv}'$$

$$\text{ΔInTransit}$$

$$\text{System}$$

$$(\exists (\text{to} = \perp \land \text{to}' = A \land \text{from}' = B \land$$

$$\text{msg}' = \langle \text{enc}(\text{pub}(A), \langle \text{non}(A), \text{non}(B), B \rangle) \rangle)$$

$$\Rightarrow$$

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.
\[
\begin{array}{l}
\text{Inv } \land A\text{SendC } \land C\text{SendB } \land B\text{Reply}^* \Rightarrow \text{Inv'} \\
\Delta \text{In Transit} \\
\Xi \text{System} \\
\end{array}
\]

\[
((t \neq \bot \land \text{from} \in \{A, B\}) \Rightarrow \\
\{ n : \text{NON} \mid n \in \text{dec(\text{prov}(t), msg)} \} \\
\subseteq \{ \text{non(from')}, \text{non(to')} \}) \land \\
to = \bot \land \text{to'} = A \land \text{from'} = B \land \\
msg' = \langle \text{enc(\text{pub}(A), \langle \text{\text{non}(A), \text{non}(B), B} \rangle) \rangle \rangle \\
\Rightarrow \\
((t' \neq \bot \land \text{from'} \in \{A, B\}) \Rightarrow \\
\{ n : \text{NON} \mid n \in \text{dec(\text{prov}(t'), msg')} \} \subseteq \\
\{ \text{non(from'), non(to')}) \}
\]

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

\[
\begin{array}{l}
\text{Inv } \land A\text{SendC } \land C\text{SendB } \land B\text{Reply}^* \Rightarrow \text{Inv'} \\
\Delta \text{In Transit} \\
\Xi \text{System} \\
\end{array}
\]

\[
((o = \bot \land \text{to'} = A \land \text{from'} = B \land \\
msg' = \langle \text{enc(\text{pub}(A), \langle \text{\text{non}(A), \text{non}(B), B} \rangle) \rangle \rangle \\
\Rightarrow \\
\{ \text{non(A), non(B)} \} \subseteq \{ \text{non(A), non(B)} \}
\]

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 \(A\text{AckC}\) was enabled in the broken protocol. We now show that the precondition of the new operation \(A\text{AckC}^*\) is true 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 [20, p.151].

\[
\begin{array}{l}
\text{pre } A\text{AckC}^* \\
\Delta \text{In Transit} \\
\Xi \text{System} \\
\exists \text{In Transit'} \bullet \\
\text{to} = A \land \text{to'} = C \land \text{from'} = A \land \\
(\exists N : \text{NON } \bullet \\
\text{msg} = \langle \text{enc(\text{pub}(A), \langle \text{\text{non}(A), N, C} \rangle) \rangle \rangle \land \\
\text{msg'} = \langle \text{enc(\text{pub}(C), \langle N) \rangle) \rangle \\
\end{array}
\]

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

\[
\begin{array}{l}
\text{pre } A\text{AckC}^* \\
\Delta \text{In Transit} \\
\Xi \text{System} \\
\text{to} = A \\
(\exists N : \text{NON } \bullet \\
\text{msg} = \langle \text{enc(\text{pub}(A), \langle \text{\text{non}(A), N, C} \rangle) \rangle \rangle \\
\end{array}
\]

To prove that the precondition of \(A\text{AckC}^*\) is not enabled, we prove that the sequence of operations leading up to \(A\text{AckC}^*\) in the fixed protocol imply the negation of the precondition of \(A\text{AckC}^*\).

\[
\begin{array}{l}
A\text{SendC} \lor C\text{SendB} \land B\text{Reply}^* \Rightarrow \neg (\text{pre } A\text{AckC}^*'). \\
\Delta \text{In Transit} \\
\Xi \text{System} \\
\end{array}
\]

\[
((o = \bot \land \text{to'} = A \land \text{from'} = B \land \\
msg' = \langle \text{enc(\text{pub}(A), \langle \text{\text{non}(A), non}(B), B} \rangle) \rangle \\
\Rightarrow \\
(\text{to'} \neq A \lor \\
(\exists N : \text{NON } \bullet \\
\text{msg'} = \langle \text{enc(\text{pub}(A), \langle \text{\text{non}(A), N, C} \rangle) \rangle \rangle \\
\]

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 \(A\text{AckC}^*\) is not applicable at this point in the fixed protocol. Again, this formal proof matches our intuition. Operation \(A\text{AckC}^*\) 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 \(A\text{AckC}^*\) 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. Similar invariants will be useful for verifying desirable security properties of other security protocols.

9 Conclusion

We have successfully devised data structures and functionality appropriate for modelling security protocols using the Z specification language. This was done by identifying every segment of data in a message as an element of a primitive type and then defining a message as a sequence of these segments. As encrypted messages are also identified as segments we allowed for recursive data structures that are often used in public key protocols.

Using these structures, we modelled the Needham-Schroeder Public Key Authentication Protocol. The data structures allowed the operations to be very concise but still captured the desired behaviour, including operations internal to the agents such as checking that certain conditions are satisfied when receiving messages and generating responses based on the content of messages received. It was then possible to conduct formal proofs using simple predicate logic.

In modelling this protocol we used a new approach where protocol steps can be grouped so that the receipt and response to a message by the same agent are represented as one operation. This is sufficient because those
operations internal to an agent are never interfered with. However, a message sent by an agent and then received by another agent must be represented as two separate operations to allow for the modelling of intrusions on a message in transit. This simplifying concept should apply to any protocol where the receipt of a message is immediately followed by some response by the same agent.

We also devised an invariant based on a property that should be true of all public key protocols that use nonces, this being that each honest agent must be able to trust all other honest agents to use nonces safely. This is expressed by ensuring that every nonce in a message sent by an honest agent must belong to either the sender or the receiver. In our example Ben has to trust Antonio, however Colin tricks Antonio into breaching the trust, resulting in Ben’s nonce being revealed to Colin. Using the Z schema calculus we proved that the invariant was not maintained after Antonio sent Ben’s nonce to the intruder. Using the same invariant for the fixed protocol, we proved that the invariant is maintained up until the protocol stops, proving that the protocol is secure.

Acknowledgments

We wish to thank Luke Wildman for reviewing a draft of this paper. This work was funded in part by Australian Research Council Discovery Grant DP0208046.

References

25. S. G. Stubblebine and R. N. Wright. An authentication logic with formal semantics supporting synchronization,