Towards the rank function verification of protocols that use temporary secrets

Rob Delicata and Steve Schneider

Department of Computer Science, Royal Holloway University of London, Egham, Surrey, TW20 0EX, UK.
{R.G.Delicata, S.Schneider}@cs.rhul.ac.uk

Corrected conference version (20th April 2004)

Abstract. The rank functions approach to protocol analysis is incomplete with respect to confidentiality properties. In particular, protocols that use temporary secrets may be un-verifiable within the framework. This paper investigates the incompleteness and proposes a solution based on a novel theory of combined rank functions. We demonstrate that an otherwise un-verifiable protocol may have a rank functions proof if the protocol is divided into two sub-protocols. We define a method for dividing a protocol and prove a result that allows us to conclude the correctness of the entire protocol from the correctness of each of its sub-protocols.

1 Introduction

The rank functions approach to protocol analysis is incomplete with respect to confidentiality properties. In particular, secure confidentiality (secrecy) protocols that use temporary secrets may be un-verifiable using the approach. The question of completeness for authentication properties is still an open issue. In this paper we propose a solution to the temporary secrets incompleteness based on a novel theory of combined rank functions. Briefly, if a secrecy protocol has no proof using the standard rank functions approach, it may be mechanically divided into two sub-protocols such that the temporary secret is a secret value in the first sub-protocol and a public value in the second sub-protocol. Separate rank functions may be defined to prove the security of each sub-protocol, and we prove a result that guarantees the correctness of the full protocol if both sub-protocols are secure.

The scope of this paper is restricted to the verification of two-party protocols that use at most one temporary secret on a small (single-run) network. It is our belief that the results in this paper can be generalised to deal with protocols running on an unbounded network with an arbitrary number of temporary secrets.
2 CSP and rank functions

In this section we review the use of rank functions for protocol analysis. A knowledge of CSP and its trace semantics is assumed and the reader is urged to refer to [10] for a thorough treatment. A rank function verification begins with the construction of a CSP protocol model, where each agent and a Dolev-Yao intruder [2] are represented by CSP processes. The network is formed as a parallel composition of the user processes, with all communication passing through the intruder. A rank function is then defined to assign a rank of 0 or 1 to every message that may pass across the CSP network. The goal of a rank function is to divide the message-space into two by assigning a rank of 1 to messages that may be safely overheard by the intruder, and a rank of 0 to all messages that should definitely remain secret. In this model, secrecy corresponds to the intruder's inability to discover a message with rank 0.

2.1 CSP model for Heather’s NSLPKP variant

As a running example we use a protocol proposed by Heather [5] as a simple variant of the Needham-Schroedel-Lowe public-key protocol (NSLPKP) [7]:

1. $a \rightarrow b : \{a.na\}_{PKb}
2. b \rightarrow a : \{b.na.k\}_{PKa}
3. a \rightarrow b : \{m\}_k.na$

This protocol aims to guarantee the secrecy of message $m$ and key $k$. The initiator begins by sending the concatenation of a nonce, $na$, and her identity, $a$, to $b$ encrypted under his public key. On receipt, agent $b$ responds by returning $na$ along with his identity $b$ and a fresh session-key $k$ all encrypted under $a$’s public key. Now that $k$ is known to $a$ and $b$, $a$ can send a message $m$ to $b$, encrypted under $k$. Since $a$ also leaks the value of $na$ in the final message it is a temporary secret. The initiator and responder for this protocol can be represented by the following CSP processes:

$\text{Init}(a, b, na, m) =$
\begin{align*}
\text{trans} & .a.b.(a.na)_{PKb} \\
\rightarrow & \text{rec}.a.b.(b.na.k)_{PKa} \\
\rightarrow & \text{trans}.a.b.(m)_k.na \\
\rightarrow & \text{Stop}
\end{align*}

$\text{Resp}(b, k) =$
\begin{align*}
\text{rec} & .b?a.(a?na)_{PKb} \\
\rightarrow & \text{trans} .b.a.(b.na.k)_{PKa} \\
\rightarrow & \text{rec}.b.a.(m)_k.na \\
\rightarrow & \text{Stop}
\end{align*}

The event $\text{trans}.a.b.\text{msg}$ represents the transmission to $b$, apparently from $a$, of a message $\text{msg}$. Similarly, $\text{rec}.a.b?\text{msg}$ represents $a$’s receipt of a message $\text{msg}$, apparently from $b$. The ‘?’ signifies that $a$ will accept any value $\text{msg}$ of the correct type. The process $\text{ENEMY}$ (omitted) models a Dolev-Yao intruder. With these processes we can construct a small-network as follows:

$\text{Initiator(a)} = \square b \in U_a, na \in N_a, m \in T_a \text{ Init}(a, b, na, m)$
\[ \text{Responder}(b) = \bigwedge_{k \in K_b} \text{Resp}(b, k) \]

\[ \text{NET} = \bigwedge_{a, b \in U_h} \left( \text{Initiator}(a) \parallel \text{Responder}(b) \right) \rightharpoonup (\text{trans}, \text{rec}) \text{ ENEMY} \]

The \( \bigwedge \) symbol represents generalised external choice, sets \( N_h, T_h \) and \( K_h \) are respectively the nonces, plaintexts and keys available to any user \( u \) in the set of honest users \( U_h \). The user processes must synchronise with \( \text{ENEMY} \) on all events on the \( \text{trans} \) and \( \text{rec} \) channels, representing the fact that all communication passes through the intruder.

### 2.2 Secrecy properties

Rank functions [11,6] have traditionally been used to verify authentication properties of security protocols, namely, that the occurrence of an event \( t \) in a protocol implies that an event \( r \) has previously occurred. For a protocol modelled by a CSP process \( \text{NET} \) and sets of messages \( R \) and \( T \) we say that \( R \) precedes \( T \) on \( \text{NET} \) if no message \( t \in T \) can appear on \( \text{NET} \) until some message \( r \in R \) has appeared. This translates into a predicate on the traces (the set of sequences of possible events) of \( \text{NET} \):

**Definition 1.** For sets \( R, T \subseteq M \), we define \( R \) precedes \( T \) as:

\[ \text{NET sat} \ R \ \text{precedes} \ T \iff \forall tr \in \text{traces}(\text{NET}) \bullet (tr \upharpoonright R = \emptyset \Rightarrow tr \upharpoonright T = \emptyset) \]

The notation \( tr \upharpoonright R \) means a trace \( tr \) restricted to events in the set \( R \) and \( M \) is the set of all messages. It was noticed in [4] that the \( \text{precedes} \) specification allows us to model secrecy properties by setting \( R = \emptyset \). The predicate \( \text{NET sat} \ \emptyset \ \text{precedes} \ T \) states that no message \( t \in T \) can ever appear on \( \text{NET} \). We will use the predicate \( \text{secrecy}(T) \) as an abbreviation for \( \emptyset \ \text{precedes} \ T \).

### 2.3 Rank functions for secrecy

The following description tailors the main rank functions result to the verification of secrecy properties.

**Definition 2.** A rank function is a function \( \rho : M \rightarrow \{0, 1\} \) from the message-space to the binary valued set \( \{0, 1\} \).

A process \( P \) preserves the rank with respect to a rank function \( \rho \) if it never transmits a message \( m \) with \( \rho(m) = 0 \) unless it has previously received a message \( m' \) with \( \rho(m') = 0 \). Intuitively, a protocol agent should never give out a secret unless he has already been told a secret.

**Definition 3.** For a rank function \( \rho \), \( M_{\rho^-} = \{ m \in M \bullet \rho(m) = 0 \} \) is the subset of the message-space that contains all messages with rank \( 0 \) and \( M_{\rho^+} = \{ m \in M \bullet \rho(m) = 1 \} \) is the subset that contains all messages with rank \( 1 \).
We lift a rank function \( \rho \) to communication events in the obvious way. For example, \( \rho(\text{trans.a.b.m}) = \rho(m) \).

**Definition 4.** For the set of users, \( \mathcal{U} \), we say that a process \( P \) preserves \( \rho \) if:
\[ P \text{ sat secrecy}(T) \quad \text{precedes} \quad \text{trans}_{\mathcal{U} \cup \mathcal{M}_P} \]

**Theorem 1.** For protocol model \( \text{NET} \) and set \( T \), \( \text{NET sat secrecy}(T) \) if there is a rank function \( \rho : \mathcal{M} \rightarrow \{0, 1\} \) satisfying:
- \( C_1 : \forall m \in \mathcal{I} \mathcal{K} \cdot \rho(m) = 1 \)
- \( C_2 : \forall S \subseteq \mathcal{M}, m \in \mathcal{M} \cdot ((\forall m' \in S \cdot \rho(m') = 1) \land S \vdash m) \Rightarrow \rho(m) = 1 \)
- \( C_3 : \forall t \in T \cdot \rho(t) = 0 \)
- \( C_4 : \forall u \in \mathcal{U} \cdot \text{Initiator}(u) \sqcap \text{Responder}(u) \cdot \text{sat preserves} \rho \)

The proof is omitted but can be found in [11]. The theorem reduces a requirement on the overall system to separate requirements on the individual components of the system. The conditions correspond to the requirements that: (\( C_1 \)) the intruder’s initial knowledge (the set \( \mathcal{I} \mathcal{K} \)) must only contain messages with rank 1, (\( C_2 \)) the intruder can only generate messages with rank 1 if he only hears messages with rank 1, (\( C_3 \)) no event in \( T \) has rank 1 and (\( C_4 \)) a user will only send a message with rank 0 if he has already received a message of rank 0. Conditions \( C_1, C_2 \) and \( C_4 \) together ensure that only messages with rank 1 can ever be passed in the system. Condition \( C_3 \) allows the conclusion that no event from \( T \) can therefore occur. Rank functions have been used to analyse a variety of security protocols. See [11,3] for a representative selection.

### 3 Temporary secrets and incompleteness

In this section we define completeness in the context of rank functions protocol verification and describe Heather’s incompleteness result for secrecy specifications.

The rank functions approach is complete if every secure protocol has an associated rank function. However, as noted in [4] there are two independent factors to consider when deciding completeness; the protocol itself and the network on which it is run. In this paper we consider a relatively weak completeness question: does every secure combination of a small (single-run) network and a secrecy protocol have an associated rank function?

### 3.1 Incompleteness

The answer to the weak completeness question turns out to be no. In [5] Heather shows that his NSLPK variant is un-verifiable using the standard rank functions approach. To see why, we can attempt to construct a rank function for it. Consider the following run:

1. \( Alice \rightarrow Bob : \{Alice.Na\}_{PK(bob)} \)
2. \( Bob \rightarrow Alice : \{Bob.Na.K\}_{PK(Alice)} \)
3. \( Alice \rightarrow Bob : \{M\}_{K.Na} \)
All messages that are available to the intruder must have a rank of 1, and we typically assume that the intruder knows the identity of all other users. We must set \( \rho(\{Alice, Na\}_{PK(bob)}) = 1 \) since Alice can send this message without receiving any inputs. Similarly, we must set \( \rho(\{Bob, Na, K\}_{PK(Alice)}) = 1 \) as Bob can construct the message having only received a message of rank 1. We must also set \( \rho(\{M\}_K, Na) = 1 \), and, since the intruder can split concatenations, we are compelled to set \( \rho(\{M\}_K) = 1 \) and \( \rho(Na) = 1 \). However, if \( \rho(Na) = 1 \) then \( \rho(\{Bob, Na, K\}_K, PK(Alice)) = 1 \) for any \( K' \) that the intruder can generate. On receipt of the message:

2. \( intruder \rightarrow Alice : \{Bob, Na, K'\}_{PK(Alice)} \)

where the intruder masquerades as Bob, Alice will send:

3. \( Alice \rightarrow Bob : \{M\}_{K', Na} \)

and since \( \rho(K') = 1 \), the intruder can deduce \( M \), forcing us to set \( \rho(M) = 1 \).

Some thought leads us to conclude that this attack is impossible, since it relies on the intruder’s knowledge of \( Na \), and Alice does not leak \( Na \) until after the session-key has been agreed. As long as the nonce \( Na \) is fresh for each run, this attack is not possible. However, Heather has shown that no rank function exists for this protocol [5]. A rank function is a static assignment of ranks to messages, with no concept of causal precedence, and Heather’s protocol illustrates a situation where a message requires a dynamic rank. In this case \( \rho(Na) \) must be 0 until the second message is received, and can safely be 1 thereafter. A rank function cannot cater for this.\(^1\)

This incompleteness is caused by the presence of a temporary secret: the nonce \( Na \). A temporary secret is a value (not initially known to the intruder) that first appears on the network as ciphertext that the intruder cannot decrypt, and then appears later as plaintext. A formal definition of a temporary secret is not needed here, but may be found (in the context of Lowe’s no temporary secrets assumption) in [8].

4 Combining rank functions on a small network

The rank of a temporary secret must initially be 0 and become 1 when the secret is made public. It has been established that rank functions cannot achieve this. We propose a solution to this problem based on combined rank functions and divided protocols. Given a security protocol \( P \) with a temporary secret \( tmp \), we identify the point immediately preceding the leakage of \( tmp \) and call it the breakpoint. We can let \( \rho(tmp) = 0 \) whenever it occurs before the breakpoint and then let \( \rho(tmp) = 1 \) for all subsequent behaviour. To achieve this apparent dynamicism of a rank function we divide the protocol \( P \) into two along the breakpoint. All behaviour preceding the breakpoint is captured by the sub-protocol \( P_0 \) and all subsequent behaviour is modelled by \( P_1 \). Dividing the protocol into

\(^1\) Failing to discover an attack does not imply that the protocol is correct, and we must be careful in labelling it as such. Crucially, the results of this paper do not depend on the correctness of this protocol.
two allows us to verify each half of the protocol separately, using two different
rank functions, ρ₀ for P₀ and ρ₁ for P₁, where ρ₀(tmp) = 0 and ρ₁(tmp) = 1. We define a procedure for dividing a protocol and prove a result that guarantees
the correctness of the full protocol given the correctness of its sub-protocols.

4.1 Establishing the breakpoint

Given a CSP protocol model that contains a temporary secret tmp, our goal
is to divide the model into two, such that tmp is a secret value in the first
model and a public value in the second model. From Heather’s protocol it is
clear that the temporary secret, na, becomes public in message 3. Some thought
leads us to conclude that the division needs to occur between a’s receipt of
message 2 and her transmission of message 3. In the CSP model, we represent
this by inserting the break event between the appropriate rec and trans events:

\[ rec.\{b, na?k\}_PK(4) \rightarrow break \rightarrow trans.\{a, b\}_k.\{m\}_k. na. \]

It should be noted that other protocols may need a breakpoint to be per-
formed by the responder, or even by a trusted server. Although the following
definitions assume that Init performs the break event, it is by no means a re-
quirement, and the definitions could be altered to suit other situations.

4.2 Dividing the protocol

We now define the traces of NET up to the breakpoint and the (suffixes of) traces that follow the breakpoint.

**Definition 5.** Given a process NET in which one agent performs a single oc-
currence of the break event, define TR₀ to be the traces of NET which either do not contain break or have break as their final event:

\[ TR₀ = \{ tr \in traces(NET) \mid break \notin tr \lor tr = s \wedge \langle break \rangle \} \]

Since traces are prefix-closed we have that \( s \wedge t \in TR₀ \Rightarrow s \in TR₀ \).

**Definition 6.** Define TR₁ to be the set of sequences of events that follow break
on NET:

\[ TR₁ = \{ tr \mid s \wedge \langle break \rangle \wedge tr \in traces(NET) \} \]

Quite simply, TR₀ represents the behaviour of NET before the temporary secret
is made public and TR₁ represents all subsequent behaviour.

**Definition 7.** Given TR₀ and TR₁, we define NET₀ to be a process such that
TR₀ \( \subseteq \) traces(NET₀), and NET₁ to be a process such that TR₁ \( \subseteq \) traces(NET₁).

**Theorem 2.** Given NET, NET₀ and NET₁ as defined above:

\[ NET₀; NET₁ \sqsubseteq T NET \]

*Proof.* Omitted, see [1]. (The semicolon denotes a sequential composition.) The
trace refinement of NET₀; NET₁ by NET guarantees that any trace specification
satisfied by NET₀; NET₁ will also be satisfied by NET.
We have defined the sub-protocols \( \text{NET}_0 \) and \( \text{NET}_1 \) in terms of their traces. However, rank functions do not operate directly on the traces of the process but on the processes themselves. Our goal is therefore to discover suitable processes \( \text{NET}_0 \) and \( \text{NET}_1 \) for a given process \( \text{NET} \).

### 4.3 Constructing the sub-protocols

Given a division of \( \text{NET} \) based on sets of traces \( \text{TR}_0 \) and \( \text{TR}_1 \) our task is now to discover processes \( \text{NET}_0 \) and \( \text{NET}_1 \) such that \( \text{TR}_0 \subseteq \text{traces} (\text{NET}_0) \) and \( \text{TR}_1 \subseteq \text{traces} (\text{NET}_1) \).

A CSP process \( P(x) = c.x \to P' \) can be completely described in terms of its name, \( P \), its set of input parameters, \( \{x\} \), and a sequence containing the events in the body of the process, \( (c.x, P') \).

**Definition 8.** For a process \( P \), let \( \text{sig}(P) \) be the set of input parameters (the signature) for \( P \) and let \( \text{events}(P) \) be a sequence representing the events in the body of \( P \).

Based on this, we define the input variables of \( P \) to be those that are initialised by input in process \( P \). These are precisely the variables of \( P \) that are not mentioned in the signature of \( P \):

**Definition 9.** For a process \( P \), \( \text{inputvars}(P) = \text{variables}(P) \setminus \text{sig}(P) \)

The following process has \( \text{variables}(P) = \{a,b,k,na\} \), \( \text{sig}(P) = \{a,b,k\} \) and \( \text{inputvars}(P) = \{na\} \):

\[
P(a,b,k) = \text{rec}.a.b.\{b?na\}P_{K(a)} \rightarrow \text{trans}.a.b.\{a,na,k\}P_{K(b)} \rightarrow \text{Stop}
\]

The key to dividing the protocol is to divide each of \( \text{Init} \), \( \text{Resp} \) and \( \text{ENEMY} \) into 0- and 1- subscripted halves (e.g., \( \text{Init}_0 \) and \( \text{Init}_1 \)). Once these processes have been derived the construction of the networks \( \text{NET}_0 \) and \( \text{NET}_1 \) is a somewhat mechanical affair. Consequently, we present methods for dividing the initiator, responder and intruder processes, but omit the details of the network construction. The interested reader may find a complete description in the associated technical report [1].

**Constructing the divided initiator** We divide \( \text{Init} \) into two processes, \( \text{Init}_0 \) and \( \text{Init}_1 \).

**Definition 10.** Let \( \text{Init}_0 \) be the process that represents the behaviour of \( \text{Init} \) until the event break is reached and then successfully terminates:

\[
\text{sig}(\text{Init}_0) = \text{sig}(\text{Init})
\]

\[
\text{events}(\text{Init}_0) = \text{cs} \cap \langle \text{Skip} \rangle \cdot \text{cs} \leq \text{events}(\text{Init}) \land \text{last}(\text{cs}) = \text{break}
\]

The process \( \text{Init}_1 \) models all subsequent behaviour:
Definition 11. Let \( \text{Init}_1 \) be the process that represents the behaviour of \( \text{Init} \) after the breakpoint event has been reached:

\[
\begin{align*}
\text{sig}(\text{Init}_1) &= \text{sig}(\text{Init}) \cup \text{inputvars}(\text{Init}_0) \\
\text{events}(\text{Init}_1) &= es \cdot \text{events}(\text{Init}) = es' \land \{\text{break} \} \land es
\end{align*}
\]

Based on the definition of \( \text{Init} \) in Section 2.1, we have \( \text{sig}(\text{Init}_0) = \{a, b, na, m\} \) and \( \text{events}(\text{Init}_0) = \langle \text{trans}.a.b.\{a,na\}PK(b), \text{rec}.a.b.\{b,na?k\}PK(a), \text{break} \rangle \). Since the variable \( k \) is introduced during \( \text{Init}_0 \) it must feature in the signature of \( \text{Init}_1 \). We therefore calculate \( \text{Init}_1 \) as \( \text{sig}(\text{Init}_1) = \{a, b, na, m, k\} \) and \( \text{events}(\text{Init}_1) = \langle \text{trans}.a.b.\{m\}_k, \text{na}, \text{Stop} \rangle \), yielding the following processes:

\[
\begin{align*}
\text{Init}_0(a, b, na, m) &= \text{Init}_1(a, b, na, m, k) \\
\text{trans}.a.b.\{a-na\}PK(b) &= \text{trans}.a.b.\{m\}_k-na \\
\rightarrow \text{rec}.a.b.\{b,na?k\}PK(a) &= \text{Stop} \\
\rightarrow \text{break} &\rightarrow \text{Skip}
\end{align*}
\]

Constructing the divided responder: Dividing the responder process is non-trivial, since we do not have a priori knowledge of \( \text{Resp} \)'s state when \( \text{Init} \) reaches the breakpoint. However, the challenge-response nature of many protocols means that we can deduce the state of \( \text{Resp} \) when \text{break} is performed. In Heather's protocol, for example, the initiator will not perform \text{break} until after she has received a response to her nonce challenge, \( na \). We can therefore be certain that (given the secrecy of long-term keys) \( \text{Resp} \) has at least performed \( \text{trans}.b.a.\{b,na,k\}PK(a) \) by the time that \text{break} is performed. We can formalise this by saying that:

\[
\text{NET sat } \{\text{trans}.b.a.\{b,na,k\}PK(a)\} \text{ precedes } \{\text{break}\}.
\]

We propose two algorithms that can calculate \( \text{Resp}_0 \) and \( \text{Resp}_1 \) based on such deductions. In the following, \#\( \text{Resp} \) denotes the number of events in \( \text{Resp} \). In other words, \#\( \text{Resp} \equiv \text{length}(\text{events}(\text{Resp})) \).

Definition 12. For a protocol model \( \text{NET} \) with agent processes \( \text{Init} \) (which performs \text{break}) and \( \text{Resp} \), the earliest breakpoint is the earliest execution point that \( \text{Resp} \) can reach before \( \text{Init} \) is able to perform \text{break}. The \text{ebp} is calculated in terms of \( \text{events}(\text{Resp}) \) and is defined recursively:

\[
\begin{align*}
\text{ebp}(\text{NET}) &= \text{ebp}'(\text{NET}, 1) \\
\text{ebp}'(\text{NET}, i) &= \begin{cases} 
  i - 1 & \text{if } \neg f(i) \\
  i & \text{if } f(i) \land i = \#\text{Resp} \\
  \text{ebp}'(\text{NET}, i + 1) & \text{otherwise}
\end{cases}
\end{align*}
\]

where \( f(i) = \begin{cases} 
  \text{true} & \text{if } \text{NET sat } \{\text{events}(\text{Resp}),i\} \text{ precedes } \{\text{break}\} \\
  \text{false} & \text{otherwise}
\end{cases} \)

In other words, \( \text{ebp}(\text{NET}) \) tests each event in \( \text{Resp} \) to determine whether it must precede \text{break} on \( \text{NET} \), and returns the index of the latest event that does so. Typically, a rank function would be used to determine the truth of \( \text{NET sat } \{e\} \text{ precedes } \{\text{break}\} \) for an event \( e \).
Definition 13. For a protocol model \( NET \) with agent processes Init (which performs break) and \( Resp \), the latest breakpoint is the latest execution point that \( Resp \) can reach before Init must perform break. The \( \mathbf{lbp} \) is calculated in terms of \( \text{events}(\text{Resp}) \) and is defined recursively:

\[
\mathbf{lbp}(NET) = \mathbf{lbp}'(NET, \mathbf{ebp}(NET) + 1)
\]

\[
\mathbf{lbp}(NET, i) = \begin{cases} 
  i - 1 & \text{if } g(i) \\
  i & \text{if } -g(i) \land i = \#\text{Resp} \\
  \mathbf{lbp}'(NET, i + 1) & \text{otherwise}
\end{cases}
\]

where \( g(i) = \begin{cases} 
  \text{true} & \text{if } \text{NET sat } \{ \text{break} \} \text{ precedes } \{ \text{events(Resp).i} \} \\
  \text{false} & \text{otherwise}
\end{cases} \)

The function \( \mathbf{lbp}(NET) \) tests each event in \( \text{Resp} \) following the \( \mathbf{ebp} \) to determine whether \text{break} must precede it on \( \text{NET} \) and returns the index of the last event that may precede \text{break}.

Definition 14. If, for a protocol model \( NET: \mathbf{ebp}(NET) = \mathbf{lbp}(NET) \), then we say that \( NET \) has a simple division, otherwise we say that \( NET \) has a complex division.

We make the sufficient but possibly unnecessary assumption that \( NET \) has a simple division. Briefly, if \( NET \) has a simple division then there exist message events \( i \) and \( j \) such that \( \text{events}(\text{Resp}) = es \land (i, j) \land es' \) and \( i \) must occur before the breakpoint and \( j \) must occur after the breakpoint. If this property is not met, then \( \text{Resp} \) may be at one of several points when \text{break} occurs, necessitating an external choice to be offered by \( \text{Resp}_0 \) and \( \text{Resp}_1 \). This external choice will introduce behaviour into the divided model that is not possible in the original protocol - behaviour, moreover, that \textit{may} introduce new attacks. The following two definitions only apply to protocols that satisfy the assumption.

Definition 15. Let \( \text{Resp}_0 \) be the process that models the behaviour of \( \text{Resp} \) in \( NET \) before \text{Init} performs break:

\[
\text{sig}(\text{Resp}_0) = \text{sig}(\text{Resp})
\]

\[
\text{events}(\text{Resp}_0) = es \land (\text{break}, \text{Skip}) \land es \leq \text{events}(\text{Resp}) \land \text{length}(es) = \mathbf{ebp}(NET)
\]

It turns out that Heather’s protocol has a simple division with \( \mathbf{ebp}(NET) = \mathbf{lbp}(NET) = 2 \) and we can construct \( \text{Resp}_0 \) using the above definition. We omit the lengthy calculation of these values in preference to an informal justification. Assuming the secrecy of the responder’s private key \( SK(b) \), any nonce \( na \) sent to \( b \) will be secret and on receipt of the message \( \{ b, na, k \}_{PK(a)} \) the initiator \( a \) can verify that the received nonce matches the initial challenge. Consequently, \( a \) will not perform \text{break} until she has received this message, which is transmitted by \( b \) as message 2. Therefore, \( \mathbf{ebp}(NET) = 2 \). Similarly, the responder \( b \) cannot receive a message encrypted under \( k \) until \( a \) has received \( k \) and performed \text{break}, so \( \text{Resp} \) cannot perform its third event, \( \text{name} b \, a \, \{ m \}_k \, na \), until after the breakpoint resulting in \( \mathbf{lbp}(NET) = 3 - 1 = 2 \).
Definition 16. Let $Resp_1$ be the process that models the behaviour of $Resp$ in NET after $Init$ performs break:

\[
\begin{align*}
\text{sig}(Resp_1) &= \text{sig}(Resp) \cup \text{inputvars}(Resp_0) \\
\text{events}(Resp_1) &= es \cdot es = \text{events}(Resp).ebp(NET) + 1 \\
\end{align*}
\]

Based on the definitions of $lbp(NET)$ and $ebp(NET)$ we construct $Resp_0$ as the prefix of $Resp$ up to (and including) the second event, and construct $Resp_1$ as the suffix of $Resp$ that follows the second event. The responder processes are therefore:

\[
\begin{align*}
Resp_0(b, k) &= \text{rec.b?a.a?na}_P K(b) \\
\quad \rightarrow \text{trans.b.a.a?na}_P K(a) \\
\quad \rightarrow \text{break} \rightarrow \text{Skip} \\
Resp_1(b, k, a, na) &= \text{rec.b.a?m}_a.na \\
\quad \rightarrow \text{Stop}
\end{align*}
\]

Constructing the divided intruder The intruder process $ENEMY_0$ models the behaviour of the intruder until $Init$ performs break. The process $ENEMY_1$ models the intruder’s subsequent behaviour. The standard intruder process is defined as $ENEMY = Enemy(IK)$, where $IK$ is the intruder’s initial knowledge.

Definition 17. For a set $IK \subseteq M$, we define:

\[ENEMY_0 = Enemy(IK_0) \triangledown \text{break} \rightarrow \text{Skip}\]

That is, $ENEMY_0$ is the process $Enemy(IK_0)$ but can be interrupted at any point and perform break before successfully terminating. The set $IK_0$ is the intruder’s initial knowledge for sub-protocol 0.

Definition 18. For a set $IK_1 \subseteq M$, we define $ENEMY_1 = Enemy(IK_1)$

The process $ENEMY_1$ is the process $Enemy$, parameterised by $IK_1$, the set of messages known to the intruder at the start of sub-protocol 1.

Although we omit details of the network construction, it should be noted that $NET_0$ and $NET_1$ are constructed analogously to $NET$: using the 0- and 1-subscripted processes, respectively. All processes in $NET_0$ and $NET_1$ synchronise on the break event.

4.4 The main results

In this section we state the main results of this paper: that a protocol model $NET$ can be divided (under certain assumptions) into sub-protocol processes $NET_0$ and $NET_1$ according to the method proposed above such that the sequential composition $NET_0; NET_1$ is trace-refined by $NET$, and that the existence of suitable rank functions for $NET_0$ and $NET_1$ guarantees the correctness of $NET$. For brevity all proofs are omitted but may be found in [1].
Theorem 3. Given processes NET₀ and NET₁ and a set \( T \subseteq \mathcal{M} \):

\[
\text{NET₀ sat secrecy}(T) \land \text{NET₁ sat secrecy}(T) \Rightarrow \\
\text{NET₀; NET₁ sat secrecy}(T)
\]

This result states that if NET₀ and NET₁ each preserve the secrecy of the set \( T \) then the sequential composition NET₀; NET₁ also preserves the secrecy of \( T \).

Theorem 4. Given a protocol model NET that has a simple division, the sets of traces \( TR₀ \) and \( TR₁ \), and the processes NET₀ and NET₁ constructed by the method proposed in Section 4.3, then \( TR₀ \subseteq \text{traces}(NET₀) \land TR₁ \subseteq \text{traces}(NET₁) \).

This result guarantees that a suitably constructed NET₀ and NET₁ respectively capture the behaviour of NET before and after the breakpoint. The proof of this theorem prompts a further proof obligation to be introduced, namely that for ENEMY₁(IIK₁), the set IIK₁ must contain all messages that the intruder could have learnt during NET₀. This makes intuitive sense since the intruder for NET₁ is not starting fresh but has some knowledge accumulated by eavesdropping on (and possibly interfering with) the first half of the protocol.

These results state that if we can find rank functions \( \rho₀ \) and \( \rho₁ \) to prove that NET₀ and NET₁ (respectively) guarantee the secrecy of \( m \) and \( k \) then it is also the case that \( \text{NET sat secrecy}(\{m, k\}) \). Such rank functions do exist, and can be found in [1].

5 Conclusion

In this paper we have presented a formal treatment of temporary secrets in security protocols, and have proposed a method of verifying two-party protocols that use a single temporary secret on a small network. In doing so we have made significant progress in overcoming a stubborn incompleteness in the rank functions framework.

Nonetheless, the present work is limited on several fronts. Firstly, by verifying single protocol runs we cannot unearth attacks that manifest themselves in subsequent runs. Secondly, the current definitions do not apply to protocols that use more than one temporary secret. Finally, our approach is restricted to the verification of two-party protocols. It certainly seems possible to extend our definitions to apply to three-party or group protocols and allowing more than one temporary secret also appears to be a simple matter.

In light of this paper, it is worth reconsidering the question of completeness. The standard rank functions approach failed to meet a weak definition of incompleteness, which we can recast in terms of combined rank functions as: does every secure composition of a small network and a confidentiality protocol have an associated combination of rank functions? Since the current theory allows a combination of at most two rank functions it is still incomplete: there exist secure protocols whose proof requires a combination of more than two rank functions. In general, a protocol that uses \( n \) temporary secrets may need a combination of up to \( n + 1 \) rank functions for the purposes of verification.
The motivation for this work was borne of a simple desire to overcome an incompleteness in the rank functions approach, and as such is merely a theoretical nicety. The result will only be of practical interest if, by overcoming the incompleteness, we widen the class of meaningful protocols that can be verified using the approach. It turns out that meaningful protocols do exist, notably the TESLA broadcast authentication protocol [9]. In this protocol signed messages are broadcast on a network and the recipients of these messages cannot check the authenticity of the signature until a later time, when the sender reveals the signing key. Clearly, the signing key is a temporary secret. Since our result deals with secrecy properties it is not yet clear whether it can be used to investigate the security of TESLA and further work is needed to confirm this. Initial work also suggests that a treatment of temporary secrets will be of use when modelling the effects of compromised keys, which may be considered as temporary secrets. Such an approach would allow properties such as forward secrecy and resistance to known-key attacks to be stated and verified within the rank functions framework.

Acknowledgements

We are grateful to Peter Ryan for his comments on this work, to Dave Cohen for reading an earlier draft and to the anonymous referees for their helpful feedback.

References