Security Analysis using Rank Functions in CSP

Communicating Sequential Processes (CSP) is an abstract language for describing processes and reasoning about their interactions within concurrent systems. It is appropriate for investigating the overall behaviour that emerges. It has a mature theory, and powerful tool support [For03] and can be considered as an off-the-shelf framework which can be customised for particular domains through additional domain-specific constructions and theory. This chapter introduces the application of CSP to the analysis of security protocols. For the purposes of this chapter, we will introduce only those parts of CSP that we use in protocol analysis. Fuller descriptions of the language and theory can be found in [Hoa85,Ros97,Sch99].

1. Review of CSP

In CSP, processes are described in terms of the patterns of *events* that they can perform. Events are occurrences in the lifetime of a process, including communications with other processes, as well as other actions that they might be performing. In our context, events will typically have several components, such as *trans.A.B.m* representing communication of a message *m* over channel *trans.A.B*. Messages will themselves have some specific structures (to include encryption and signing), and are drawn from a defined set of messages.

Processes are described using a language comprised of a number of process constructors:

Prefix If e is an event, and P is a process, then $e \rightarrow P$ is the process which initially is prepared only to perform event e, after which it behaves as P.

Output For a channel c and value v, the process $c!v \rightarrow P$ outputs v along channel c, and subsequently behaves as P. Semantically it is equivalent to $c.v \rightarrow P$, with the '!' symbol used to indicate output.

Input If $P(x)$ is a family of processes indexed by x, then $c?x \rightarrow P(x)$ is a process which initially reads a value x on channel c, and subsequently behaves as $P(x)$. We also make frequent use of *pattern matching*. We can input messages of a particular form or with some particular values, by giving the general pattern. Only inputs that match the pattern are accepted, and the variables in the pattern are bound according to the value received. For example, to accept triples in which the second value is 3, we could use the input $c?(x,3,y) \rightarrow P(x,y)$. As another example, to accept messages encrypted only with a particular key K we could use the input $c?\{|m|\}^{\mathsf{S}}_{K} \rightarrow P(m)$.

Termination The process *STOP* indicates termination (or deadlock): this process can perform no further events.

Choice If $P(i)$ is a finite or infinite family of processes indexed by $i \in I$, then the process $\Box_i P(i)$ offers the choice of all of the $P(i)$ processes, and can behave as any of them. The choice is made on occurrence of the first event.

Interleaving The process $|||_i P(i)$ is the interleaving of all of the $P(i)$ processes. All of the $P(i)$ processes run independently and concurrently, and can be scheduled in any order. There is also a binary form $P \;|||\; Q$.

Parallel The parallel combination $P \,|[\, A \,]|\, Q$ of two processes runs P and Q concurrently, but they must synchronise on all events in the set A. A special case is the parallel combination $P \parallel Q$, which requires synchronisation on all events. This synchronisation mechanism is the way in which processes interact with each other. For example, the processes in $(c!v \rightarrow P) \parallel (c?x \rightarrow Q(x))$ share the channel c, so the parallel combination will communicate the value v along c, and subsequently behave as $P \parallel Q(v)$. If A is a singleton set $\{a\}$ then the set brackets may be elided. Thus the combination $P \,|[\, a \,]|\, STOP$ behaves as P blocked on the event a.

Recursion Processes can be defined recursively, using (parameterised) process names: the definition $N(p) \mathrel{\widehat{=}} P(p)$ defines process $N(p)$ with parameter p to behave as $P(p)$. The name N also appears in the body of P, corresponding to a recursive call.

The language also includes internal choice, abstraction, timeout, event renaming, and interrupts, but these are not needed here so will not be considered further.

The semantic foundation we will use is the *traces model* for CSP. A *trace* is a (finite) sequence of events. Traces are written as sequences of events listed between angled brackets: $\langle e_1, \ldots, e_n \rangle$.

The traces model associates every CSP process with a set of traces, consisting of all the traces that might possibly be observed in some execution of that process. For example,

$$\begin{aligned} \mathit{traces}(\mathit{in}?x \rightarrow \mathit{out}!x \rightarrow \mathit{STOP}) = {} & \{\langle\rangle\} \\ & \cup \{\langle \mathit{in}.v \mid v \in M \rangle\} \\ & \cup \{\langle \mathit{in}.v, \mathit{out}.v \mid v \in M \rangle\} \end{aligned}$$

where M is the type of channel in.

The theory of CSP gives ways of reasoning directly about the set of traces of any system described in CSP.

Specifications are concerned with allowable system behaviours. A trace specification will describe which traces are allowable, and a CSP system can then be checked against the specification by considering all its traces and demonstrating that they are all acceptable. We write P **sat** S to indicate that every trace of $\mathit{traces}(P)$ meets the predicate S. For example, consider the predicate a **precedes** b defined on traces as follows:

$$a \textbf{ precedes } b = (\mathit{last}(tr) = b \Rightarrow a \textbf{ in } tr)$$

where $\mathit{last}(tr)$ is the last event in trace tr (undefined if $tr = \langle\rangle$), and a **in** tr means that a appears somewhere in the trace tr. Then

$$(a \rightarrow b \rightarrow \mathit{STOP}) \textbf{ sat } a \textbf{ precedes } b$$

2. Protocol modelling and verification

We will apply this framework to security protocol analysis. This involves developing a CSP model of a system which expresses the protocol, and specifying the security property to be considered. In order to judge the CSP model against the specification we make use of the 'rank function theorem' which is at the heart of the approach. We use the Corrected handshake protocol of Chapter *"Introduction"* Figure 1 as a running example, and begin with the standard notation:

$$A \to B : \left\{\left| [A.B.k]_{sk(A)} \right|\right\}^{\mathsf{a}}_{pk(B)} \qquad [k \text{ fresh}]$$
$$B \to A : \{|s|\}^{\mathsf{s}}_{k}$$

This protocol description is written in terms of the messages to be exchanged between the protocol participants, and describes how each protocol participant is expected to behave.

In order to model this protocol in CSP, we need to define the set of possible messages that can be communicated. For this protocol we must consider that messages can be constructed using concatenation, shared key encryption, public key encryption, and signing. Participant identifiers, shared keys, and plain text are available as message components. We define the space of messages as follows, and consider all messages in the CSP model to come from this set:

$M_1, M_2 ::=$		messages
I	$(\in USER)$	agent identities
S	$(\in PLAIN)$	plaintext
K	$(\in KEY)$	shared keys
$M_1.M_2$		concatenation of messages
$\{\|M\|\}^{\mathsf{s}}_{K}$		symmetric encryption of message M by key K
$\{\|M\|\}^{\mathsf{a}}_{pk(I)}$		asymmetric encryption of message M by agent I's public key
$[M]_{sk(I)}$		signing of message M with agent I's signature key

We assume sets *USER* of user identifiers; *PLAIN* of plaintext messages (from which the payload in message 2 is drawn), and *KEY* of shared keys. This approach treats messages that are constructed differently as different, thus building into the model that any particular message can be constructed in only one way, an assumption known as *perfect encryption* [PQ00].

We are now in a position to model the protocol participants. The protocol initiator, denoted A, chooses a fresh key k and a protocol partner j, assembles the first message by signing and then encrypting with j's public key, and transmits it to j. A then awaits a message encrypted with the key k. A's run of the protocol may be described in CSP as follows:

$$INIT_A(k) = \Box_j \, trans.A!j!\left\{\left| [A.j.k]_{sk(A)} \right|\right\}^{\mathsf{a}}_{pk(j)}$$
$$\to rec.A.j?\{|s|\}^{\mathsf{s}}_{k} \to STOP$$

We use channel *trans* for outputs from a protocol participant, and channel *rec* for inputs. $trans.A.j.m$ is a transmission of message m from A, intended for j. As we shall see in the

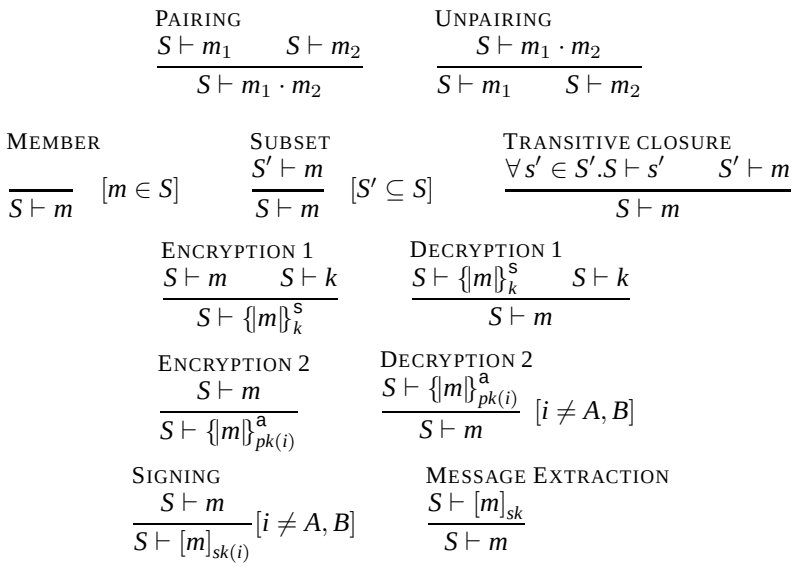

Figure 1. Attacker inference rules

model, we allow for the fact that it might not reach j. Observe the initial choice of j, and the use of pattern matching in the received message: the key of the input message must be k, though any message s can be accepted as the contents.

The protocol responder, denoted B, receives an initial message signed by i and encrypted with his public key $pk(B)$. That message contains a key k, which B then uses to encrypt a response s.

$$RESP_B(s) = rec.B?i?\left\{\left|[i.B.k]_{sk(i)}\right|\right\}^{a}_{pk(B)}$$
$$\rightarrow trans.B!i!\{|s|\}^{s}_{k} \rightarrow STOP$$

We must also model the hostile environment within which A and B communicate. We use the Dolev-Yao model [DY83], in which the attacker has complete control of the network. Messages are transmitted to the attacker for forwarding to the intended recipient; and messages received come from the attacker. The attacker can also divert, invent, replay, destroy, and alter messages. We also assume that the attacker can participate in protocol runs as a legitimate user, so some of the user identities in *USER* are under the control of the attacker (i.e. the attacker can decrypt messages encrypted under their public keys, and can sign with their signature keys).

Conversely, the attacker may not decrypt messages without possession of the decryption key; nor sign messages without knowledge of the signature key. The messages that the attacker is able to generate, and thus use in an attack, are limited by the attacker's knowledge. Figure 2 gives the rules that govern how the attacker can generate new messages: $\vdash$ is defined as the smallest relation closed under all of these rules. Thus $S \vdash m$ means that the attacker can construct message m from the set of messages S. Note that rules DECRYPTION 2 and SIGNING encapsulate the assumption that the attacker controls any users other than A and B.

The CSP model of the attacker will be a process *ENEMY* parameterised by the set S of messages that the attacker knows. This will be a combination of those messages known initially, together with those sent out by protocol participants. $ENEMY(S)$ can receive any message m from any user i sent to any other user j, in which case the set S is augmented with m. $ENEMY(S)$ can also supply any message m that can be generated from S, to any user i, as if it came from user j.

$$ENEMY(S) = trans?i?j?m \rightarrow ENEMY(S \cup \{m\})$$
$$\Box$$
$$\Box_{\substack{i \in USER \\ j \in USER \\ m|S \vdash m}} rec!i!j!m \rightarrow ENEMY(S)$$

This defines the threat model in terms of attacker *capabilities*. Correctness in the context of this attacker indicates that there are no attacks from an attacker with these capabilities. The attacker behaviour includes the possibility of passing messages on correctly, as well as the standard manouevres used in attacks: blocking, redirecting, spoofing, combining, dissecting, and replaying messages. However, the key point in this model is that the attacker is unable to attack the cryptographic mechanisms.

The definition of the attacker in the model will need to identify the initial knowledge: a set *IK* of messages that the attacker is considered to have available initially. This will include all user names, some plaintext, and some session keys. However, any fresh keys or text used by honest agents in protocol runs will not be in *IK*, to model our expectation that the attacker should not be able to guess them. The attacker can learn them only through protocol runs.

We then define *ENEMY* to be $ENEMY(IK)$.

A model of a single run of the protocol with A as initiator and B as responder, with specific key k_{AB} and secret s_{AB} will be:

$$SYS = (INIT_A(k_{AB}) \;|||\; RESP_B(s_{AB})) \;|[\, trans, rec \,]|\; ENEMY$$

A and B do not synchronise on any events directly: all of their communications are via *ENEMY*. On the other hand *trans* and *rec* are in the alphabets of both sides of the parallel operator, so all occurrences of *trans* and *rec* involve one of the protocol participants, and also *ENEMY*. Thus *ENEMY* is involved in all communications within the system.

SYS describes a model involving just a single pair of protocol participants A and B on a single run of the protocol. In order to explain the essence of the approach we will consider this model through the majority of this chapter. However, this can be generalised to the case of arbitrarily many concurrent runs, discussed in Section 6.

2.1. Specification

Having developed the model of the protocol, we now consider the properties that we wish to demonstrate. We will consider authentication of each party by the other, and secrecy.

Authentication is concerned with establishing the identity of the other party. If A runs the protocol with B, then the intention is that by the end of the protocol run A can be confident that the protocol was indeed run with B. From B's point of view, B wishes to be confident that a protocol run apparently with A was indeed with A.

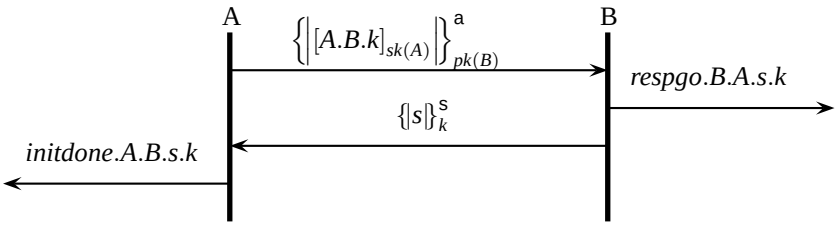

Figure 2. Introducing specification events for A authenticating B

This can be captured within the traces of *SYS*, stating that any trace which includes a protocol run of A with B should also contain a corresponding protocol run of B with A. However, we find it cleaner to introduce particular specification events into the protocol to state explicitly that a protocol participant has reached a particular stage, and what information they have used in that protocol run. By instrumenting the protocol with such events we can give specifications directly on those events rather than implicitly in terms of the sequence of events that have gone previously.

Thus for A to authenticate B we add an event *initdone*.A.B.s.k which A uses to signal completion of a protocol run with B, with key k, where s is the plaintext that has been received in message 2. We also add an event *respgo*.B.A.s.k into B's run to indicate that B is running the protocol with A, with plaintext s, and with key k received in message 1. For authentication, *respgo* needs to appear causally prior to *initdone* in the protocol run. The placement of these additional events is illustrated in Figure 2.

The precise authentication property can be varied by varying the specification events selected. Agreement purely on the key is captured by the pair *initdone*.A.B.k and *respgo*.B.A.k; agreement only on the protocol participants would be captured by using the pair *initdone*.A.B and *respgo*.B.A. This would provide A with assurance that the other participant is B, but no assurance that they agree on the key k or text s. A hierarchy of authentication properties is discussed in [Low97].

To consider the authentication property, the descriptions of $INIT_A$ and $RESP_B$ are adjusted to incorporate these specification events:

$$\begin{aligned} INIT_A(k_{AB}) = {} & trans.A!B!\{|[A.B.k_{AB}]_{sk(A)}|\}^{a}_{pk(B)} \\ & \rightarrow rec.A.B?\{|s|\}^{s}_{k_{AB}} \\ & \rightarrow initdone.A.B.s.k_{AB} \rightarrow STOP \end{aligned}$$

$$\begin{aligned} RESP_B(s_{AB}) = {} & rec.B?j?\{|[j.B.k]_{sk(j)}|\}^{a}_{pk(B)} \\ & \rightarrow respgo.B.j.s_{AB}.k \\ & \rightarrow trans.B!j!\{|s_{AB}|\}^{s}_{k} \rightarrow STOP \end{aligned}$$

The authentication property from *A*'s point of view is that whenever the signal even *initdone*.*A*.*B*.*s*.*k* occurs with specific *s* and *k*, then *respgo*.*B*.*A*.*s*.*k* should previously have occurred, with the same *s* and *k*. This will indicate that *B* is running the protocol, with *A*, and that they agree on the content of the protocol run. This requirement as a trace specification is that any trace of *SYS* which contains *initdone*.*A*.*B*.*s*.*k* also contains *respgo*.*B*.*A*.*s*.*k*. The attacker cannot perform *initdone* or *respgo* events since they have been introduced purely for the purposes of specification—they are modelling points in the protocol rather than communications that the attacker can engage in.

A violation of the property will be a trace of *SYS* in which *A* performs *initdone*.*A*.*B*.*s*.*k* without *B* having previously performed the corresponding *respgo*.*B*.*A*.*s*.*k*. In that case *A* will have been brought to a point where the protocol run apparently with *B* has completed, but *B* has not been involved in the same protocol run. This will be either because there is a mistake in the protocol, or because an attack is possible.

We fix (arbitrary) key k_{AB} and text s_{AB}, and consider authentication with respect to these. The form of specification *SYS* is required to satisfy is then:

$$\mathit{respgo}.B.A.k_{AB}.s_{AB} \textbf{ precedes } \mathit{initdone}.A.B.k_{AB}.s_{AB}$$

which states that any trace in which the *initdone* event appears must have the corresponding *respgo* appear earlier in the trace.

2.2. A general theorem for proving authentication

The following theorem gives conditions for establishing that an event *a* precedes another event *b* in a network *SYS* consisting of a number of users USER_i in parallel with *ENEMY*. It makes use of a *rank function* ρ which associates messages and signals with integers. If every component within the system can only introduce messages of positive rank when *a* is blocked, and if *b* has non-positive rank, then it follows that *b* cannot occur when *a* is blocked. Thus in the unblocked system any occurrence of *b* must follow an occurrence of *a*.

In the theorem, conditions 1 and 2 establish that *ENEMY* cannot introduce messages of non-positive rank; condition 3 states that *b* has non-positive rank; and condition 4 states that if each user *i* only receives messages of positive rank, then it can communicate messages and signals only of positive rank.

Rank Function Theorem

If $\rho : \mathit{MESSAGE} \cup \mathit{SIGNAL} \to \mathbb{Z}$ is such that:

1. $\forall m \in \mathit{IK}.\rho(m) > 0$
2. $\forall S \subseteq \mathit{MESSAGE}.(\rho(S) > 0 \wedge S \vdash m) \Rightarrow \rho(m) > 0$
3. $\rho(b) \leqslant 0$
4. $\forall i.(\mathit{USER}_i \,|[\, a \,]|\, \mathit{Stop})$ **sat** $\rho(tr \upharpoonright \mathit{rec}) > 0 \Rightarrow \rho(tr) > 0$

then $(|||_i \mathit{USER}_i) \,|[\, \mathit{trans}, \mathit{rec} \,]|\, \mathit{ENEMY}$ **sat** *a* **precedes** *b*. □

In condition 4, the notation $tr \upharpoonright \mathit{rec}$ denotes the projection of trace *tr* onto the channel *rec*: in other words, the subsequence of *rec* events occurring within *tr*. This requirement on USER_i blocked on *a* is that if only positive rank messages are received, then no non-positive rank message should be produced. The proof of the theorem is given in [Sch98b].

We have abused notation and extended ρ to apply not only to messages and signals, but also to events, traces, and sets:

- $\rho(c.m) = \rho(m)$
- $\rho(S) = \min\{\rho(s) \mid s \in S\}$
- $\rho(tr) = \min\{\rho(s) \mid s \textbf{ in } tr\}$

For any particular protocol specification the challenge is to identify a suitable ρ that meets the conditions. Identification of such a ρ establishes correctness of the protocol with respect to that specification.

2.3. Application of the theorem

We require in this particular case that:

1. every message in *IK* has positive rank;
2. if every message in a set *S* has positive rank, and $S \vdash m$, then *m* has positive rank;
3. $initdone.A.B.k_{AB}.s_{AB}$ does not have positive rank;
4. $INIT_A(k_{AB})$ maintains positive rank: if it has only received messages of positive rank then it only outputs messages of positive rank. Note that $INIT(k_{AB}) \,|[\, respgo.B.A.s_{AB}.k_{AB} \,]|\, STOP = INIT_A(k_{AB})$;
5. $RESP_B(s_{AB}) \,|[\, respgo.B.A.s_{AB}.k_{AB} \,]|\, STOP$ maintains positive rank: if it has only received messages of positive rank then it only outputs messages of positive rank. Observe we are considering $RESP_B$ with the *respgo* event blocked.

If we can find a rank function that meets all these conditions, then we will have established that *SYS* satisfies $respgo.B.A.k_{AB}.s_{AB}$ **precedes** $initdone.A.B.k_{AB}.s_{AB}$, and hence that the protocol provides the authentication guarantee required.

Figure 3 gives a rank function that meets all of the required properties.

1. We assume that $k_{AB} \notin IK$ since it is fresh for user *A*. Thus all the messages in *IK* will have positive rank.
2. This condition is established inductively over the inference rules. In particular, we can check for each rule in turn that if it is true for the premisses, then it is also true for the conclusion.
3. $initdone.A.B.k_{AB}.s_{AB}$ does not have positive rank, by definition of ρ.
4. $INIT_A(k_{AB})$ maintains positive rank. It outputs a single protocol message, which has positive rank; and it can only perform the final $initdone.A.B.s_{AB}.k_{AB}$ event if it has previously received a message of rank 0: a message encrypted with k_{AB}. Thus if it only receives messages of positive rank it will only perform events of positive rank.
5. It is useful first to expand the restricted $RESP_B$:

$$
\begin{aligned}
&RESP_B(s_{AB}) \,|[\, respgo.B.A.s_{AB}.k_{AB} \,]|\, STOP \\
&= rec.B?j?\left\{\left|[j.B.k]_{sk(j)}\right|\right\}^{\mathsf{a}}_{pk(B)} \rightarrow \begin{array}{l} \text{if } (j = A \wedge k = k_{AB}) \\ \quad \text{then } STOP \\ \quad \text{else } respgo.B.j.s_{AB}.k \\ \qquad \rightarrow trans.B!j!\{|s_{AB}|\}^{\mathsf{s}}_{k} \rightarrow STOP \end{array}
\end{aligned}
$$

   The only time *B* can send a message of rank 0 is when the received key *k* is in fact k_{AB}. In this case we must have $j \neq A$ to reach that point in the restricted protocol.

$$\rho(i) = 1$$
$$\rho(s) = 1$$
$$\rho(k) = \begin{cases} 0 \text{ if } k = k_{AB} \\ 1 \text{ otherwise} \end{cases}$$
$$\rho(m_1.m_2) = \min\{\rho(m_1).\rho(m_2)\}$$
$$\rho(\{|m|\}^{\mathsf{s}}_k) = \begin{cases} 0 & \text{if } k = k_{AB} \\ \rho(m) & \text{otherwise} \end{cases}$$
$$\rho(\{|m|\}^{\mathsf{a}}_{pk(i)}) = \begin{cases} 1 & \text{if } i = B \wedge m = [A.B.k']_{sk(A)} \\ \rho(m) & \text{otherwise} \end{cases}$$
$$\rho([m]_{sk(i)}) = \begin{cases} 0 & \text{if } m = i.B.k_{AB} \\ \rho(m) & \text{otherwise} \end{cases}$$
$$\rho(sig) = \begin{cases} 0 \text{ if } sig = initdone.A.B.s_{AB}.k_{AB} \\ 1 \text{ otherwise} \end{cases}$$

Figure 3. A rank function for authentication

But then the rank of the received message is 0: $\rho(\left\{\left|[j.B.k]_{sk(j)}\right|\right\}^{\mathsf{a}}_{pk(B)}) = 0$. Hence transmission of a message of rank 0 follows receipt of a message of rank 0. Thus $RESP_B(s_{AB}) \,|\, [\, respgo.B.A.s_{AB}.k_{AB}\,] \,|\, STOP$ maintains positive rank.

We can conclude that A's run of the protocol authenticates B.

2.4. Protocol simplification

If the participants are removed from message 1 of the protocol, then we obtain the simplified (flawed) version:

$$A \to B : \left\{\left|[k]_{sk(A)}\right|\right\}^{\mathsf{a}}_{pk(B)}$$
$$B \to A : \{|s|\}^{\mathsf{s}}_k$$

We will consider how this simplification affects the correctness proof.

The natural change to make to the rank function is to change the message in the definition of ρ to follow the change in the protocol, resulting in the following alternative clauses (the other clauses are unaffected):

$$\rho(\{|m|\}^{\mathsf{a}}_{pk(i)}) = \begin{cases} 1 & \text{if } i = B \wedge m = [k']_{sk(A)} \\ \rho(m) & \text{otherwise} \end{cases}$$
$$\rho([m]_{sk(i)}) = \begin{cases} 0 & \text{if } m = k_{AB} \\ \rho(m) & \text{otherwise} \end{cases}$$

The models for analysis also change to reflect the simpler first message:

$$INIT_A(k_{AB})$$
$$= trans.A!B!\left\{\left|[k_{AB}]_{sk(A)}\right|\right\}^{a}_{pk(B)} \rightarrow rec.A.B?\{|s|\}^{s}_{k_{AB}} \rightarrow initdone.A.B.s.k_{AB} \rightarrow STOP$$

$$RESP_B(s_{AB}) \,|[\, respgo.B.A.s_{AB}.k_{AB} \,]|\, STOP$$
$$= rec.B?j?\left\{\left|[k]_{sk(j)}\right|\right\}^{a}_{pk(B)} \rightarrow \begin{array}{l}\text{if } (j = A \wedge k = k_{AB}) \\ \text{then } STOP \\ \text{else } respgo.B.j.s_{AB}.k \\ \qquad \rightarrow trans.B!j!\{|s_{AB}|\}^{s}_{k} \rightarrow STOP\end{array}$$

We find that the revised rank function with the revised CSP models still meets all the rank function properties. Thus the simplified (flawed!) protocol still establishes the authentication property that A authenticates B, and they agree on the session key k_{AB} and the secret message s_{AB}. Although flawed in other ways, it still provides this authentication property.

3. Responder authenticating initiator

The previous section verified that the initiator authenticates the responder. We are also concerned with authentication in the other direction. The same approach is taken: a pair of events to specify authentication are introduced into the model of the protocol; a suitable model of a protocol run is defined in CSP, this time from the responder's point of view; a rank function is identified which meets the properties of the rank function theorem, establishing the authentication property.

The authenticating events in this instance are *respdone* and *initgo*. The event *respdone* occurs after the message received from A. The event *initgo* should be causally prior to that message, so must occur before A's first communication. At that point A has the key k but not s, so the event will be $initgo.A.B.k$. This should be followed by $respgo.B.A.k$. This is pictured in Figure 4.

In the CSP model to analyse for this property, we are concerned with B's use of the first protocol message in authenticating the initiator. We therefore fix the user A that B is responding to, and the key k_{AB} that B receives in that message.

Since this authentication property is relative to B, we model A as being able to initiate with any party j, and with any key k. The rank function theorem requires restriction on $initgo.A.B.k_{AB}$. We therefore obtain the following processes for the initiator and the responder, which should maintain positive ρ for any proposed rank function ρ:

$$INIT_A \,|[\, initgo.A.B.k_{AB} \,]|\, STOP =$$
$$\Box_{j,k} \begin{array}{l}\text{if } j = B \wedge k = k_{AB} \\ \text{then } STOP \\ \quad \text{else } initgo.A.j.k \rightarrow trans.A!j!\left\{\left|[A.j.k]_{sk(A)}\right|\right\}^{a}_{pk(j)} \rightarrow rec.A.j?\{|s|\}^{s}_{k} \rightarrow STOP\end{array}$$

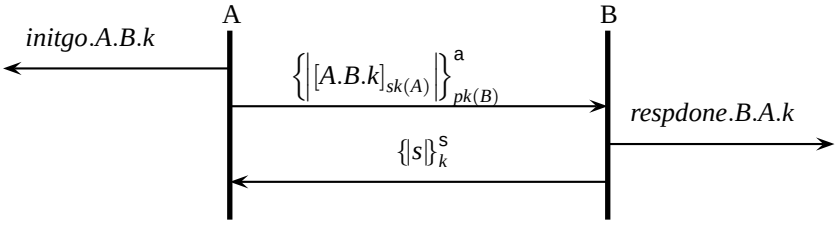

Figure 4. Introducing specification events for B authenticating A

$$RESP_B(s_{AB}, k_{AB}) =$$
$$rec.B.A?\{|[A.B.k_{AB}]_{sk(A)}|\}^{a}_{pk(B)} \rightarrow respdone.B.A.k_{AB} \rightarrow trans.B!A!\{|s_{AB}|\}^{s}_{k_{AB}} \rightarrow STOP$$

The following rank function meets all the conditions of the rank function theorem:

$$\rho(i) = 1$$
$$\rho(s) = 1$$
$$\rho(k) = 1$$
$$\rho(m_1.m_2) = \min\{\rho(m_1).\rho(m_2)\}$$
$$\rho(\{|m|\}^{s}_{k}) = \rho(m)$$
$$\rho(\{|m|\}^{a}_{pk(i)}) = \rho(m)$$
$$\rho([m]_{sk(i)}) = \begin{cases} 0 & \text{if } i = A \wedge m = A.B.k_{AB} \\ \rho(m) & \text{otherwise} \end{cases}$$
$$\rho(sig) = \begin{cases} 0 \text{ if } sig = respdone.B.A.k_{AB} \\ 1 \text{ otherwise} \end{cases}$$

This rank function captures the requirement that the enemy cannot generate or obtain the message $[A.B.k_{AB}]_{sk(A)}$, even if it knows k_{AB} (note that $\rho(k_{AB}) = 1$, allowing for the enemy to be able to generate it). This fact is sufficient to guarantee to B that A must have initiated the protocol run with B, with key k_{AB}, establishing authentication.

3.1. Protocol simplification

As previously, if the participants are removed from message 1 of the protocol, then we obtain the simplified version:

$$A \rightarrow B : \{|[k]_{sk(A)}|\}^{a}_{pk(B)}$$
$$B \rightarrow A : \{|s|\}^{s}_{k}$$

The revised CSP protocol descriptions are:

$$INIT_A \,|[\, initgo.A.B.k_{AB} \,]|\, STOP =$$

$$\Box_{j,k} \begin{array}{l} \text{if } j = B \wedge k = k_{AB} \\ \text{then } STOP \\ \text{else } initgo.A.j.k \rightarrow trans.A!j!\left\{\left|\, [k]_{sk(A)} \,\right|\right\}^{\mathsf{a}}_{pk(j)} \rightarrow rec.A.j?\{|s|\}^{\mathsf{s}}_{k} \rightarrow STOP \end{array}$$

$$RESP_B(s_{AB}, k_{AB}) =$$

$$rec.B.A?\left\{\left|\, [k_{AB}]_{sk(A)} \,\right|\right\}^{\mathsf{a}}_{pk(B)} \rightarrow respdone.B.A.k_{AB} \rightarrow trans.B!A!\{|s_{AB}|\}^{\mathsf{s}}_{k_{AB}} \rightarrow STOP$$

The natural change to the rank function is in the clause for signed messages, which becomes:

$$\rho(\{|m|\}^{\mathsf{a}}_{sk(i)}) = \begin{cases} 0 & \text{if } i = A \wedge m = k_{AB} \\ \rho(m) & \text{otherwise} \end{cases}$$

However, we now find that $INIT_A \,|[\, initgo.A.B.k_{AB} \,]|\, STOP$ no longer meets condition 4 of the rank function theorem, since if $k = k_{AB}$ and $j \neq B$ then it can communicate $trans.A.j.\left\{\left|\, [k_{AB}]_{sk(A)} \,\right|\right\}^{\mathsf{a}}_{pk(j)}$ and thus transmit a communication of rank 0 without having first received one.

Two responses to this observation are possible: either seek another rank function which does work; or explore if the reason the rank function fails is because there is an attack. In this case there is the man in the middle attack seen earlier in Chapter *"Introduction"* Figure 1: *B* accepts the first message as evidence that *A* has initiated the protocol with *B*, but in fact *A* might have initiated it with a different party. We write the attack as follows:

$$\begin{array}{ll} \alpha: & A \rightarrow E: \left\{\left|\, [k_{AB}]_{sk(A)} \,\right|\right\}^{\mathsf{a}}_{pk(E)} \\ \beta: & E(A) \rightarrow B: \left\{\left|\, [k_{AB}]_{sk(A)} \,\right|\right\}^{\mathsf{a}}_{pk(B)} \\ \beta: & B \rightarrow E(A): \{|s_0|\}^{\mathsf{s}}_{k_{AB}} \end{array}$$

where $E(A)$ represents E masquerading as A.

The predicate $initgo.A.B.k_{AB}$ **precedes** $respdone.B.A.k_{AB}$ is not met by the trace corresponding to this attack. The 'confirmation' signal $respdone.B.A.k_{AB}$ is in fact preceded by $initgo.A.E.k_{AB}$. Hence the authentication property is not satisfied by the simplified protocol.

4. Secrecy

There are also secrecy requirements on this protocol.

SECRECY_INIT The secrecy requirement for the initiator is that if *s* is accepted as secret after the protocol run, then *s* should not be known to the intruder (provided the responder is honest).

SECRECY_RESP Similarly, the secrecy requirement for the responder is that if s is sent in the protocol run, then it should not be known to the intruder (provided the initiator is honest).

The assumption of honesty in the other party is natural, since the secret is being shared with them—if the other party is dishonest then there can be no guarantees about secrecy.

For reasons of space we will carry out the analysis for *SECRECY_RESP*. The analysis for *SECRECY_INIT* is very similar.

4.1. Modeling for secrecy analysis

The intruder's acquisition of message s can be modelled by its capability to perform $trans.E.E.s$ or some other communication demonstrating possession of s. To establish that such communications cannot occur it is sufficient to provide a rank function such that s has rank 0. Secrecy is concerned with the impossibility of a particular communication, rather than establishing a precedence relationship between two events. Usefully, this can be expressed in the form required by the rank function theorem. The rank function theorem can be applied by introducing an impossible event *imp* which no participant performs: the statement *imp* **precedes** $trans.E.E.s$ is equivalent to requiring that $trans.E.E.s$ can never occur (since *imp* can never occur). Expressing it in the form *imp* **precedes** $trans.E.E.s$ allows direct application of the rank function theorem. Observe that in this case no additional specification events need to be introduced, and since *imp* is not in the alphabet of any process, restricting the system's behaviour on *imp* makes no difference to the behaviour of any of the participants.

4.1.1. SECRECY_RESP

The model for analysis of secrecy with respect to the responder B fixes on A as the initiator. $RESP_B$ therefore describes a run with A. The initiator A is modelled as following the protocol faithfully (since B assumes A is honest), though possibly with a different participant. Thus A chooses an arbitrary party j with whom to run the protocol.

$$INIT_A(k_0) =$$

$$\Box_j\, trans.A!j!\left\{\left|[A.j.k_0]_{sk(A)}\right|\right\}^{\mathsf{a}}_{pk(j)} \rightarrow rec.A.j?\{|s|\}^{\mathsf{s}}_{k_0} \rightarrow STOP$$

$$RESP_B(s_0) =$$

$$rec.B.A?\left\{\left|[A.B.k]_{sk(A)}\right|\right\}^{\mathsf{a}}_{pk(B)} \rightarrow trans.B!A!\{|s_0|\}^{\mathsf{s}}_{k} \rightarrow STOP$$

The secret sent by B is s_0, so any suitable rank function will necessarily assign s_0 a rank of 0. Observe that here we must assume that $s_0 \notin IK$, though interestingly this assumption was not necessary for the authentication properties.

The following rank function meets all the conditions of the rank function theorem:

$$\rho(i) = 1$$

$$\rho(s) = \begin{cases} 0 \text{ if } s = s_0 \\ 1 \text{ otherwise} \end{cases}$$

$$\rho(k) = \begin{cases} 0 \text{ if } k = k_0 \\ 1 \text{ otherwise} \end{cases}$$

$$\rho(m_1.m_2) = \min\{\rho(m_1).\rho(m_2)\}$$

$$\rho(\{|m|\}^{\mathsf{s}}_k) = \begin{cases} 1 & \text{if } k = k_0 \wedge s = s_0 \\ \rho(m) & \text{otherwise} \end{cases}$$

$$\rho(\{|m|\}^{\mathsf{a}}_{pk(i)}) = \begin{cases} 1 & \text{if } i = B \wedge m = [A.B.k_0]_{sk(A)} \\ \rho(m) & \text{otherwise} \end{cases}$$

$$\rho([m]_{sk(i)}) = \begin{cases} 0 & \text{if } i = A \wedge m = A.B.k \\ \rho(m) & \text{otherwise} \end{cases}$$

The clause for $\rho([m]_{sk(i)})$ captures the fact that the enemy cannot obtain any message of the form $[A.B.k]_{sk(A)}$. This is the key to how the protocol provides the secrecy property: that B can be assured that any such signed message must indeed have been generated by A, and hence that the key k is not known to the attacker.

4.2. The simplified version

For the simplified version of the protocol, the natural change to make to the rank function is to simplify the messages in the definition of ρ, resulting in the following alternative clauses (the other clauses are unaffected):

$$\rho(\{|m|\}^{\mathsf{a}}_{pk(i)}) = \begin{cases} 1 & \text{if } i = B \wedge m = [k_0]_{sk(A)} \\ \rho(m) & \text{otherwise} \end{cases}$$

$$\rho([m]_{sk(i)}) = \begin{cases} 0 & \text{if } i = A \wedge m = k(\in KEY) \\ \rho(m) & \text{otherwise} \end{cases}$$

However, we now find that condition 4 for a rank function no longer holds: $INIT_A$ can immediately transmit a message of rank 0: the message $\left\{\left|[k_0]_{sk(A)}\right|\right\}^{\mathsf{a}}_{pk(E)}$. In this case this leads us to the following attack:

$$\begin{aligned} \alpha: &\quad A \to E : \left\{\left|[k_0]_{sk(A)}\right|\right\}^{\mathsf{a}}_{pk(E)} \\ \beta: &\quad E(A) \to B : \left\{\left|[k_0]_{sk(A)}\right|\right\}^{\mathsf{a}}_{pk(B)} \\ \beta: &\quad B \to E(A) : \{|s_0|\}^{\mathsf{s}}_{k_0} \end{aligned}$$

The responder relies on the contents of the first message (i.e. the session key) being secret. However, in the simplified case it might have gone to another party before reaching the responder, hence the protocol is flawed with respect to responder secrecy. In the original version, the inclusion of the identifiers A and B are sufficient for B to know that A encrypted the first message with B's public key, ensuring secrecy of the session key and hence the payload.

5. Multiple runs

In general, several concurrent overlapping runs of the protocol might take place, and protocol participants might be involved in more than one protocol run, possibly in different roles.

The general behaviour of such a protocol participant can be described within CSP, as an interleaving of initiator and responder runs, each with an arbitrary protocol partner. A general initiator run and a general responder run are first defined, and then a user is constructed from collections of these. Fresh messages required for the runs are modelled by requiring that each run uses a different such message, and different agents all use different messages.

A general initiator run for user C with a fresh key k chooses a partner j and runs the protocol:

$$INIT_C(k) =$$
$$\Box_j\, trans.C.j.\left\{\left|[C.j.k]_{sk(C)}\right|\right\}^{\mathsf{a}}_{pk(j)} \rightarrow rec.C.j?\{|s|\}^{\mathsf{s}}_{k} \rightarrow initdone.C.j.s.k \rightarrow STOP$$

A general responder run for user C with a fresh secret s is ready to engage in the protocol: it awaits contact from an initiator i and then follows the protocol with i:

$$RESP_C(s) =$$
$$rec.C?i?\left\{\left|[i.C.k]_{sk(i)}\right|\right\}^{\mathsf{a}}_{pk(C} \rightarrow respgo.C.i.s.k \rightarrow trans.C!i!\{|s|\}^{\mathsf{s}}_{k} \rightarrow STOP$$

A general participant C can then engage in arbitrarily many protocol runs concurrently as sender and receiver. This is captured as the interleaving of initiator and responder runs:

$$USER_C = (|||_{k\in KEY_C}\, INIT_C(k)) \;|||\; (|||_{s\in MSG_C}\, RESP(s))$$

Observe that in this description each initiator run has a different key k, and each responder run has a different message s. Each agent C has its own set of fresh keys KEY_C and messages MSG_C, and in the model these will be pairwise disjoint so any fresh key or message is associated with at most one agent, modelling the expectation that the probability of key or message overlap is negligible.

As an example of how the general case can be established, we will consider the property $respgo.B.A.s_{AB}.k_{AB}$ **precedes** $initdone.A.B.s_{AB}.k_{AB}$: that A authenticates B. In fact we can use the same rank function, given in Figure 3 as we used in the case of a single protocol run. The composition rules of Figure 5 allow the proof obligations on $USER_C$ to be reduced to individual runs. These rules follow from the trace semantics of general choice and general interleaving [Ros97,Sch99].

Checking that $USER_C \,|[\, respgo.B.A.s_{AB}.k_{AB} \,]|\, STOP$ **sat** maintains ρ then reduces (by rule INTERLEAVING of Figure 5) to checking the following:

- that each $INIT_C(k) \,|[\, respgo.B.A.s_{AB}.k_{AB} \,]|\, STOP$ **sat** maintains ρ;
- that each $RESP_C(s) \,|[\, respgo.B.A.s_{AB}.k_{AB} \,]|\, STOP$ **sat** maintains ρ.

$$\textsc{Interleaving}\quad \frac{\forall i.(P_i \textbf{ sat maintains } \rho)}{|||_i P_i \textbf{ sat maintains } \rho}$$

$$\textsc{Choice}\quad \frac{\forall i.(P_i \textbf{ sat maintains } \rho)}{\Box_i P_i \textbf{ sat maintains } \rho}$$

Figure 5. Composition rules for maintains ρ

These each reduce to consideration of the possible cases. We will work through $INIT_C(k)$ as an example.

$INIT_C(k)$ **sat** maintains ρ as long as the initiating message has positive rank. Thus by rule CHOICE we must show that $\left\{\left|[C.j.k]_{sk(C)}\right|\right\}^{a}_{pk(j)}$ has positive rank, for any C, j, and $k \in KEY_C$.

- **Case 1**: If $j = B$ and $C = A$ and $k = k_{AB}$ then $\rho(\left\{\left|[C.j.k]_{sk(C)}\right|\right\}^{a}_{pk(j)}) = 1$ from the definition of ρ.
- **Case 2**: Otherwise $\rho(\left\{\left|[C.j.k]_{sk(C)}\right|\right\}^{a}_{pk(j)}) = \rho([C.j.k]_{sk(C)})$
- **Subcase 2.1**: If $j = B$ and $k = k_{AB}$, then $C = A$. This follows from the fact that $k_{AB} \in KEY_C$ for some unique C, but for the particular key k_{AB} we know that $k_{AB} \in KEY_A$. Hence $j = B$, $k = k_{AB}$, and $C = A$. But this is case 1, so Subcase 2.1 is impossible.
- **Subcase 2.2**: $j \neq B$ or $k \neq k_{AB}$. Then $\rho([C.j.k]_{sk(C)}) = \rho(C.j.k)$. If $k = k_{AB}$ then $C = A$, and we also have from the model that k_{AB} is the key used in a session A initiates with B, thus we have $j = B$, contradicting the condition for the case. Otherwise $k \neq k_{AB}$, so $\rho(k) = 1$. Then $\rho(C.j.k) = 1$, so $\rho(\left\{\left|[C.j.k]_{sk(C)}\right|\right\}^{a}_{pk(j)}) = 1$ as required.

In all cases therefore we have that $\rho(\left\{\left|[C.j.k]_{sk(C)}\right|\right\}^{a}_{pk(j)}) = 1$, establishing that $INIT_C(k)$ **sat** maintains ρ.

A similar consideration of the cases in the responder definition establishes that each $RESP_C(s) \,||\, [\, respgo.B.A.k_{AB}.s_{AB} \,] \,||\, STOP$ **sat** maintains ρ.

Combining all these results yields that $USER_C \,||\, [\, respgo.B.A.k_{AB}.s_{AB} \,] \,||\, STOP$ **sat** maintains ρ for all users C, establishing condition 4 of the rank function theorem.

In this way we can prove that the protocol does allow the initiator to authenticate the responder in the fully general case allowing any number of concurrent protocol runs between any participants.

6. Extensions

This chapter has introduced the approach of using rank functions to the analysis and verification of security protocols. We have shown how protocols can be instrumented with signals to allow various flavours of authentication properties to be expressed (more detailed discussion of the flavours of authentication can be found in [Low97,Sch98b,

SBS09]), and also shown how secrecy can be specified. The rank function approach was first presented in [Sch97,Sch98b], and expounded at greater length in [RSG+00]. An introduction to the approach also appeared in [SD04] as an application area of CSP. The basic approach has been extended in a number of ways, both in terms of extending the theory and in terms of developing tool support.

6.1. Timestamps

An approach to handling *timestamps* was presented in [ES00,Eva03]. Timestamps are another common mechanism used within security protocols to provide assurances of freshness and prevent replay attacks. Their handling requires the modelling of the passage of time, the protocol parties' awareness of the correct time and ability to make decisions, and the fact that some delays between message creation and message receipt must be allowed for. An authentication protocol will aim to establish that if a timestamped message is received at a particular time then it must have been generated within some previous time window.

6.2. Other properties

A rank function approach was developed to handle *non-repudiation* protocols [Sch98a] in which parties each seek evidence from the other that the protocol has taken place, to prevent the other party from repudiating the transaction at a later date. In such cases, each party in the protocol is untrusted by the other, and is effectively modelled as the enemy. The aim is to collect sufficient evidence to convince a third party that the other protocol party must have participated—essentially that the evidence produced can only follow some activity by the other protocol party, in much the same way as an authentication property.

The approach has also been extended to handle various forms of *forward secrecy*. Forward secrecy can be taken to mean that the payload of the protocol is secret even if some secret elements of the protocol, such as a session key, become known to the attacker at a later stage. In this case, the classical rank function considers either that the enemy will never obtain the message, or that the enemy might as well have it from the beginning. However, this approach is not appropriate for temporary secrets such as session keys. Instead, in [Del06,DS07] the notion of a rank function is generalised to a *temporal rank function* so that ranks range across positive integers (together with infinity), which may be thought of as corresponding to the time at which a message might be available to the enemy. This allows analysis of protocols which rely on the secrecy of some information at a particular point in time. A generalised version of the rank function theorem is able to establish long-term secrecy of messages in these cases.

In the context of group protocols, concern can also focus on whether secrets established by honest members of a group can be exposed at some other stage if an enemy joins the group. The rank function approach has been applied in this context [GT07,Gaw08] for both forward secrecy (secret keys cannot be obtained from later runs) and backward secrecy (secret keys cannot be obtained from earlier runs).

6.3. Algebraic properties of cryptographic primitives

Some cryptographic schemes have particular properties (for example, commutativity of encryption) useful for constructing protocol schemes, but which might allow other possibilities of attack. The rank function approach extends to handle these cases, where the properties can be captured as equations on messages, or as further message derivation clauses (in the 'generates' relation). In one example, an analysis of Gong's protocol built around exclusive-or [Gon89] was presented in [Sch02]. Exclusive-or has several properties, such as commutativity, self-inverse of encryption keys, and cancellation properties. The analysis modelled these as equations on the algebra of messages, and the additional requirement on a rank function is that it must be well-defined in the context of the equations: if two (differently constructed) messages are equal, then they should have the same rank. Since rank functions tend to be defined by induction over the BNF for constructing messages, establishing well-definedness is an additional requirement. This approach was also used in [DS07] for a class of group Diffie-Hellman authenticated key-agreement protocols: keys can be constructed using exponentiation in a number of different ways, and it is important that all constructions of the same key have the same rank.

6.4. Tool support

Various forms of tool support have been developed for the rank function approach, in some cases with underlying theory to underpin the approach. A theory of rank functions on top of CSP was developed in the theorem-prover PVS [DS97]. This theory allowed definitions of rank functions, CSP descriptions of protocol participants, and verification of the conditions of the rank function theorem. Since much of the work in carrying out such a proof is mechanical house-keeping the provision of tool support is natural. The PVS theories for CSP and for rank functions were refactored and extended (to handle time) in [Eva03,ES05]. PVS has also been used to implement inference systems based on rank functions to check whether attacks are possible [GBT09]. In this approach, various properties of a rank function are given, and the inference system is used to establish whether an attack is possible from the protocol rules.

As an alternative to theorem-proving, an approach for automatically generating a rank function for a given protocol was developed in [Hea00,HS00]. This approach constructs a minimal function whose positive messages include those of the enemy's initial knowledge, are closed under the message generation rules, and are closed under the protocol agents' behaviour for outputting. If the resulting function also gives a rank of 0 to the authenticating message, then it meets all the conditions of the rank function theorem, and the protocol is verified. Conversely if the resulting function gives a positive rank, then there can be no rank function that will meet all the conditions of the rank function theorem.

Acknowledgements

I am grateful to Roberto Delicata for comments on this chapter.

References

[Del06] R. Delicata. *Reasoning about Secrecy in the Rank Function framework*. PhD thesis, University of Surrey, 2006.

[DS97] B. Dutertre and S.A. Schneider. Embedding CSP in PVS: an application to authentication protocols. In *tpHOL*, 1997.

[DS07] Rob Delicata and Steve Schneider. An algebraic approach to the verification of a class of diffie-hellman protocols. *Int. J. Inf. Sec.*, 6(2-3):183–196, 2007.

[DY83] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.

[ES00] N. Evans and S.A. Schneider. Analysing time dependent security properties in CSP using PVS. In *ESORICS*, volume 1895 of *LNCS*, 2000.

[ES05] N. Evans and S.A. Schneider. Verifying security protocols with PVS: Widening the rank function approach. *Journal of Logic and Algebraic Programming*, 2005.

[Eva03] N. Evans. *Investigating Security Through proof*. PhD thesis, Royal Holloway, University of London, 2003.

[For03] Formal Systems (Europe) Ltd. FDR2 user manual, 2003.

[Gaw08] A. Gawanmeh. *On the formal verification of group key security protocols*. PhD thesis, Concordia University, Canada, 2008.

[GBT09] A. Gawanmeh, A. Bouhoula, and S. Tahar. Rank functions based inference system for group key management protocols verification. *International Journal of Network Security*, 8(2), 2009.

[Gon89] L. Gong. Using one-way functions for authentication. *Computer Communications Review*, 19(5), 1989.

[GT07] A. Gawanmeh and S. Tahar. Rank theorems for forward secrecy in group key management protocols. In *IEEE International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, 2007.

[Hea00] J.A. Heather. *"Oh! Is it* really *you?"—Using rank functions to verify authentication protocols*. PhD thesis, Royal Holloway, University of London, 2000.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HS00] James Heather and Steve Schneider. Towards automatic verification of authentication protocols on an unbounded network. In *CSFW*, pages 132–143, 2000.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *CSFW*, 1997.

[PQ00] O. Pereira and J-J. Quisquater. On the perfect encryption assumption. In *WITS '00: Workshop on Issues in the Theory of Security*, 2000.

[Ros97] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[RSG+00] P.Y.A. Ryan, S.A. Schneider, M.H. Goldsmith, G. Lowe, and A.W. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2000.

[SBS09] S. Shaikh, V. Bush, and S. Schneider. Specifying authentication using signal events in CSP. *Computers & Security*, 28(5), 2009.

[Sch97] Steve Schneider. Verifying authentication protocols with CSP. In *CSFW*, pages 3–17, 1997.

[Sch98a] S.A. Schneider. Formal analysis of a non-repudiation protocol. In *11th IEEE Computer Security Foundations Workshop*, 1998.

[Sch98b] Steve Schneider. Verifying authentication protocols in CSP. *IEEE Trans. Software Eng.*, 24(9):741–758, 1998.

[Sch99] S.A. Schneider. *Concurrent and Real-time Systems: the CSP Approach*. Addison-Wesley, 1999.

[Sch02] S.A. Schneider. Verifying security protocol implementations. In *FMOODS'02: Formal Methods for Open Object-based Distributed Systems*, 2002.

[SD04] S. Schneider and R. Delicata. Verifying security protocols: An application of CSP. In *Communicating Sequential Processes: the First 25 Years*, pages 243–263, 2004.