VERIFYING AUTHENTICATION PROTOCOL IMPLEMENTATIONS*

Steve Schneider
Department of Computer Science,
Royal Holloway, University of London
S.Schneider@cs.rhul.ac.uk

Abstract Formal methods for verifying authentication protocols tend to assume an idealised, perfect form of encryption. This approach has been spectacularly successful in finding flaws, but when we aim for proofs of correctness then we need to consider this assumption more carefully, and perhaps to weaken it to reflect properties of real cryptographic mechanisms. This paper reviews the existing CSP approach to verifying protocols, and considers how algebraic properties of real cryptographic mechanisms can be incorporated within a rank function verification. The approach is illustrated with an authentication protocol which makes use of exclusive-or.

1. Introduction

Security protocols aim to provide security guarantees between parties communicating over an insecure network, and possibly in the presence of malicious agents who can interfere with and disrupt communication traffic. One important security requirement is authentication: that agents have guarantees about the identity of the agent they are communicating with. Other desirable security properties include confidentiality, non-repudiation, and anonymity. For reasons of space, authentication is the only property that this paper will focus on.

Security protocols, first proposed in Needham and Schroeder, 1978, make use of cryptography in order to achieve their aims. Although malicious agents between communicating parties can block, redirect, and spoof messages, they cannot decrypt messages, or produce encrypted messages, unless they have the corresponding key, and so this provides

*Presented at FMOODS 2002

some measure of protection in a hostile environment. It is possible to make use of knowledge of ownership of keys to establish guarantees about the originators of corresponding messages. Authentication protocols thus describe message exchanges which aim to provide the appropriate guarantees to the parties involved.

However, in practice such protocols are difficult to design correctly, because it is extremely difficult to forsee all the ways potential attackers might be able to subvert them. This is demonstrated by the fact that a number of published protocols have since been shown to contain flaws. In the past decade, there has been an explosion of activity in applying formal methods to the analysis of security protocols, from two directions. Firstly, applying model-checking technology to searching for attacks (see e.g. Germeau and Leduc, 1997; Kemmerer, 1989; Kemmerer et al., 1994; Lowe, 1995; Lowe, 1996; Marrero et al., 1997; Meadows, 1992; Millen, 1995; Roscoe, 1995); and secondly, in establishing correctness with respect to particular properties, and with regard to certain assumptions about their environment (see e.g. Schneider, 1996; Paulson, 1998; Fábrega et al., 1998; Abadi and Gordon, 1998; Gordon and Jeffrey, 2001). Such analyses generally assume perfect encryption: that the decryption key must be known in order to extract the encrypted text, and that a ciphertext can only be generated using encryption with the appropriate key and message. Any real mechanism, such as RSA, exhibits additional algebraic properties which can introduce new vulnerabilities, even into protocols whose abstract design is proved secure. Although this has been known for some considerable time (see e.g. Even et al., 1985), more recent formal analysis methods for protocols have tended to neglect this implementation aspect. This means that even formally verified protocols might have flawed implementations: an example is given in Ryan and Schneider, 1998, where the use of exclusive-or as an encryption mechanism introduced a flaw into a previously verified protocol.

This paper reviews the author's use of CSP to model and verify protocols, and considers how algebraic properties of real cryptographic mechanisms can be incorporated within this verification framework. The approach is illustrated with an authentication protocol which makes use of exclusive-or.

2. CSP notation

Process algebras provide a particular approach to the study of concurrency and interaction. This paper uses the process algebra CSP (Communicating Sequential Processes). A full account of this process algebra

can be found in Roscoe, 1997 or Schneider, 1999. It provides a language for describing interacting systems, together with a semantic theory for understanding them. This section provides a brief reminder of those aspects most relevant to this paper.

The language of CSP is constructed around *events*: instantaneous synchronisations which provide the communication primitive. Events may have some structure, the most common being a channel communication of the form $c.v$, where c is the channel name, and v is the value communicated. Channel names and values can themselves have structure. For example, $rec.A$ can be a channel passing values of the form $j.m$. The event corresponding to the transmission of such a message would then be $rec.A.j.m$.

Processes are used to describe possible patterns of interaction. The process $c!v \rightarrow P$ describes a process which is prepared to output v on channel c, and behave subsequently as P. The input $c?x \rightarrow P(x)$ may take in some value v along channel c and behave subsequently as $P(v)$. The process $STOP$ is not prepared to perform any events at all, and is generally used in this paper to denote the end of a process' execution pattern. A general choice between a family of processes $\{P_i \mid i \in I\}$ is given as $\Box_{i \in I} P_i$. Condition statements using if then else, and let statements, are also possible within the language.

Processes may be put in parallel: $P \underset{A}{\parallel} Q$ behaves as P running concurrently with Q, synchronising on events in A, and performing other events independently. Values are passed between parallel processes by means of synchronisations on the channels, linking an output channel of one to an input channel of another. An interleaving of two processes, $P \mid\mid\mid Q$, simply executes P and Q concurrently, independently of each other without any communication between them. An interleaving of a family of processes $\{P_i \mid i \in I\}$ is written $|||_{i \in I} P_i$. Each of the P_i executes independently of and concurrently with all the others.

Processes may also be recursively defined, by giving equations which contain the names of the processes being defined as subterms of the process expressions. For example, a two place buffer $2COPY$ may be defined using the following family of equations:

$$\begin{aligned} 2COPY &= in?x \rightarrow HOLDS(x) \\ HOLDS(x) &= in?y \rightarrow out!x \rightarrow HOLDS(y) \\ &\quad \Box\ out!x \rightarrow 2COPY \end{aligned}$$

The semantics of processes are given in terms of observations. A process is identified with the set of behaviours that may possibly be observed of it, where the kind of behaviour considered determines the

nature of the model. In this paper we are concerned only with the *Traces Model* is concerned with the traces of a process: the (finite) sequences of events that it can perform during some execution. For example,

$\langle in.3, out.3, in.5, in.8, out.5 \rangle$

is a trace of $2COPY$. The set of all traces of a process P is denoted $traces(P)$.

Safety specifications are given as predicates on traces, and a process P satisfies a specification $S(tr)$ if all of its traces satisfy $S(tr)$:

$$P \text{ sat } S(tr) \quad \Leftrightarrow \quad \forall\, tr \in traces(P) \bullet S(tr)$$

For example, to specify the requirement that some event from the set A should precede any event from B, we can define a predicate A precedes B which means that if some $b \in B$ appears in the trace tr then some $a \in A$ must have appeared in the trace previously. This can be defined as follows:

$$(b \in B \wedge tr = tr_1 \frown \langle b \rangle \frown tr_2) \Rightarrow \exists\, a \in A.a \text{ in } tr_1$$

where $\frown$ is sequence concatenation, and a in tr_1 means that a appears at some point within the trace tr_1. The authentication properties considered in this paper will be expressed in this way, in terms of the precedes predicate.

3. Verifying security protocols with CSP

Security protocols are generally described in terms of the sequence of messages exchanged by the protocol participants. Messages are often encrypted using either shared-key or public-key cryptography, and generally include special information such as agent names, secrets shared between some of the agents, nonces (newly generated messages created specifically for a particular protocol run), or newly generated keys to be distributed between the participants.

3.1. Gong's protocol

This paper will consider a protocol proposed in Gong, 1989 as an illustration of the approach. The protocol is described in Figure 1. It makes use of two one-way functions f and g. We assume that such one-way functions are collision-free, and that it is computationally infeasible to extract the value of m from $f(m)$ or $g(m)$. The outputs of the one-way functions used in this protocol are treated as triples (k, ha, hb). The protocol also makes use of the exclusive-or operator $\oplus$ to combine

1. $A \longrightarrow B \quad A, B, na$
2. $B \longrightarrow S \quad A, B, na, nb$
3. $S \longrightarrow B \quad ns\ ,\ f(ns, nb, A, P(B)) \oplus (k, ha, hb)\ ,\ g(k, ha, hb, P(B))$
4. $B \longrightarrow A \quad ns, hb$
5. $A \longrightarrow B \quad ha$

Figure 1. Gong's authentication protocol

messages. The protocol aims to provide authentication of A to B, and authentication of B to A. It makes use of a trusted server S, with which each of A and B shares a secret, $P(A)$ and $P(B)$ respectively.

- In message 1, A generates a nonce na and sends it to B, along with the two agent names.
- In message 2, B generates another nonce nb, and forwards both na and nb to S along with the two agent names. On receipt of some message 2, S generates a nonce ns and uses the one-way function f on $ns, na, B, P(A)$ to produce $f(ns, na, B, P(A)) = (k, ha, hb)$, which is xor'd with $f(ns, nb, A, P(B))$ for message 3. $g(k, ha, hb, P(B))$ is also computed to make up message 3.
- On receipt of message 3, B is able to compute $f(ns, nb, A, P(B))$ because it receives ns from the server, and already has nb, A, and $P(B)$. (No agent or enemy can compute this without $P(B)$.) Thus it retrieves (k, ha, hb) from the second item, and hence computes $g(k, ha, hb, P(B))$ to compare with the third component of message 3. If they match then B accepts k as having been freshly generated by S.
- Message 4 consists of the nonce ns and the value hb. On receipt of this message, A computes $f(ns, na, B, P(A))$ using the value of ns received, to obtain (k, ha, hb), and check whether hb matches the one received. If it does, then A has authenticated B, since extraction of hb requires the secret $P(B)$.
- Message 5 consists of A responding with ha. On receipt of this message, B has authenticated A, since extraction of ha requires the secret $P(A)$.

The intention is that at the end of the protocol, A and B have each authenticated the other, and they each have the fresh secret key k.

m	::=	t	plaintext
		i	agent identity
		n	nonce
		k	key
		$P(i)$	shared secret
		f	one-way function
		$\pi_i(m)$	projection of message ($i = 1, 2, 3$)
		$m_1.m_2$	message concatenation
		$m_1 \oplus m_2$	exclusive-or of two messages

Figure 2. Messages used in Gong's protocol

3.2. CSP description

A CSP description of a protocol such as that given here describes the behaviour of each agent involved in a protocol run. Whereas the original protocol description of Gong, 1989 is given in terms of a correct run of the protocol, a process algebraic approach (in common with many of the approaches mentioned in the introduction) describes the protocol from the point of view of each of the participants. These are described in terms of the messages that they can send and receive.

In order to enable a formal analysis, we first define formally the space of messages *MESSAGE* that can be sent and received by protocol agents. This is defined in Figure 2. In this protocol, we allow agent names, nonces, keys, shared secrets, as well as concatenating messages, combining them via exclusive-or, applying the one-way functions f and g, and extracting parts of messages obtained by applying f and g by use of the projection functions π_i. Other protocols make use of other kinds of message component, such as encrypted messages, or timestamps; and the analysis of such protocols will require the inclusion of such components in the definition of the message space.

The protocol is captured within CSP by first describing a single run for each role of the protocol. We use the channel $trans.i$ to carry messages m that are transmitted as output to j by an agent i. Occurrence of such communications will be represented by events of the form $trans.i.j.m$. Similarly, the channel $rec.i$ will carry those messages received as input to an agent i.

For example, the role played by A in the description of Figure 1 is that of the protocol *initiator*, who starts off a run of the protocol. This is done by sending out a nonce as message 1, and then waiting for a response

$$
\begin{aligned}
INIT(i, ni) \;=\; & \Box_j \, trans.i.j.(i.j.ni) \\
& \rightarrow rec.i.j?(n.m) \\
& \rightarrow \begin{cases} trans.i.j.\pi_2(f(n.ni.j.P(i))) \rightarrow STOP \\ \qquad\qquad\qquad\qquad \text{if } m = \pi_3(f(n.ni.j.P(i))) \\ STOP \\ \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}
\end{aligned}
$$

Figure 3. CSP description of initiating run

$$
\begin{aligned}
SERVER(ns) \;=\; & rec.s?j?(i.j.ni.nj) \\
& \rightarrow trans.s.j.(ns. \\
& \qquad\qquad (f(ns.nj.i.P(j)) \oplus f(ns.ni.j.P(i))). \\
& \qquad\qquad g(f(ns.ni.j.P(i)).P(j)) \\
& \rightarrow STOP
\end{aligned}
$$

Figure 4. CSP description of a run of the server

as message 4. Finally, the initiator sends out a further confirmation message. Since any agent can play the role of initiator with any nonce, we define the general initiator role as a process $INIT(i, ni)$ parameterised by the initiating agent's name i and initiating nonce ni. The resulting CSP description is given in Figure 3. This run begins by choosing some agent j to communicate with, and then sending the first message $i.j.ni$. It then receives a message with two components n and m, which it can use to construct $f(n.ni.j.P(i))$ (since it is already in possession of ni, j, and its secret $P(i)$) and check that the third component of the result matches the other input m. If it does, then the final message (which is the second component) is transmitted. Otherwise we model the run as simply stopping at that point.

A run of the *responder* role (taken by B in the protocol description) and of the trusted server (described by S in the protocol description) are described within CSP using the same approach. The server's involvement in a single run (using nonce ns) is given as $SERVER(ns)$ in Figure 4, and agent j playing the role of the responder with nonce nj is given as $RESP(j, nj)$ in Figure 5.

$$
\begin{array}{l}
RESP(j, nj) = \\
\quad rec.j?i?(i.j.ni) \\
\quad\quad \rightarrow trans.j.s.(i.j.ni.nj) \\
\quad\quad\quad \rightarrow rec.j.s?(n.m.l) \\
\quad\quad\quad\quad \rightarrow \left\{ \begin{array}{l} trans.j.i.(n.\pi_3(m \oplus f(n.nj.i.P(j)))) \\ \quad \rightarrow rec.j.i.\pi_2(m \oplus f(n.nj.i.P(j))) \rightarrow STOP \\ \qquad\qquad \text{if } l = g((m \oplus f(n.nj.i.P(j))).P(j)) \\ STOP \\ \qquad\qquad \text{otherwise} \end{array} \right.
\end{array}
$$

Figure 5. CSP description of a run of the responder

3.3. Protocol analysis

Security protocols are designed to provide certain guarantees in the face of a hostile environment in which the protocol can be 'attacked' in particular ways. Hence any protocol analysis must include the possibility of such attacks, in order to establish that they do not happen. In this context, the 'Dolev-Yao model' (first introduced to analyse a particular class of protocols in Dolev and Yao, 1983) has proven a very powerful abstraction. This model allows the hostile environment to be described in a very general way, in terms of the *capabilities* of potential attackers or enemies, rather than in terms of particular attacks which they can carry out (which would not guard against new and currently undiscovered kinds of attack).

In its most general form, the Dolev-Yao model considers the protocol running in an environment consisting of a single enemy who intercepts all messages, and is able to send any message that it can generate (which includes those it has intercepted, and messages it can generate from them) to any protocol agent. This also enables the enemy to block or redirect any message. The enemy is also able to play the role of a genuine protocol agent, and thus interact with some other agents on that basis. Thus the enemy has complete control over the communications medium, and in modelling this in CSP it is equivalent to treat the enemy *as* the communications medium: that the transmission of a message m from i to j involves i passing m to the enemy in the hope that the enemy will then pass m on to j. The model is pictured in Figure 6.

This extreme model is appropriate only for safety properties, since the enemy could simply block all communication and hence subvert any

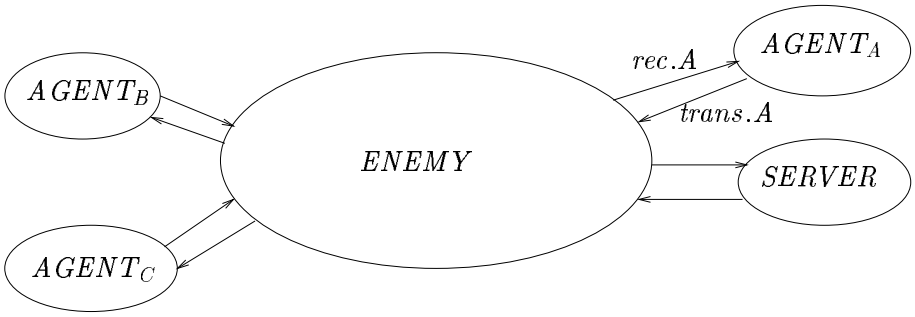

Figure 6. The CSP architecture of the Dolev-Yao model

liveness requirement. However, many security properties, including authentication, are indeed safety properties, and hence can be verified using this model.

The enemy is constrained by cryptography. For example, it is unable to encrypt or decrypt messages if it does not have the appropriate key. In this paper, the enemy is unable to extract either m_1 or m_2 if it has only the message $m_1 \oplus m_2$.

The enemy's ability to deduce or generate new messages from those it already knows about is captured by a 'generates' relation $\vdash$, which defines the precise ways in which the enemy can create messages. The relation $S \vdash m$ states that the message m can be generated from the set of messages S. There are three general closure conditions for the relation:

- $\{m\}s \vdash m$ for any m;
- If $S_2 \vdash m$ and $S_2 \subseteq S_1$ then $S_1 \vdash m$;
- If $S \vdash m'$ for every $m' \in T$, and $T \vdash m$, then $S \vdash m$

The relation $\vdash$ is then defined to be the smallest relation closed under these three conditions, and such that:

- $\{m_1, m_2\} \vdash m_1.m_2$;
- $\{m_1.m_2\} \vdash m_1$;
- $\{m_1.m_2\} \vdash m_2$;
- $\{f, m\} \vdash f(m)$;
- $\{m\} \vdash \pi_i(m)$;

- $\{m_1, m_2\} \vdash m_1 \oplus m_2$;
- $\{m_1, m_1 \oplus m_2\} \vdash m_2$;
- $\{m_2, m_1 \oplus m_2\} \vdash m_1$.

Thus messages can be concatenated and de-concatenated; they can be transformed under one-way functions; they can be projected; and they can be exclusive-or'd and extracted from exclusive-or messages. The enemy can generate messages only using these rules. Observe that there is no rule for obtaining m from $f(m)$, reflecting the defining property of one-way functions.

Even before any protocol runs occur, the enemy knows some messages: the names of the agents, the one-way functions, its own secret shared with the server (since the enemy is also able to take the role of a protocol agent), and the ability to generate some nonces. We will take the set IK to represent the set of messages that the enemy knows initially. The enemy is also able to add to the store of messages that it knows by observing messages being sent along the *trans* channel by other agents. At any stage it could also send any message (along *rec*) that it can generate to any protocol agent (though of course agents might not be expecting such messages and could simply ignore or reject them). Such an enemy can be described as the following recursive process, where S is the set of messages the enemy currently has:

$$\begin{aligned} ENEMY &= ENEMY(IK) \\ ENEMY(S) &= trans?i?j?m \rightarrow ENEMY(S \cup \{m\}) \\ &\quad \Box \\ &\quad \Box_{\{m \mid S \vdash m\}} \Box_{i,j} rec.i.j.m \rightarrow ENEMY(S) \end{aligned}$$

The system we analyse consists of the individual agents communicating with the enemy. In general, any individual agent might be able to perform a number of concurrent runs of the protocol, taking either initiator or responder role in each case. This is described as an interleaved combination of $INIT$ and $RESP$ runs:

$$AGENT_i = (|||_{ni \in IN_i} INIT(i, ni)) \;|||\; (|||_{ni \in NR_i} RESP(i, ni))$$

where NI_i is the (infinite) set of nonces i uses in initiator runs; and NR_i is the (infinite) set of nonces i uses in responder runs. These sets do not overlap, and they are disjoint with the sets of nonces used by other agents.

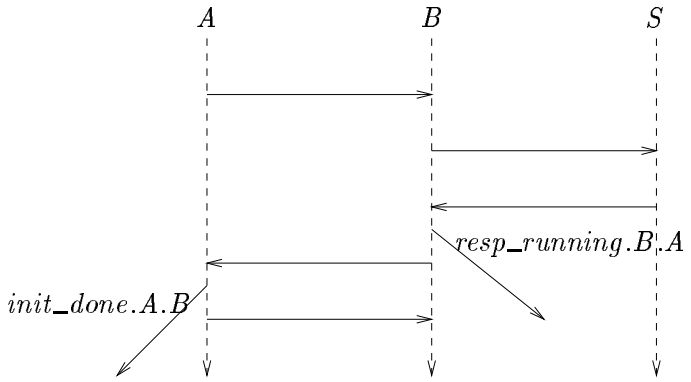

Figure 7. Signals for an authentication requirement: A authenticates B

The server is described in a similar fashion, using its own set of nonces NS, able to handle any number of runs at the same time:

$$SERVER \quad = \quad |||_{ns \in NS} \, SERVER(ns)$$

Finally, the entire system is described as all the protocol agents in parallel with the enemy:

$$SYSTEM \quad = \quad (SERVER \; ||| \; (|||_i \, AGENT_i)) \underset{\{trans, rec\}}{\|} ENEMY(IK)$$

3.4. Security properties

The aim of authentication is to obtain some assurance about the identity of the other party involved in the protocol. Considered in terms of the messages that are sent and received within a protocol run, an authentication property will use messages received by one party to provide certain guarantees about messages that must have been transmitted by the other party. In its simplest form, it can require that receipt (by A) of a particular message apparently from B to A is only possible if B did indeed send that message to A. In Gong's protocol, for example, receipt by A of message 4 apparently from B might provide a guarantee that B indeed transmitted that message, even in the presence of the enemy. However, the values ns and hb must be validated by A (and not simply received) for authentication to occur. Thus to express the authentication requirements more explicitly, it is useful to introduce additional 'signal' messages into the CSP descriptions of the protocol agents, which indicate various points that they are along their execution runs. Thus we

will introduce an additional event *resp_running.i.j* into the description of the responder run, before message 4, which indicates that the responder i has reached a point where it can claim to be running the protocol with responder j. We also introduce an event *init_done.i.j* into the initiator run after the information received in message 4 has been checked successfully. The protocol with these additional messages is pictured in Figure 7. This authentication property is captured as a property on *SYSTEM*: that any occurrence of *init_done.A.B* must be preceded by the occurrence of *resp_running.B.A*. Formally this is written as

$$SYSTEM \quad \mathsf{sat} \quad resp\_running.B.A \ \mathsf{precedes} \ init\_done.A.B$$

In fact there is a whole range of authentication properties that can be expressed in this kind of way. The information associated with the signal events can be changed to reflect the required level of matching between the two protocol runs. For example:

- *resp_running.B* precedes *init_done.A.B* indicates that A has authenticated the fact that B has been running the protocol, and hence that B is active; but not that B considers A to be the other party;
- *resp_running.B.A.k* precedes *init_done.A.B.k* states that A authenticates B, and that they agree on the value of the key k delivered in the protocol run;
- 'the number of occurrences of *init_done.A.B* should be no greater than the number of occurrences of *resp_running.B.A*' states that each authentication of B by A should correspond to a different protocol run by B, thus avoiding multiple authentications of the same responder run.

A hierarchy of CSP authentication properties is discussed in greater detail in Lowe, 1997.

3.5. Rank functions for verification

The main result of Schneider, 1998 is a theorem providing circumstances in which it can be proven that *SYSTEM* sat R precedes T, where R and T are both sets of events. This enables authentication properties written in this form to be established for authentication protocols. For example, authentication of B to A in the protocol above corresponds to the requirement $\{resp\_running.B.A\}$ precedes $\{init\_done.A.B\}$.

The theorem makes use of a *rank function* ρ which maps messages to some domain of values. This domain was the integers in Schneider,

1998, and was simplified to the set $\{0,1\}$ in Heather, 2000 and Heather and Schneider, 2000. In fact, any set $RANK$ will do, so we will present the theorem here in a generalised form, with respect to an arbitrary set of rank values $RANK$.

Theorem 1 *If the set $RANK$ is partitioned into sets $RANK^+$ and $RANK^-$, and for sets of events R and T, there is a rank function $\rho : MESSAGE \to RANK$ satisfying*

1 $\forall\, m \in IK \bullet \rho(m) \in RANK^+$

2 $\forall\, S \subseteq MESSAGE, m \in MESSAGE \bullet$
$((\forall\, m' \in S \bullet \rho(m') \in RANK^+) \wedge S \vdash m) \Rightarrow \rho(m) \in RANK^+$

3 $\forall\, t \in T \bullet \rho(t) \in RANK^-$

4 $\forall\, J \bullet USER_j \underset{R}{\|} STOP$ sat preserves $RANK^+$

then $SYSTEM$ sat R precedes T.

The theorem reduces a requirement on the overall system to separate requirements on the individual components of the system. The conditions correspond to the following requirements:

1 the enemy starts off only with messages with ranks in $RANK^+$;

2 the enemy can only generate messages with ranks in $RANK^+$ unless it already has some message with rank not in $RANK^+$;

3 no event in T has rank in $RANK^+$;

4 for every user when blocked on events from R: if it only ever inputs messages with ranks in $RANK^+$, it will only output messages with ranks in $RANK^+$. This is formalised in the predicate preserves $RANK^+$ on traces.

Conditions 1, 2, and 4 together ensure that only messages with rank within $RANK^+$ can ever be passed in the system blocked on R. Condition 3 allows the conclusion that no event from T can therefore occur.

Hence the problem of verifying a protocol against a certain authentication property reduces to the problem of finding a rank function which meets all these conditions. This has been done for Gong's protocol (see Evans and Schneider, 2001a), and Figure 9 provides such a rank function. That rank function appears after the next section because it has been constructed to take into account the considerations of the next section, concerning the algebraic properties of exclusive-or.

4. Weakening the perfect encryption assumption

Most formal protocol analysis to date has incorporated the perfect encryption assumptions (discussed in Pereira and Quisquater, 2000): that

1 the decryption key must be known in order to extract the plaintext corresponding to a given ciphertext;

2 there is enough redundancy in the crypto system that a ciphertext can only be generated using encryption with the appropriate key and message.

These assumptions have been required for tractability of the analysis, and protocols shown to be flawed even under these powerful assumptions will remain flawed in implementation. The way we have modelled the enemy using the $\vdash$ relation above incorporates the perfect encryption assumption, since that assumption is encapsulated by those rules, and the enemy can generate and obtain messages *only* by using these rules.

However, although they are a powerful abstraction, the assumptions are obviously not true in practice. Common encryption mechanisms which violate them include

- RSA (Rivest et al., 1978), whose multiplicative structure means that $\{m_1\}_k\{m_2\}_k = \{m_1.m_2\}_k$ (of course, 'multiplication' of messages is a different kind of operation not mentioned in the definition of messages). Also, RSA encryption is commutative: $\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$
- Vernam encryption: exclusive-or of a key and a text to produce the ciphertext. Exclusive-or has many algebraic properties which might be exploited to attack a protocol. For example, possession of m and $\{m\}_k$ allow the extraction of k, which is not normal for encryption systems.

In order to allow verification of protocol implementations which make use of particular cryptographic mechanisms, the properties of the mechanisms should be incorporated into the analysis.

One way of doing this is by extending the $\vdash$ relation to allow additional clauses, corresponding to further ways in which messages can be generated. For example, to capture the first property of RSA above, we would add the following clause:

$$\{\{m_1\}_k, \{m_2\}_k\} \vdash \{m_1.m_2\}_k$$

The rank function Theorem 1 remains applicable, and the only change is that the set $RANK^+$ must be preserved even under these additional

clauses. In other words, clause (2) needs to be checked under these additional conditions.

However, if there are algebraic identities introduced by the implementation of encryption, then it is more appropriate to introduce such equations onto the message space. The rank function theorem again remains applicable, but some additional checking is required on rank function definitions, since rank functions tend in practice to be defined inductively over the message space. It will be necessary to ensure that any such rank function is well-defined: that different constructions of the same message do not give rise to different rank values.

In the case of Vernam encryption, a message m is encrypted with a key k by taking the exclusive-or $\oplus$ of m and k: $m \oplus k$. We will assume that the arguments to an exclusive-or are the same length.

$$\{m\}_k \quad = \quad m \oplus k$$

The set of messages of length n under $\oplus$ forms an abelian group in which every element is its own inverse. The identity of this group will be the sequence of n 0's. The identities of $\oplus$ were not mentioned explicitly in the space of messages, but we will treat them as plaintext, covered by the clause t.

Furthermore, $\oplus$ interacts with concatenation as follows: if m_1 and m_1' have the same length, and m_2 and m_2' have the same length, then

$$(m_1.m_2) \oplus (m_1'.m_2') \quad = \quad (m_1 \oplus m_1').(m_2 \oplus m_2') \tag{1}$$

Unfortunately, this equation is not compatible with the original analysis of the Gong protocol given in Evans and Schneider, 2001a (which was not concerned with this identity). In that paper, the rank function included the definitions:

$$\begin{aligned} \rho(m_1.m_2) &= min(\rho(m_1), \rho(m_2)) \\ \rho(m_1 \oplus m_2) &= \begin{cases} 1 & \text{if } \rho(m_1) = \rho(m_2) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

The first clause is completely standard in rank function analysis: it captures the fact that if either m_1 or m_2 cannot circulate in the system, then neither can $m_1.m_2$. The second clause allows $m_1 \oplus m_2$ to circulate even if m_1 and m_2 are both secret. This is necessary to reflect the fact that encrypted messages are sent over the network.

However, this rank function is not well-defined in the presence of Equation 1. If $\rho(m_1) = \rho(m_2') = 1$ and $\rho(m_1') = \rho(m_2) = 0$, then by applying the definitions we find that

$$\begin{aligned} \rho((m_1.m_2) \oplus (m_1'.m_2')) &= 1 \\ \rho((m_1 \oplus m_1').(m_2 \oplus m_2')) &= 0 \end{aligned}$$

Yet these are two constructions of the same message, and so the rank function should give the same result for each construction.

A closer inspection reveals that although $m_1.m_2$ should have rank 0 because of m_2, and $m_1'.m_2'$ should have rank 0 because of m_1', the ranks of the components of the messages do not match and hence should have rank 0 when xor'd together.

On the other hand, the different message $m_1.m_2 \oplus m_2'.m_1'$ should have rank 1, since the ranks match for each component of the two messages.

Thus in the presence of Equation 1 it is necessary to maintain a rank value for each component in a concatenated message, and not simply one value overall. Hence it is most straightforward to use a (finite) *sequence* of values (of 0 or 1) as a rank value. The set $RANK^+$ will be those sequences consisting entirely of 1's.

$$\begin{aligned} RANK &= \{0,1\}^* \\ RANK^+ &= \{1\}^* \\ RANK^- &= RANK - RANK^+ \end{aligned}$$

For reasons of space, we consider only authentication of responder B by initiator A, for two arbitrary users A and B, with A making use of a nonce N_A.

Authentication of B to A is achieved by $INIT(A)$ receiving the fourth message $n.m$ of the protocol. As illustrated in Figure 7 earlier, this is made explicit by the introduction of the signal event $init\_done.A.B$ event, which signals the point at which A is deemed to have successfully authenticated B. This event can occur only if $m = hb$, where $f(n.ni.j.P(i)) = (k, ha, hb)$. Authentication can be shown by analysing the system resulting from all occurrences of $resp\_running.A.B$ being blocked: if it is impossible for A to perform that event in the resulting restricted system, then we have shown that A can only perform its signal event if B has previously performed its own.

Previous results (see Ryan et al., 2000 and Heather and Schneider, 2000) mean that we need only provide a rank function which is preserved by $INIT(A, N_A)$, by any server run $SERVER(N_S)$, and by any responder run $RESP(B, N_B)$ in which $resp\_running.A.B.N_A$ is blocked, as shown in Figure 8.

The use of sequences of values for $RANK$ allows an appropriate rank function to be constructed in a straightforward fashion. Figure 9 gives such a function, which meets all of the conditions of Theorem 1 and is well-defined in the presence of the additional algebraic properties of exclusive-or discussed above. Rather than make use of the projection functions explicitly within the definition of messages, we are now taking account of various equations on messages. Thus we can make explicit

$$
\begin{array}{l}
RESP(B, N_B) = \\
\quad rec.B?i?(i.B.ni) \\
\quad\quad \rightarrow trans.B.s.(i.B.ni.N_B) \\
\quad\quad\quad \rightarrow rec.B.s.(n.m.l) \\
\quad\quad\quad\quad \rightarrow \left\{ \begin{array}{ll}
resp\_running.i.B & \text{if } i \neq A \\
\rightarrow \mathsf{let}(k, hc, hd) = m \oplus f(n.N_B.i.P(B)) \ \mathsf{in} & \\
\quad \left\{ \begin{array}{ll}
trans.B.i.(n.hd) \rightarrow & \\
rec.B.i.hc \rightarrow STOP & \text{if } l = g(k.hc.hd.P(B)) \\
STOP & \text{otherwise}
\end{array} \right. & \\
STOP & \text{if } i = A
\end{array} \right.
\end{array}
$$

Figure 8. Responder with $resp\_running.A.B$ blocked

that the result of applying the function f is a message of the form $k.h_1.h_2$, where h has now been introduced as a new kind of message representing the second and third components of the range of f. We also require that any h appears in the image of no more than one message, so $f(m)$ and $f(m')$ have no subfields in common if m and m' are different. This requirement is stronger than collision-freeness on f, and was identified as necessary within the analysis in Evans and Schneider, 2001a.

The algebraic equivalences are built into the definition of the rank function, in the clauses of the rank function which consider whether the message is equal to some particular message (i.e. the clauses for h and for $f(m)$).

The rank of $m_1 \oplus m_2$ is obtained by combining the ranks of m_1 and m_2, applying the function from Line 2 pointwise.

The lengths of messages is an issue that has been mentioned but skirted in this paper. Here we are assuming that all atomic messages have the same unit length, and that $\oplus$ only combines messages of the same length. A more complete treatment would incorporate a more explicit notion of length into the message notation, perhaps describing a message as a pair (m, n) where n gives its length.

5. Discussion

In this paper we have reviewed the rank function approach to verifying authentication protocols. We have considered the impact of introducing equations reflecting algebraic properties of concrete crypto-mechanisms,

$$
\begin{array}{rcl}
\rho(t) & = & \langle 1\rangle \\
\rho(i) & = & \langle 1\rangle \\
\rho(n) & = & \langle 1\rangle \\
\rho(k) & = & \langle 0\rangle \\
\rho(f) & = & \langle 1\rangle \\
\rho(P(i)) & = & \begin{cases} \langle 0\rangle & \text{if } i = A \text{ or } i = B \\ \langle 1\rangle & \text{otherwise} \end{cases} \\
\rho(h) & = & \begin{cases} \langle 0\rangle & \text{if } \exists\, n_1, n_2, k, h_1. \\ & \qquad f(n_1.n_2.A.P(B)) = k.h.h_1 \\ & \qquad \vee\, f(n_1.n_2.A.P(B)) = k.h_1.h \\ & \qquad \vee\, f(n_1.n_2.B.P(A)) = k.h.h_1 \\ & \qquad \vee\, f(n_1.n_2.B.P(A)) = k.h_1.h \\ \langle 1\rangle & \text{otherwise} \end{cases} \\
\rho(m_1.m_2) & = & \rho(m_1) \frown \rho(m_2) \\
\rho(m_1 \oplus m_2) & = & \eta(\rho(m_1), \rho(m_2)) \\
\rho(f(m)) & = & \begin{cases} \langle 0,0,0\rangle & \text{if } \exists\, n_1, n_2. m = n_1.n_2.A.P(B) \\ \langle 0,0,0\rangle & \text{if } \exists\, n_1, n_2. m = n_1.n_2.B.P(A) \\ \langle 1,1,1\rangle & \text{otherwise} \end{cases}
\end{array}
$$

where

$$
\begin{array}{rcl}
\eta(m,n) & = & \begin{cases} 1 & \text{if } m = n \\ 0 & \text{otherwise} \end{cases} \\
\eta(ms_1 \frown ms_2, ns_1 \frown ns_2) & = & \eta(ms_1, ns_1) \frown \eta(ms_2, ns_2)
\end{array}
$$

Figure 9. Rank function: $\rho : MESSAGE \to seq(\{0,1\})$

with particular reference to an example making use of exclusive-or. The central theorem remains applicable in this case, but the definition of the rank function, generally given inductively over the space of messages, must be more carefully considered to ensure that it is well-defined in the presence of the additional equations. In this case we found it more appropriate to provide a more elaborate space of rank values to reflect this need. The paper contains an initial investigation into the impact of equations on the message space. More protocol examples, with equations for different mechanisms, will need to be carried out in order to obtain any general picture of the practical impact on this approach to protocol verification.

Much previous work by this author and others on protocol analysis relies on the perfect encryption assumption. The impact of introducing equations into the various approaches is not clear and is an area of current research. For example, there are results that allow an analysis to assume that all messages are well-typed (see Heather et al., 2000). It is not clear that these results always remain applicable when equations are introduced. Obtaining conditions when these results are applicable is a current area of research activity.

Mechanised assistance for the rank function approach to protocol verification has also been developed. A general theory for CSP has been provided in the theorem-prover PVS (Owre et al., 1993), together with a security protocol-specific theory on top of that, given in Dutertre and Schneider, 1997, Bryans and Schneider, 1997, and Evans and Schneider, 2001b. It seems likely that generalising the results to use an arbitrary set $RANK$ as introduced in this paper would be straightforward.

With regard to the introduction of equations, some aspects of the algebra of exclusive-or were already incorporated into the PVS analysis reported in Evans and Schneider, 2001a. However, it is not yet clear whether in the most general case additional proof obligations concerning well-definedness of rank functions could be introduced within the existing framework. Abstract datatypes in PVS are freely generated, giving rise to useful axioms enabling inductive reasoning. The introduction of algebraic identities undermines these results, for example by requiring functions to be defined on the message space rather than as constructors of the message datatype. This will make reasoning more messy and less automated.

There are also circumstances in which rank functions to verify protocols can be generated automatically (where they exist), as detailed in Heather, 2000. This is possible because of the result that in an automated construction of a rank function we only need to consider messages which are sub-messages of real protocol messages. In the presence of equations the notion of 'sub-message' is less clear, and we are currently investigating conditions on equations for which the approach remains valid.

Acknowledgements

I am grateful to Neil Evans and James Heather for their detailed comments on an earlier draft of this paper.

References

Abadi, M. and Gordon, A. (1998). A calculus for cryptographic protocols: the spi calculus. *Information and Computation*. also DEC Research Report 149, 1998.

Bryans, J. and Schneider, S. (1997). CSP, PVS, and a recursive authentication protocol. In *DIMACS Workshop on Design and Formal Verification of Crypto Protocols*.

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

Dutertre, B. and Schneider, S. (1997). Embedding csp in pvs: an application to authentication protocols. In *tpHOL*.

Evans, N. and Schneider, S. A. (2001a). An authentication protocol using one-way functions: a pvs analysis. Technical report, Royal Holloway, University of London.

Evans, N. and Schneider, S. A. (2001b). A practical introduction to using CSP and PVS to prove authentication properties of security protocols. Technical report, Royal Holloway, University of London.

Even, S., Goldreich, O., and Shamir, A. (1985). On the security of ping-pong protocols when implemented using the RSA. In *Advances in Cryptology*—CRYPTO *'85*.

Fábrega, F. J. T., Herzog, J. C., and Guttman, J. D. (1998). Strand spaces: why is a security protocol correct? In *IEEE Symposium on Security and Privacy*.

Germeau, F. and Leduc, G. (1997). Model-based design and verification of security protocols using lotos. In *DIMACS Workshop on Design and Formal Verification of Crypto Protocols*.

Gong, L. (1989). Using one-way functions for authentication. *Computer Communication Review*, 19(5).

Gordon, A. D. and Jeffrey, A. (2001). Authenticity by typing for security protocols. In *14th IEEE Computer Security Foundations Workshop*.

Heather, J., Lowe, G., and Schneider, S. (2000). How to prevent type flaw attacks on security protocols. In *13th IEEE Computer Security Foundations Workshop*.

Heather, J. A. (2000). *"Oh! Is it* really *you?"—Using rank functions to verify authentication protocols*. PhD thesis, Royal Holloway.

Heather, J. A. and Schneider, S. A. (2000). Towards automatic verification of authentication protocols on unbounded networks. In *13th IEEE Computer Security Foundations Workshop*.

Kemmerer, R. (1989). Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7.

Kemmerer, R., Meadows, C., and Millen, J. (1994). Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2).

Lowe, G. (1995). An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters, 56,*, 56.

Lowe, G. (1996). Breaking and fixing the needham-schroeder public-key protocol using FDR,. In *TACAS*, volume 1055 of *LNCS*. Springer.

Lowe, G. (1997). A hierarchy of authentication specifications. In *10th IEEE Computer Security Foundations Workshop*.

Marrero, W., Clarke, E., and Jha, S. (1997). Model checking for security protocols. In *DIMACS Workshop on Design and Formal Verification of Crypto Protocols*.

Meadows, C. (1992). Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1).

Millen, J. (1995). The interrogator model. In *IEEE Computer Society Symposium on Research in Security and Privacy.*

Needham, R. and Schroeder, M. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12).

Owre, S., Shankar, N., and Rushby, J. (1993). The PVS specification language. Technical report, Computer Science Lab, SRI International.

Paulson, L. (1998). The inductive approach to verifying cryptographic protocols. *Journal of Computer Security.*

Pereira, O. and Quisquater, J.-J. (2000). On the perfect encryption assumption. In *Workshop on Issues in the Theory of Security.*

Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM.*

Roscoe, A. (1995). Modeling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop.*

Roscoe, A. (1997). *The Theory and Practice of Concurrency.* Prentice-Hall.

Ryan, P. and Schneider, S. (1998). An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters.*

Ryan, P. Y. A., Schneider, S. A., Goldsmith, M. H., Lowe, G., and Roscoe, A. W. (2000). *Modelling and Analysis of Security Protocols.* Addison-Wesley.

Schneider, S. (1996). Security properties and CSP. In *IEEE Computer Society Symposium on Research in Security and Privacy.*

Schneider, S. (1999). *Concurrent and Real-time Systems: the CSP Approach.* Addison-Wesley.

Schneider, S. A. (1998). Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering.*