Temporal rank functions for forward secrecy

Rob Delicata and Steve Schneider
Department of Computing, University of Surrey, UK.
{R.Delicata, S.Schneider}@surrey.ac.uk

Abstract

A number of key establishment protocols claim the property of forward secrecy, *where the compromise of a long-term key does not result in the compromise of previously computed session-keys. We describe how such protocols can be modelled using the process algebra CSP and explain why the well-known rank function approach is incapable of proving their correctness. This shortcoming motivates us to propose a generalised proof technique based on the novel concept of a* temporal *rank function. We apply this approach to two examples: a protocol due to Boyd and the Cliques (A-GDH.2) group key agreement protocol.*

1. Introduction

A number of cryptographic protocols have appeared in the literature that claim to provide *forward secrecy*. The idea of forward secrecy is that if a long-term key is compromised then any session-keys that were previously established using the long-term key should remain secret. Forward secrecy is important in scenarios where session-keys need protection beyond the time-span during which they are used. These situations typically arise when session-keys are used for data encryption, rather than just authentication. There appears to be a disparity between the growing number of protocols that claim forward secrecy [2, 7, 17, 16, 1, 18, 15] and the work carried out on its formal analysis. In contrast to secrecy and authentication, the formal verification of forward secrecy has, with some exceptions [20, 4], received little attention in the literature. This paper fills the gap for the rank function approach.

This paper demonstrates how the notion of a rank function can be generalised to permit the verification of forward secrecy. The rank function approach [23] has previously been used to reason about authentication, secrecy and non-repudiation in a number of cryptographic protocols [23, 25, 9, 5]. In the standard approach, a rank function is used to partition the message-space of a protocol such that all messages which should remain secret are assigned a rank of sec and all messages that might be public are assigned a rank of pub. Typically, the protocol is modelled using the process algebra Communicating Sequential Processes (CSP) [22, 24]. The central rank theorem gives a series of healthiness conditions that the function must satisfy in order for us to conclude that the protocol does actually maintain the secrecy of a given set of messages. Stated another way, the rank theorem gives the conditions under which it is reasonable to conclude that the sets of public and secret messages are disjoint.

It turns out that this all-or-nothing view of secrecy is unable to capture properties of protocol models which include compromised keys. Since compromised keys become public after some initial period of secrecy they can neither be classified as entirely secret nor entirely public, and the standard approach cannot model values that inhabit this middle-ground. Since forward secrecy is dependent on the concept of compromised keys, protocols which provide forward secrecy are unverifiable using the standard approach. Although this incompleteness motivates the richer concept of a *temporal rank* that we propose here, a discussion of the shortcoming is somewhat incidental to the main thrust of the paper. For this reason we delay it until Section 5. Our goal is not to introduce yet-another-verification-technique but to show how an existing technique can be generalised to permit reasoning about a wider class of security properties.

In the next section we describe how compromised keys can be modelled using CSP and show how forward secrecy can be formalised as a trace specification over such a protocol model. In Section 3 we introduce the idea of a temporal rank function and establish a central theorem that gives conditions under which we can conclude that a protocol guarantees forward secrecy. We demonstrate this approach on the running example of a protocol due to Colin Boyd and in Section 4 consider a further example: the Cliques (A-GDH.2) group key agreement protocol. As mentioned, Section 5 discusses why the original rank function approach fails in the presence of compromised keys, and Section 6 concludes. Some knowledge of CSP is assumed but we provide a summary of relevant notation in Appendix A.

2. Formalising forward secrecy

The concept of forward secrecy was first introduced by Günther [10] who used the term *perfect forward secrecy*. We follow the lead of other authors [14, 3] in dropping the word perfect and avoiding confusion with the unrelated concept of perfect secrecy.

Forward secrecy is captured by the following definition:

Definition 1 *A key establishment protocol provides forward secrecy if compromise of the long-term keys of a set of principals does not compromise the session-keys established in previous protocol runs involving those principals.*

As a running example we use a protocol due to Boyd [3] that takes its cue from an earlier idea by Wiener [26]. The protocol allows a session-key k to be established between two principals, a and b. a generates a fresh asymmetric key-pair (pk, sk) and sends the public half to b along with a fresh nonce, na, and the signature of pk under a long-term signing key $Sig(a)$. b uses the public-key to encrypt the session-key k and sends it to a along with a signature containing a hash of the session-key:

$$\begin{array}{llcll} 1. & a & \rightarrow & b & : \quad pk \cdot na \cdot \{pk \cdot b\}_{Sig(a)} \\ 2. & b & \rightarrow & a & : \quad \{k\}_{pk} \cdot \{h(k) \cdot a \cdot na\}_{Sig(b)} \end{array}$$

(we write $m_1 \cdot m_2$ for the pairing of messages m_1 and m_2 and use $\{m\}_k$ to denote the encryption of m under a key k). This protocol aims to meet the forward secrecy property of Definition 1 if a and b are honest and the long-term signing keys $Sig(a)$ and $Sig(b)$ are not compromised until after the protocol has ended. This can be argued informally in the following way. The keys $Sig(a)$ and $Sig(b)$ are used only to authenticate the session-key, and although their disclosure allows an attacker to masquerade as a or b in future protocol runs, it does not allow him to recover the past session-key. The session-key can only be recovered with the private-key, sk, which is freshly generated for each protocol run. Asymmetric key-pairs, such as (pk, sk), which are discarded after a single protocol run are termed *ephemeral* keys, and it is the use of ephemeral keys that enables the protocol to provide forward secrecy.

Boyd and Mathuria [3] call the property in Definition 1 *full* forward secrecy in distinction to the weaker property of *partial* forward secrecy:

Definition 2 *A protocol provides partial forward secrecy if compromise of the long-term keys of one or more specific principals does not compromise the session keys established in previous protocol runs involving those principals.*

The difference between full and partial forward secrecy centres around the number of long-term keys that are compromised. For example, a protocol involving two honest parties, A and B, will provide full forward secrecy if a past session-key remains secure when *both* A's and B's long-term keys are compromised. If this property is not met, however, the protocol may still provide partial forward secrecy if the compromise of just one long-term key (either A's or B's, but not both) prevents the intruder from discovering a past session-key. Full forward secrecy takes a pessimistic view by assuming the disclosure of all long-term keys, and for the purpose of protocol *verification* it will often be prudent to apply this stronger condition. For protocol *analysis*[1], however, it is often useful to consider partial forward secrecy since, if the protocol does not meet the weaker goal, it will certainly not satisfy the stronger condition. For simplicity, the present work concentrates on the notion of partial secrecy, although this is by no means a limitation of our approach. In particular, we will show that Boyd's protocol achieves partial forward secrecy with respect to the key $Sig(A)$ for some (honest) initiator A.

Pereira makes an interesting distinction between what he terms *complete* forward secrecy and *individual* forward secrecy [19]. In complete forward secrecy it is assumed that the past protocol runs were executed without any interference by the intruder, who is passive and merely records the messages that pass on the network. In contrast, individual forward secrecy assumes an intruder who has potentially manipulated the messages of one or more principals in previous runs. The terms 'complete' and 'individual' are appropriate for Pereira's focus on principals in group Diffie-Hellman protocols. Here, however, we will refer to them as *passive* and *active* forward secrecy, respectively, in recognition of the intruder's role in past protocol runs. When we talk about forward secrecy in this paper we are referring specifically to the notion of active forward secrecy. Clearly, a protocol that provides active forward secrecy also achieves the weaker goal of passive forward secrecy.

2.1. Compromised keys

We model protocols in CSP where each principal, and a special intruder process called *ENEMY*, are represented by processes. A principal A can transmit a message M to B by performing the event $trans.A.B.M$. Conversely, B can receive this message by performing $rec.B.A.M$. We assume a worst-case-scenario where all data communication (anything passed on the channels *trans* and *rec*) passes through the intruder. This arrangement is shown in Figure 2. A side-effect of this network composition is that *ENEMY*, unlike the user processes, 'hears' messages on channel *trans* and 'says' messages on channel *rec*.

1 Our distinction between protocol verification and protocol analysis lies in the goal of the endeavour: the first aims to prove correctness whilst the second attempts to discover attacks. In general, failure to find an attack does not imply correctness.

PAIRING
$$\frac{S \vdash m_1 \qquad S \vdash m_2}{S \vdash m_1 \cdot m_2}$$

UNPAIRING
$$\frac{S \vdash m_1 \cdot m_2}{S \vdash m_1 \qquad S \vdash m_2}$$

MEMBERSHIP
$$\frac{}{S \vdash m} \quad (m \in S)$$

HASHING
$$\frac{S \vdash m}{S \vdash h(m)}$$

ENCRYPTION
$$\frac{S \vdash m \qquad S \vdash k}{S \vdash \{m\}_k}$$

DECRYPTION
$$\frac{S \vdash \{m\}_k \qquad S \vdash k}{S \vdash m}$$

Figure 1. Intruder deductions

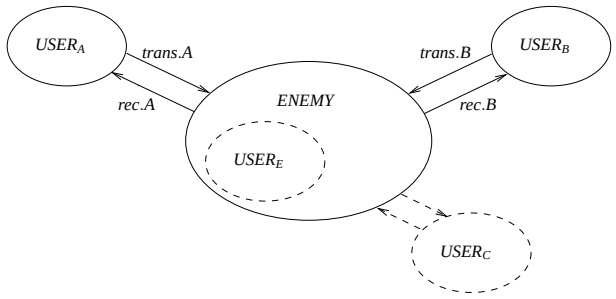

Figure 2. Network arrangement

Forward secrecy becomes relevant in situations where long-term keys can become compromised. We are not concerned with how the keys are leaked (as a result of cryptanalysis, for example), but only with the fact that they are. We develop a process *ENEMY* that represents an active Dolev-Yao style intruder [8] with the additional ability to learn long-term keys that have been leaked.

2.1.1. The intruder process The model of *ENEMY* is built around a set S of facts that the intruder knows and a 'generates' relation $\vdash$ that the intruder can use to deduce new facts. We write $S \vdash m$ if the intruder can generate the message m by knowing all messages in the set S. The capabilities captured by $\vdash$ are given in Figure 1. The process *ENEMY* is parameterised by a set *IIK* — the Intruder's Initial Knowledge — that contains the facts available to the intruder at the start of the protocol. This set will typically include the usernames of all principals, any long-term public-keys and other keys that the intruder may reasonably know.

The intruder can (i) hear any message on the network and add that message to his knowledge set, (ii) send to any principal any message generable under $\vdash$, and (iii) compromise any key in a set *TS* of temporary secrets and add it to his knowledge set:

$$\begin{aligned}
&ENEMY = Enemy(IIK)\\
&Enemy(S) =\\
&\quad trans?a?b?m \rightarrow Enemy(S \cup \{m\})\\
&\quad \Box \Box_{\substack{a:\mathsf{U},\\ b:\mathsf{U},\\ m \mid S \vdash m}} rec.a.b.m \rightarrow Enemy(S)\\
&\quad \Box \Box_{m:TS} leak.m \rightarrow Enemy(S \cup \{m\})
\end{aligned}$$

(where U is the set of identities of all principals). The definition of *ENEMY* is not entirely general since we can adapt it to facilitate the compromise of different sets of keys. Doing so will be dependent on both the protocol under consideration and the precise forward secrecy goal that we intend to verify. To reason about full forward secrecy in Boyd's protocol, for example, we would define $TS = \{Sig(a) \mid a \in \mathsf{U}\}$. Note that the intruder can always signal his knowledge of a message M by performing the event $trans.E.E.M$, where E is the intruder's identity.

2.1.2. The principals We model each principal that takes part in the protocol as a CSP process representing the protocol steps performed by that principal. For example, consider $User^I(a, b, pk, na)$ from Figure 3. This process models a principal a running Boyd's protocol as initiator (denoted by the superscripted I) with b, where pk is the ephemeral public-key and na is the nonce used in the run. Similarly, $User^R(b, a, k)$ is a principal b running as responder, apparently with a, using the session-key k. On receipt of the first protocol message, b verifies the signature to satisfy himself that the message originated from a. The responder asserts his belief that k will be a secret shared only with a by performing the signal event $signal.b.a.k$.

The network composition is shown in Figure 3. The process $USER_a$ represents the entire behaviour of a principal a in both initiator and responder roles. In the initiator case a session variable pk is chosen from the set PK_{ab} of public-keys that a will use in runs with a particular principal b. The function *nonce* is an injective mapping from public-keys to nonces such that $nonce(pk)$ is the nonce that a will use in the run involving pk. Similarly, as a responder, the interleaving is indexed by the session-key $k \in \mathsf{K}_{ba}$. The sets PK_{ab} and K_{ab} of public- and session-keys are all pairwise disjoint.

We incorporate the *ENEMY* and user processes into a standard CSP protocol model in which an arbitrary number of principals can engage in arbitrarily many instances of the protocol concurrently. By convention, the process representing the entire network is called *NET*.

2.2. Statement of forward secrecy

A CSP process can be characterised by its traces, where a *trace* represents one possible sequence of events that the process can perform. For a process P, $traces(P)$ is the set of all possible traces of P. The set $traces(P)$ is always non-empty since every process can perform the empty trace $\langle\rangle$

$$User^I(a,b,pk,na) =$$
$$\quad trans.a.b.(pk \cdot na \cdot \{pk \cdot b\}_{Sig(a)})$$
$$\quad \rightarrow rec.a.b?(\{k\}_{pk} \cdot \{h(k) \cdot a \cdot na\}_{Sig(a)})$$
$$\quad \rightarrow Stop$$

$$User^R(b,a,k) =$$
$$\quad rec.b.a?(pk \cdot na.\{pk \cdot b\}_{Sig(a)})$$
$$\quad \rightarrow signal.b.a.k$$
$$\quad \rightarrow trans.b.a.(\{k\}_{pk} \cdot \{h(k) \cdot a \cdot na\}_{Sig(a)})$$
$$\quad \rightarrow Stop$$

$$USER_a =$$
$$\quad |||_{b:\mathsf{U}} \quad \left(|||_{pk:\mathsf{PK}_{ab}} User^I(a,b,pk,nonce(pk))\right) \; ||| \; \left(|||_{k\in\mathsf{K}_{ab}} User^R(a,b,k)\right)$$

$$NET = \left(|||_{a:\mathsf{U}} USER_a\right) \; |[\,\{|\, trans, rec \,|\}\,]| \; ENEMY$$

Figure 3. CSP model of Boyd's protocol

and is prefix-closed since, if $s \frown t$ is a trace of P, then s is also a trace of P.

A trace specification is a predicate on traces, and we write $W(tr)$ if the predicate W holds for a trace tr. A process P satisfies W (written P **sat** W) if all its traces satisfy W:

$$P \textbf{ sat } W \Leftrightarrow \forall\, tr \in traces(P).W(tr)$$

We are already in a position to prove that any message that can be sent by *ENEMY* must be generable from the set *IIK* together with the messages input on channels *trans* and *leak*.

Theorem 1

$$ENEMY \textbf{ sat } (IIK \cup (tr \Downarrow \{trans, leak\})) \vdash tr \Downarrow rec$$

Proof (sketch). We prove by a mutual recursion induction that $Enemy(S)$ **sat** $(S \cup (tr \Downarrow \{trans, leak\})) \vdash tr \Downarrow rec$ and the result follows from the fact that $ENEMY = Enemy(IIK)$. □

We introduce two important trace specifications. The first is the **secret** predicate which states that no event $t \in T$ occurs in the trace tr:

Definition 3

$$\textbf{secret } T = tr \upharpoonright T = \langle\rangle$$

(where $tr \upharpoonright T$ is the trace tr restricted to events in T). Although T is a set of events, it will be convenient to think of it as a set of messages that we wish to keep secret, and we abuse the notation to write (for example) $T = \{M\}$ where we should more properly have written $T = \{trans.a.b.M \mid a \in \mathsf{U}, b \in \mathsf{U}\}$.

The second important trace predicate is **St_precedes** which states that, for disjoint sets R and T, any occurrence of an event $t \in T$ in a trace will be strictly preceded by some occurrence of an event $r \in R$:

Definition 4

$$R \textbf{ St\_precedes } T =$$
$$\quad tr \upharpoonright T \neq \langle\rangle \Rightarrow$$
$$\qquad \exists\, tr' < tr.tr' \upharpoonright R \neq \langle\rangle \wedge tr' \upharpoonright T = \langle\rangle$$

Consider the process *NET* from Figure 3. If a trace $tr_0 \in traces(NET)$ contains the event $leak.Sig(A)$ we know that the intruder has, at that point, learned the signing-key of principal A. Similarly, if tr_0 contains the event $signal.B.A.K$ we can conclude that, at that point, B was willing to establish a key K with a principal who B believed to be A. If $leak.Sig(A)$ appears in the trace *before* $signal.B.A.K$ then the ephemeral key used to encrypt K may have come from the intruder, who invents a key-pair (PK, SK) and a nonce NA and uses $Sig(A)$ to masquerade as A:

1. $E(A) \rightarrow B \quad : \{PK, B\}_{Sig(A)}$
2. $B \rightarrow E(A) \quad : \{K\}_{PK}, \{h(K), A, NA\}_{Sig(B)}$

(where $E(A)$ denotes the intruder masquerading as A). If, on the other hand, $signal.B.A.K$ occurs *before* $leak.Sig(A)$ (i.e., if the predicate $\{signal.B.A.K\}$ **St_precedes** $\{leak.Sig(A)\}$ holds) then the forward secrecy property tells us that the compromise of $Sig(A)$ should not result in the compromise of the (previously established) session-key K. We can therefore state the claim **secret**$\{K\}$: the intruder never learns K.

This is the heart of our statement of forward secrecy: that all session-keys established between honest principals *before* the compromise of the long-term key should remain secret even *after* the long-term key has been leaked. If k is a session-key established between two honest principals a and b and lk is a long-term key used to establish k, we define the trace specification **fs** that captures this notion of forward secrecy:

Definition 5

$$\textbf{fs}(k, lk) =$$
$$\quad \{signal.b.a.k\} \textbf{ St\_precedes } \{leak.lk\}(tr)$$
$$\quad \Rightarrow \textbf{secret}\{k\}(tr)$$

For our running example, represented by the process *NET*, we can state the specification as:

$$NET \textbf{ sat } \textbf{fs}(K, Sig(A))$$

That is, if B is willing to establish the session-key $K \in \mathsf{K}_{BA}$ with A, before the key $Sig(A)$ is compromised, then K remains a secret known only to A and B. Note that, if B will-

ingly engages in a run with a dishonest principal then neither secrecy nor forward secrecy of the resulting session-key can be expected.

2.3. Restricting the network

Let $k_0 \in \mathsf{K}_{ba}$ be a session-key established using a long-term key lk_0 between two honest principals a and b. Our goal is to prove that NET **sat** $\mathbf{fs}(k_0, lk_0)$ for some protocol model NET. However, we can ease this task by first applying some simple CSP manipulation. Expanding the definition of **fs** for NET, we obtain the goal:

$$NET \textbf{ sat } (\{signal.b.a.k_0\} \textbf{ St\_precedes } \{leak.lk_0\} \Rightarrow \textbf{secret } \{k_0\})$$

Let TR be the set of traces of NET that satisfy the antecedent $\{signal.b.a.k_0\}$ **St_precedes** $\{leak.lk_0\}$:

Definition 6

$$TR = \{tr \in traces(NET) \mid \{signal.b.a.k_0\} \textbf{ St\_precedes } \{leak.lk_0\}(tr)\}$$

Lemma 1 $TR \subseteq traces(NET)$

This is an immediate consequence of Definition 6.

TR is both non-empty (since the empty trace $\langle\rangle$ satisfies the antecedent) and prefix-closed (since, if $s \frown t$ satisfies the antecedent then so will s). The trace set TR can be seen as representing a restriction of the network described by NET. Since a CSP process is defined in terms of its traces we can (at least in this case) define a new process, NET', whose traces are precisely those in the set TR. Intuitively, NET' is the process whose traces are those of NET in which $signal.b.a.k_0$ always precedes $leak.lk_0$.

Let K be the set of session-keys from which k_0 is drawn and let LK be the set of long-term keys from which lk_0 is drawn. RES is a process that blocks the event $leak.lk_0$ until such time as $signal.b.a.k_0$ has occurred:

$$\begin{aligned} RES \quad = \quad & signal?b?a?k \rightarrow \\ & \quad \text{if } k = k_0 \text{ then } RUN_X \text{ else } RES \\ & \Box \quad leak?k : (\mathsf{LK} \setminus \{lk_0\}) \rightarrow RES \end{aligned}$$

where $X = \{| \, signal, leak \, |\}$

RUN_X is a special process that is always willing to communicate any event in the set X (See Appendix A). Based on this we can define NET' as follows:

Definition 7 $NET' = NET \,|[\, X \,]|\, RES$

NET' forces the synchronisation of all *leak* and *signal* events between NET and RES, and we can demonstrate that NET' has as its traces precisely the set TR:

Theorem 2 $traces(NET') = TR$

Proof. Consider a trace $tr \in traces(NET')$. By definition of NET', $tr \in traces(NET \,|[\, \{| \, signal, leak \, |\} \,]|\, RES)$. The semantics of CSP interface parallel then tell us that $tr \in traces(NET)$ (i) and $tr \upharpoonright \{| \, signal, leak \, |\} \in traces(RES)$. By definition of RES we have that whenever $leak.lk_0$ appears in tr it is preceded by an event $signal.b.a.k_0$ for some a, b, and so $\{signal.a.b.k_0\}$ **St_precedes** $\{leak.lk_0\}(tr)$ (ii). Finally from Definition 6, (i) and (ii) yield that $tr \in TR$. The same argument works in the opposite direction to show that $tr \in TR \Rightarrow tr \in traces(NET')$. □

The assumption $\{signal.b.a.k_0\}$ **St_precedes** $\{leak.lk_0\}$ on NET has been absorbed into NET' and this enables us to rewrite the **fs** trace specification to a simpler form:

$$NET' \textbf{ sat secret } \{k_0\}$$

In doing so we have reduced a forward secrecy predicate over the whole network (NET) to a secrecy predicate over a restricted subset (NET') of that network. If we can prove that NET' **sat secret** $\{k_0\}$ then we will have shown that NET **sat** $\mathbf{fs}(k_0, lk_0)$. Applied to our running example, we are left to prove that NET' **sat secret** $\{K\}$ where $K \in \mathsf{K}_{ba}$ for some honest principals a and b.

3. Temporal rank functions

In this section we introduce a proof technique suitable for verifying whether the forward secrecy property of the previous section holds for a given protocol. In particular, we obtain a specialised theorem that applies to secrecy properties on the protocol network. This theorem is the core of the proof strategy presented in this paper; it provides a sufficient list of conditions whose achievement guarantees that NET' **sat secret** T for a set T.

3.1. Time tags

Consider an instance of the Boyd protocol where the session-key K is established between A and B using the long-term key $Sig(A)$ and that $Sig(A)$ is subsequently compromised at some time n. For simplicity we assume that B's signing key, $Sig(B)$, is not leaked, and so define the set TS of temporary secrets as $TS = \{Sig(A)\}$. The intruder's capabilities are increased at time n since from this point he can use $Sig(A)$ to sign anything he knows and can send the resultant message to any principal willing to accept it, whilst masquerading as A. The messages that the intruder can generate can be divided into several categories: (i) those he can generate before time n, (ii) those he can generate at or after time n, and (iii) those he can never generate. In fact, we can go further and tag each message m in the space of the protocol with a label $t \in \mathbb{N}^{\infty}$ representing the earliest time

at which *m* can be assumed to be generable by the intruder without affecting the correctness of the protocol.

For example, since usernames are generally included in the intruder's initial knowledge we would tag *A* and *B* with 0, denoting the fact that the intruder knows them at time 0. Similarly, since *A* makes the ephemeral public-key *pk* available before $Sig(A)$ is leaked we would tag it with a label in the range $0 \leq t < n$. Furthermore, since $Sig(A)$ becomes available at time *n* we would label it as such. Finally, messages (such as the session-key *K*) which should never be generable by the intruder would be tagged with infinity (∞). It may be helpful to think of ∞ as denoting that the message is not generable in finite time, although for practical purposes it will only mean that it is computationally infeasible for the intruder to deduce the message.

Our verification approach is based on the interpretation of a time tag as a *rank*, and we will say that a message *m* has a rank *t* (written $\rho(m) = t$) if it is safe to assume that *m* is generable by the intruder at time *t*. We will wish to prove that, at time *t*, the intruder is incapable of generating any message with a rank greater than *t*. This will enable us to conclude that messages such as *K* are actually not generable by the intruder until time ∞ and, therefore, remain secret. In the remainder of this section we formalise the notion of a temporal rank, introduce our central theorem, and show how we can use it to prove that Boyd's protocol meets its forward secrecy goal.

3.2. Temporal ranks

A rank function is a function from the space of messages M to the set of natural numbers with infinity: $\mathbb{N}^\infty$.

Definition 8 *Let* M *be the message-space of a protocol. Then define* ρ *to be the function* $\rho : \mathsf{M} \to \mathbb{N}^\infty$

Our intention is to define a rank function over the message-space of NET' such that the rank of a message *m*, written $\rho(m)$, respects the time at which *m* becomes available to the intruder. Messages known initially to the intruder ($m \in IIK$) will have $\rho(m) = 0$. If, on the other hand, *m* should be unavailable to the intruder we will typically assign it a rank of ∞. Messages that become available to the intruder during the course of the protocol will be assigned some (possibly non-zero) finite rank. A rank function effectively partitions the message-space into the set of messages that the intruder can know (those with finite ranks) and those that should remain secret (those with a rank of ∞). Rank functions operate on messages communicated as events in the traces of a CSP process. We lift ranks to events in the sense that $\rho(trans.a.b.m) = \rho(rec.a.b.m) = \rho(leak.m) = \rho(m)$ and to sets by defining $\rho(S)$ to be the rank of the highest ranked message in *S*.

Generally speaking, the traces of a CSP process do not have any concept of time. However, we can impose a sense of discrete time onto traces by taking the time of an event to be its position in a trace. We define the *leak-time* of a key *k* in a trace *tr* (written $\tau(k, tr)$) to be the position in the sequence *tr* at which *leak.k* first occurs. If *leak.k* does not occur in *tr* then $\tau(k, tr) = \infty$.

Definition 9 *Assuming that* $k \notin IIK$*, if* $\neg(leak.k \text{ in } tr)$ *then* $\tau(k, tr) = \infty$*, else let* $tr = tr' \frown \langle leak.k \rangle \frown tr''$ *such that* $\neg(leak.k \text{ in } tr')$*, then* $\tau(k, tr) = \#tr' + 1$

The following examples will help to clarify this:

1. If $tr_0 = \langle leak.k \rangle$ then $\tau(k, tr_0) = 1$.
2. If $tr_1 = \langle trans.a.b.m, rec.b.a.m, signal.a.b.k, leak.k \rangle$ then $\tau(k, tr_1) = 4$.
3. If $tr_2 = \langle trans.a.b.m_1, rec.b.a.m_1, trans.a.b.m_2 \rangle$ then $\tau(k, tr_2) = \infty$.

Note that, since *leak.k* is not preceded by the corresponding signal event in tr_0, $tr_0 \notin traces(NET')$.

As discussed earlier, the rank of a message may be dependent on the leak-time of some long-term key. Furthermore, the leak-time will not generally be fixed but will occur at different times for different traces of NET'. Rather than define a distinct rank function for each possible scenario we use the concept of a family of rank functions $\underline{\rho}$, parameterised by *n*, such that the rank function ρ_n will apply precisely when the leak-time of *k* in a trace $tr_0 \in traces(NET')$ is *n*. When *n* is understood we may omit the subscript and simply write ρ.

Let *T* be a set of values that should not appear in any trace of NET'. *T* will usually contain the session-key *k*, but may also include long-term secrets that we disallow the intruder from compromising. We are now in a position to give a list of conditions that, if satisfied, guarantee that no member of *T* can be discovered by the intruder in NET'. Following the statement of the central theorem, we discuss each condition in turn.

Theorem 3 *Let T be a set of messages and let IIK be the intruder's initial knowledge. If there exists a family of rank functions,* $\underline{\rho}$*, such that each member* ρ_n *satisfies:*

C1 $\rho_n(IIK) = 0$

C2 $\forall S \subseteq \mathsf{M}, m \in \mathsf{M}. S \vdash m \wedge \rho_n(S) = t \Rightarrow \rho_n(m) \leq t$

C3 $\forall m \in T. \rho_n(m) = \infty$

C4 $\forall u \in \mathsf{U} \setminus \{E\}, tr \in traces(NET').$
$tr \upharpoonright \{| trans.u, rec.u, signal.u |\}$ **holds** ρ_n

C5 $\forall m \in TS. \rho(leak.m) < \infty$

then NET' **sat secret** *T*

Proof. See Appendix B.

C1 — Condition **C1** essentially amounts to an assumption on the initial information available to the intruder. The requirement that $\rho_n(IIK) = 0$ states that, at the start of the

protocol, the intruder does not know any secret or compromisable keys.

C2 — The second condition:

$$\forall S \subseteq \mathsf{M}, m \in \mathsf{M}.S \vdash m \wedge \rho_n(S) = t \Rightarrow \rho_n(m) \leq t$$

is a requirement on the generates relation, $\vdash$. It says that if the intruder knows a set of messages S with rank t, any message m generated from S under $\vdash$ should have a rank no greater than t. Stated another way: at any time, the intruder can only generate messages which are safe for him to know at that time. Since, from **C1**, we have that the intruder begins by only knowing messages of rank 0, **C2** allows us to conclude that the intruder cannot send a message of rank ∞ unless he has previously received (from some other principal or via *leak*) a message with infinite rank.

C3 — This is a statement about the contents of the set T. Since T should contain the messages (or, more correctly, the events) that we wish to keep secret we must ensure that each of these messages has a rank of ∞. **C3** allows us to conclude that this is indeed the case.

C5 — The final condition is a straightforward obligation on the set of leakable secrets, namely, that they are assigned a finite rank.

C4 — This condition is essentially an obligation on the users of the system: that no honest principal can send a message of infinite rank unless it has previously received a message of infinite rank. The specification **holds** ρ_n is defined as follows:

Definition 10

$$\begin{array}{l}\textbf{holds}\ \rho_n \Leftrightarrow \\ \quad \rho_n(tr \Downarrow rec) \neq \infty \Rightarrow \rho_n(tr \Downarrow trans) \neq \infty\end{array}$$

By showing that **C4** holds we rule out the possibility that any of the honest principals introduces a message of infinite rank, and if the network admits such a message it must therefore have been introduced by the intruder. (Since **C1**, **C2** and **C5** tell us that the intruder cannot introduce such a message either, the combination of **C1**, **C2**, **C4** and **C5** tells us that no infinite rank message can pass through the system. Since all messages in T have rank ∞ (**C3**) we can conclude that all elements of T remain secret.) Curiously, the stronger, and more obvious, definition of 'holding the rank' — where the rank of emitted messages is not greater than the rank of received messages — is unnecessary (see the proof of Theorem 3 in Appendix B).

Note that **C4** is a condition upon the traces of events communicated by each principal and it is worthwhile to consider how the traces of a process $USER_a$ in NET are related to the traces involving a in NET'. Consider an arbitrary trace $tr_0 \in traces(NET')$ restricted to the events in which a principal A takes part:

$$tr_A = tr_0 \upharpoonright \{|\ trans.A, rec.A, signal.A\ |\}$$

NET', recall, is the parallel composition:

$$NET\ ||[\ \{|\ signal, leak\ |\}\ ]||\ RES$$

which blocks the performance of the event $leak.lk$ until $signal.a.b.k$ has been communicated. *leak* events originate at the *ENEMY* and not with user processes, so $tr_A \in traces(USER_A)$. Put another way, any behaviour possible by A in NET' will be exhibited by $USER_A$ in NET. When we come to prove **C4** for a given rank function it will be convenient, and sufficient, to do so by inspection of the processes $USER_a$ together with the assumption that, whenever $leak.Sig(A)$ occurs then $signal.a.b.K$ must have previously been communicated.

3.3. Proving Boyd's protocol correct

We now conclude our running example by showing how the above strategy can be used to prove that Boyd's protocol provides partial forward secrecy on an unbounded network:

$$\begin{array}{llllll} 1. & a & \rightarrow & b & : & pk \cdot na \cdot \{pk \cdot b\}_{Sig(a)} \\ 2. & b & \rightarrow & a & : & \{k\}_{pk} \cdot \{h(k) \cdot a \cdot na\}_{Sig(b)} \end{array}$$

The forward secrecy goal that we wish to establish corresponds to the following predicate (for some session-key $K \in \mathsf{K}_{BA}$ where A and B are honest):

$$NET\ \textbf{sat}\ \textbf{fs}(K, Sig(A))$$

We consider the CSP model of Figure 3 and look at a particular, but arbitrarily chosen, run in which A acts as initiator and attempts to establish a session-key with B. Similarly, B acts as responder and attempts to establish the session-key K with A. We tailor the *ENEMY* process by setting $TS = \{Sig(A)\}$. Our goal is to construct a family of rank functions that satisfy conditions **C1**–**C5** by characterising the times at which messages become known to the intruder. We define $T = \{K\}$ to be the set of values that must remain secret. We assume that $Sig(A)$ becomes available to the intruder via the channel *leak*:

Assumption 1 *$Sig(A)$ is unknown to the intruder until it appears on channel leak.*

Some care is needed in stating this assumption since it presumes the intruder cannot manipulate earlier protocol messages in a way that will cause $Sig(A)$ to be revealed. In this case the assumption can be justified by applying Guttman's concept of *immediate safety* [11]: $Sig(A)$ is immediately safe since it is not known initially to the intruder and is not sent as a component of any message. Under the perfect encryption assumption, $Sig(A)$ remains secret until it appears on channel *leak*.

A suitable rank function is given below. We omit a formal proof but instead present a discussion of the rank function and an informal argument as to why **C1**–**C5** hold. In

particular, we show how the restriction of *NET* to *NET′* is fundamental to the proof strategy.

We begin by defining the rank of usernames and nonces, which in both cases we set to be 0:

$$\begin{array}{lrcl} \textsc{Usernames} & \rho_n(u) & = & 0 \\ \textsc{Nonces} & \rho_n(na) & = & 0 \end{array}$$

We assume that the intruder has all usernames in his initial knowledge. Since the protocol uses public nonces we must assign them a finite rank. However, it turns out that the protocol is still secure if we assume, as we do here, that they are known to the intruder initially (by assigning a rank of 0). The idea of assigning a lower rank to a message than is strictly necessary is a useful trick that can help to reduce the complexity of a rank function. It is sound since it assumes the worst-case: if we can prove security in this context then the security goal will still hold if we later decide to assign the message a higher rank.

$$\begin{array}{lrcl} \textsc{Hashes} & \rho_n(h(m)) & = & \rho_n(m) \\ \textsc{Pairs} & \rho_n(m_1 \cdot m_2) & = & max(\rho_n(m_1), \rho_n(m_2)) \end{array}$$

We define the rank of a hashed message as equal to the rank of the unhashed message, since the intruder knows the hash function h and can deduce $h(m)$ as soon as he learns m. The rank of a concatenation $m_1 \cdot m_2$ is the greatest rank of its components.

$$\begin{array}{lrcl} \textsc{Eph-keys} & \rho_n(pk) & = & 0 \\ & \rho_n(pk^{-1}) & = & \begin{cases} \infty & \text{if } pk \in \mathsf{PK}_{AB} \\ 0 & \text{otherwise} \end{cases} \\ \textsc{Sig-keys} & \rho_n(Sig(u)) & = & \begin{cases} \infty & \text{if } u = B \\ n & \text{if } u = A \\ 0 & \text{otherwise} \end{cases} \\ & \rho_n(Sig^{-1}(u)) & = & 0 \\ \textsc{Sess-keys} & \rho_n(k) & = & \begin{cases} \infty & \text{if } k = K \\ 0 & \text{otherwise} \end{cases} \end{array}$$

When assigning a rank to keys, we give a rank of 0 unless it is absolutely necessary to do otherwise. The intention is to state the secrecy of just those values required to meet the security goal. So we assume, for example, that all session-keys apart from K are known initially to the intruder. Note that the rank of signing-keys respects the fact that $Sig(B)$ is never leaked and $Sig(A)$ is leaked at time n. We assume that all other signing-keys are known to the intruder from the start.

$$\textsc{Encrypts} \quad \rho_n(\{k\}_{pk}) = \begin{cases} \infty & \text{if } pk \notin \mathsf{PK}_{AB} \\ & \quad \wedge\, k = K \\ 0 & \text{otherwise} \end{cases}$$

The rank of encryptions follows from our assumption about the secrecy of session-keys. The intruder should never learn K and should never learn an encryption of K with any key other than those used by A to initiate to B.

The most subtle aspect of this rank function is its treatment of signatures. $Sig(B)$ is never leaked and, therefore, the only messages signed using $Sig(B)$ will be those matching the pattern of the protocol messages. For each principal u these messages are captured by $\mathcal{S}(u)$:

$$\begin{array}{rcl} \mathcal{S}(u) & = & \{pk \cdot v \mid pk \in \mathsf{PK}_u, v \in \mathsf{U}\} \\ & \cup & \{h(k) \cdot v \cdot na \mid k \in \mathsf{K}_u, v \in \mathsf{U}, na \in \mathsf{N}\} \end{array}$$

where $\mathsf{PK}_u = \bigcup_{v \in \mathsf{U}} \mathsf{PK}_{uv}$. We define the rank of signatures as follows:

$$\begin{array}{lrcl} \textsc{Signatures} & \rho_n(\{m\}_{Sig(B)}) & = & \begin{cases} 0 & \text{if } m \in \mathcal{S}(B) \\ \infty & \text{otherwise} \end{cases} \\ & \rho_n(\{m\}_{Sig(A)}) & = & \begin{cases} 0 & \text{if } m \in \mathcal{S}(A) \\ \max(\rho_n(m), n) & \\ & \text{otherwise} \end{cases} \\ & \rho_n(\{m\}_{Sig(u)}) & = & \rho_n(m) \quad (u \notin \{A, B\}) \end{array}$$

The interesting case involves the key $Sig(A)$. Before time n the only messages signed with $Sig(A)$ are those that the protocol designer intended: the messages in the set $\mathcal{S}(A)$. We can safely assume that such messages are available to the intruder initially, and so assign them a rank of 0. After n, however, the intruder can sign any message m he knows using $Sig(A)$. In this case, the rank of the resultant message is the greater of the two ranks $\rho_n(m)$ and n.

Since the intruder cannot engage in events on the *signal* channel, we can safely define $\rho_n(signal.a.b.k) = 0$.

Informally, the set *IIK* can be constructed to conform to condition **C1**: that all messages in *IIK* have a rank of 0. **C3**: that every message in T has a rank of ∞ — trivially holds since we have defined T to be the singleton set $\{K\}$ and $\rho_n(K) = \infty$.

C2 may be proven by considering each of the clauses defining $\vdash$ in turn. For example, if we consider the rule for hashing then we need to prove that $\rho_n(m) = t \Rightarrow \rho_n(h(m)) \leq t$. From the definition of the rank of hashes we see that $\rho_n(h(m)) = \rho_n(m)$ so the implication holds.

C4 must be proven for all $USER_a$ processes. If we proceed by a case analysis we find we must show that each of the following processes hold the rank:

1. $User^I(a, b, pk, na)$ for $a \neq A$
2. $User^I(A, b, pk, na)$ for $b \neq B$
3. $User^I(A, B, pk, na)$
4. $User^R(b, a, k)$ for $b \neq B$
5. $User^R(B, a, k)$ for $a \neq A$
6. $User^R(B, A, k)$

The first five cases are straightforward to prove. The final case, $User^R(B, A, k)$, is more interesting:

$$\begin{aligned} &User^R(B, A, k) = \\ &\quad rec.B.A?(pk \cdot na \cdot \{pk \cdot B\}_{Sig(A)}) \\ &\quad \rightarrow signal.B.A.k \\ &\quad \rightarrow trans.B.A.(\{k\}_{pk} \cdot \{h(k) \cdot A \cdot na\}_{Sig(A)}) \\ &\quad \rightarrow Stop \end{aligned}$$

According to the definition of **C4** we are required to show that, if $\rho(pk \cdot na \cdot \{pk \cdot b\}_{Sig(A)}) \neq \infty$ then $\rho(\{k\}_{pk} \cdot \{h(k) \cdot A \cdot na\}_{Sig(A)}) \neq \infty$.

From the rank function we can immediately deduce that $\rho(\{h(k) \cdot A \cdot na\}_{Sig(A)}) = 0$. By the rank of concatenation it remains to show that $\rho(\{k\}_{pk}) \neq \infty$. By assumption $\rho(pk \cdot na \cdot \{pk \cdot B\}_{Sig(A)}) \neq \infty$. An examination of the rank function shows that there are two cases to consider, either (i) $\rho(\{pk \cdot B\}_{Sig(A)}) = 0$ or (ii) $\rho(\{pk \cdot B\}_{Sig(A)}) = n$.

(i) If $\rho(\{pk \cdot B\}_{Sig(A)}) = 0$, then we can deduce that $pk \in \mathsf{PK}_{AB}$ which results in $\rho(\{k\}_{pk}) = 0$.

(ii) If $\rho(\{pk \cdot B\}_{Sig(A)}) = n$ the rank function tells us that $pk \notin \mathsf{PK}_{AB}$ and that $Sig(A)$ has been compromised. Assumption 1 then enables us to deduce that $leak.Sig(A)$ has occurred. However, since:

$$NET' \textbf{ sat } \{signal.A.B.K\} \textbf{ St\_precedes } \{leak.Sig(A)\}$$

K must already have been established as a session-key and therefore $k \neq K$ in $\{k\}_{pk}$. Finally, the rank function tells us that $k \neq K \Rightarrow \rho(\{k\}_{pk}) = 0$, and so the condition holds.

Finally, it is clear that **C5** holds, since only messages from TS can appear on the channel $leak$, $TS = \{Sig(A)\}$ and $\rho(Sig(A)) = n$.

Since **C1**–**C5** hold of ρ we can conclude that NET' guarantees the secrecy of K and, by implication, we can conclude that the forward secrecy goal:

$$NET \textbf{ sat } \mathbf{fs}(K, Sig(A))$$

is also met. Establishing this result only allows us to conclude partial forward secrecy with respect to $Sig(A)$. To establish full forward secrecy we would also need to consider the compromise of B's long-term key. By taking the pessimistic view — that both keys become available as soon as the first is compromised — such a proof is straightforward to construct.

4. A further example: the Cliques protocol

The Cliques protocols [1] provide key distribution services for dynamic groups. We consider a particular protocol from the Cliques suite, A-GDH.2, which is used to establish a Diffie-Hellman key between a group of principals of arbitrary size, such that each principal contributes to the key. The protocol consists of two stages. In the first (upflow) stage the group is traversed by a message that collects the contributions made by each principal. The final group member to receive this message is designated as the *controller*, who carries out the second stage by broadcasting the collected keying material to each member of the group.

For a group of size s, each principal u_i ($i \in [1, s]$) chooses an ephemeral Diffie-Hellman value x_i and maintains a long-term key K_{is}^{-1} whose inverse, K_{is}, is known to the controller u_s. A suitable prime-order subgroup of $\mathbb{Z}_p^*$ and generator g of $\mathcal{G}$ have been (publicly) agreed upon:

Upflow: round $i \mid i \in [1, s-1]$

$$u_i \longrightarrow u_{i+1} : \{g^{\frac{x_1 \ldots x_i}{x_k}} \mid k \in [1, i], g^{x_1 \ldots x_i}\}$$

Broadcast: round s

$$u_s \longrightarrow u_{i|i \in [1,s-1]} : g^{\frac{x_1 \ldots x_s}{x_i} \cdot K_{is}}$$

Upon receipt of the above, each u_i computes

$$g^{(\frac{x_1 \ldots x_s}{x_i} \cdot K_{is}) \cdot K_{is}^{-1} \cdot x_i} = g^{x_1 \ldots x_s} = K$$

Since the long-term keys cancel, the commutativity of exponentiation means that K is shared between all of the principals. We can use a temporal rank function to reason about the forward secrecy of this protocol.

We take an abstract view of the Diffie-Hellman operation in which every value is an exponentiation of the generator g by some product of random numbers and long-term keys. We write $DH(x, y)$ to represent the exponentiation x^y mod p and write $DH(g, x \cdot y)$ as a shorthand for $DH(DH(g, x), y)$. We make use of an ingredients function ι that returns the multiset of components that make up a message. For example, $\iota(DH(g, x \cdot x \cdot z)) = [\![x, x, z]\!]$. We also extend the model of the intruder to enable him to perform exponentiation and to compute the multiplicative inverse of a value:

$$\text{DH}\ \frac{S \vdash d \qquad S \vdash d_0}{S \vdash DH(d, d_0)} \qquad \text{INVERSE}\ \frac{S \vdash d}{S \vdash d^{-1}}$$

Since computation takes place in an abelian group we must also address the algebraic properties of such computation. In particular, we need to take into account the commutativity of exponentiation (since $g^{xy} = g^{yx}$) and the cancellation of multiplicative inverses (since $g^{xyx^{-1}} = g^y$). Previous work has captured the equivalence of such messages using a term rewriting system [5]. Here it will suffice to introduce three algebraic equivalences onto the message space:

$$\begin{aligned} DH(x, y \cdot z) &= DH(x, z \cdot y) \\ DH(x, y \cdot 1) &= DH(x, y) \\ DH(x, X \cdot w \cdot Y \cdot w^{-1} \cdot Z) &= DH(x, X \cdot Y \cdot Z) \end{aligned}$$

(where any of X, Y and Z may be the identity element 1). These equations result in a further proof obligation on the rank function; namely, that whenever two Diffie-Hellman messages are equivalent (according to the above equations) we assign them the same rank.

We consider (partial) forward secrecy from the point of view of a principal $u_1 = A$ who contributes $x_1 = x_A$ to compute the key K in a run of the protocol in which $u_s = C$ is the group controller and $K_{1s} = K_{AC}$ is the long-term key. As before, the protocol is modelled by a CSP process *NET*, and the security condition is:

$$NET \textbf{ sat } \mathbf{fs}(K, K_{AC})$$

where the process representing C performs the signal event for A on receipt of the upflow message, and $leak.K_{AC}$ signals that K_{AC} has been compromised. In this instance we adapt our intruder process *ENEMY* so that no key other than K_{AC} can be leaked.

As before, we consider the restricted process NET' which satisfies the condition that, in any trace of NET', an occurrence of $leak.K_{AC}$ is preceded by $signal.A.C.K$. We define $n = \tau(K_{AC}, tr)$ as the time at which the key K_{AC} is compromised. Our goal is to construct a family of rank functions, parameterised by n, that characterises the messages that can occur before, at, and after time n.

We assign a rank of n to the compromised long-term key K_{AC}:

$$\rho_n(k) = \begin{cases} n & \text{if } k = K_{AC} \\ \infty & \text{if } k = K_{uC} \wedge u \neq E \\ 0 & \text{otherwise} \end{cases}$$

Assigning a rank to Diffie-Hellman values is non-trivial, but we can make use of the fact that the group controller, C, only ever transmits messages to which he has added both a long-term secret and an ephemeral secret. Other principals will only transmit messages to which they have added an ephemeral secret. Let $cs(d)$ be the number of C's ephemeral secrets and $lt(d)$ the number of long-term secrets in a Diffie-Hellman value d. Formally, if LK is the set of all long-term secrets shared by honest principals and S^C is the set of ephemeral secrets known to the controller C, define:

$$\begin{aligned} cs(d) &= \#(\mathsf{S}^C \lhd \iota(d)) \\ lt(d) &= \#(\mathsf{LK} \lhd \iota(d)) \end{aligned}$$

(where $U \lhd R$ is the relation R restricted to domain U). Using this, we define the rank of a Diffie-Hellman value d as:

$$\rho(d) = \begin{cases} 0 & \text{if } (cs(d) + lt(d)) \bmod 2 = 0 \\ \infty & \text{otherwise} \end{cases}$$

Unfortunately, we find that this function does not satisfy condition **C2** of Theorem 3: the intruder is able to produce a message of rank $> n$ from messages with rank $\leq n$. Consider the case where the intruder uses the message $DH(g, x_A)$ (rank 0) and the key K_{AC} (rank n) to produce the message $DH(g, x_A \cdot K_{AC})$. Since $DH(g, x_A \cdot K_{AC})$ contains one long-term secret and none of the controller's ephemeral secrets it receives a rank of ∞ and so **C2** fails.

Generally speaking, we cannot conclude protocol insecurity from the failure of a rank function since a different rank function may exist which is sufficient to prove protocol correctness. In this case, however, the failure leads us to deduce the following attack (described for a group of size 3):

$$\begin{array}{lll} 1. & A \rightarrow B & : \{g, g^{x_A}\} \\ 2. & B \rightarrow C & : \{g^{x_A}, g^{x_B}, g^{x_A x_B}\} \\ 3a. & C \rightarrow E(A) & : g^{x_B x_C K_{AC}} \\ 3a' & E(C) \rightarrow A & : g^{x_B} \\ 3b. & C \rightarrow B & : g^{x_A x_C K_{BC}} \end{array}$$

In the attack, the intruder observes the first two protocol messages (noting the value $g^{x_A x_B}$ in message 2), intercepts A's component of the broadcast message from C ($3a$) and replaces it with g^{x_B}($3a'$). On receipt of the broadcast, A computes the key $g^{x_B K_{AC}^{-1} x_A}$. At a later stage K_{AC} becomes available to the intruder who then deduces K_{AC}^{-1} and uses the value $g^{x_A x_B}$ (remembered from earlier) to compute the key $g^{x_A x_B K_{AC}^{-1}}$. The intruder now shares a key with A, marking the failure of forward secrecy.

This attack is a simpler version of one proposed by Pereira and Quisquater [20] who demonstrated attacks on each of the main security goals of the Cliques protocols. The same authors have more recently shown that any protocols built using the rationale of Cliques are irreparably flawed, at least in situations where the group contains four or more principals [21].

5. Discussion

As promised, we now discuss why the standard (two-valued) rank function approach is incapable of verifying forward secrecy properties.

In the original rank function approach [23] positive ranks were used to denote public messages and non-positive ranks were assigned to secret messages. It was soon discovered [13] that this is equivalent to using just two ranks, pub for public and sec for secret messages[2]. This binary view of secrecy allows us to make all-or-nothing assertions about the

2 Previous work has tended to use the ranks 0 and 1 to denote secret and public messages. For the temporal ranks described above we use 0 for a different purpose, namely, to denote messages available to the intruder at the start of a protocol. We use the emotive ranks pub and sec in this discussion to avoid confusion.

secrecy of messages: a message m is either completely secret ($\rho(m) = \mathsf{sec}$) or completely public ($\rho(m) = \mathsf{pub}$). In fact, it turns out that setting $\rho(m) = \mathsf{pub}$ is equivalent to saying that it should be safe for the intruder to know m at the start of the protocol. This can be stated another way. Consider some protocol P, and a (two-valued) rank function ρ that is sufficient to prove that P meets some secrecy goal with respect to the intruder's initial knowledge: IIK^0. If there exists a message $m \notin IIK^0$ with $\rho(m) = \mathsf{pub}$ then ρ is also sufficient to show that P meets the same secrecy goal with respect to the initial knowledge $IIK^1 = IIK^0 \cup \{m\}$. Recall that, for a security goal expressed as a trace specification, the existence of a rank function allows us to conclude that the protocol meets the security goal.

Theorem 4 *Let ρ be a rank function with respect to the intruder's initial knowledge IIK^0. If there exists some $m \notin IIK^0$ satisfying $\rho(m) = \mathsf{pub}$ then ρ is a rank function with respect to $IIK^1 = IIK^0 \cup \{m\}$.*

Proof. The two-valued central rank theorem [23] gives four healthiness conditions that a rank function must satisfy. It is straightforward to show that, for each of these, if ρ satisfies the condition w.r.t IIK^0 then ρ must also satisfy the condition w.r.t IIK^1. □

This goes against intuition. A message m may become available to the intruder at some point in a protocol run, but this does certainly not mean that the message should be known to the intruder initially. In fact, Heather [12] has given an example of a (contrived) protocol that contains a *temporary secret* — a value that must remain secret for a protocol to reach its secrecy goal but may then be safely leaked — and has shown that such protocols, even if correct, are unverifiable using two-valued rank functions. Arguing pragmatically, this incompleteness is only of theoretical interest unless it can be shown to hamper our ability to reason about real (i.e., uncontrived) protocols. However, since compromised keys are temporary secrets, we are faced with the conclusion that forward secrecy properties are unverifiable using the standard approach.

We can illustrate this with an example. Consider an instance of the Boyd protocol, modified so that A makes $Sig(A)$ public after the key exchange has taken place:

$$\begin{array}{llllll} 1. & A & \rightarrow & B & : & PK \cdot NA \cdot \{PK \cdot B\}_{Sig(A)} \\ 2. & B & \rightarrow & A & : & \{K\}_{PK} \cdot \{h(K) \cdot A \cdot NA\}_{Sig(B)} \\ 3. & A & \rightarrow & B & : & Sig(A) \end{array}$$

The results of the previous section allow us to conclude that a run of the protocol should provide forward secrecy; that, even after $Sig(A)$ is leaked, the session-key K should remain secret. To see why a two-valued rank function cannot be used to prove this property we can attempt to create one. We assume that the usernames of principals are public; in particular, $\rho(A) = \mathsf{pub}$. Since $Sig(A)$ is sent in the clear we are forced to set $\rho(Sig(A)) = \mathsf{pub}$. For any ephemeral key-pair (PK', SK') invented by the intruder we have $\rho(PK') = \rho(SK') = \mathsf{pub}$, and since the intruder can use these facts to construct the message $\{PK' \cdot B\}_{Sig(A)}$ we are compelled to set $\rho(\{PK' \cdot B\}_{Sig(A)}) = \mathsf{pub}$. Now, if the intruder, masquerading as A, sends this message to B, B will respond with $\{K\}_{PK'}$ and the intruder can use SK' to deduce the session-key K. We are therefore forced to set $\rho(K) = \mathsf{pub}$, whereas our hope was for K to remain secret. Some thought leads us to conclude that this attack will not work since $Sig(A)$ is not made public until *after* K has been accepted by A. However, Theorem 4 tells us that rank functions cannot distinguish a message that is made public at the end of a protocol from a message that is public at the start of the protocol. Therefore, using a two-valued rank function means that setting $\rho(Sig(A)) = \mathsf{pub}$ is the same as saying that $Sig(A) \in IIK$. If $Sig(A)$ is known to the intruder at the start then the protocol clearly cannot guarantee the secrecy of K.

A rank function is an abstraction technique that allows us to prove trace specifications of CSP processes without requiring us to consider the traces directly. A trace records one possible history of a process. In the above example, if the event $trans.A.B.Sig(A)$ appears in a trace, it will always be preceded by the event $rec.A.B.\{K\}_{PK'} \cdot \ldots$, representing the fact that A accepts the session-key before $Sig(A)$ is leaked. Thus, a trace does not simply convey information about which messages appear, but also when they appear. A two-valued rank function abstracts away all of the temporal information present in the trace and just tells us what messages can occur. However, for protocols containing temporary secrets, this temporal information is vital if we are to prove their correctness. In such cases two-valued rank functions abstract away too much.

Temporal ranks overcome this problem by lifting state information from the trace into the rank function. A temporal rank function for the example above would say that the password is leaked at some time n and set $\rho(Sig(A)) = n$. The ranks of other messages pivot around n in the sense that a message made public before n is assigned some rank $0 \leq r < n$ and a message available at or after n is assigned a rank in the range $n \leq r \leq \infty$. Crucially, the offending message $\{PK' \cdot B\}_{Sig(A)}$ would be assigned a rank of n and the message $\{K\}_{PK}$ would be assigned a rank less than n, representing the fact that $A's$ reception of the session-key precedes the compromise of the password. At time n, A is therefore unwilling to accept the intruder's faked message and so the forward secrecy property holds.

6. Conclusion

In this paper we have shown how forward secrecy properties of crypto-protocols can be expressed and reasoned about using rank functions. We have introduced the concept

of a temporal rank function that not only allows us to reason about *what* messages an attacker can deduce, but *when* in the protocol he can deduce them. We have applied the approach to two examples, establishing a proof of correctness in one case and rediscovering an attack in the other. Both of these examples considered a protocol running on an unbounded network that allows principals to engage in arbitrarily many protocol runs concurrently. We also described the motivation for using temporal ranks by demonstrating that some protocols may be unverifiable when using techniques that classify values as simply *public* or *secret*.

The present work can be seen as contributing to a growing body of research that aims to move beyond the familiar realms of perfect encryption, authentication, and secrecy, by relaxing the restrictions on the intruder and considering the wider class of security properties that become relevant as a result.

It is interesting to note that all of the rank functions given in this paper are three valued: 0, n and ∞ and, in fact, it would have been sufficient to fix n at some arbitrary finite, non-zero, value. In general, a temporal rank function will be three-valued if all compromised values are assumed to be leaked at the same time. In the worst case, a temporal rank function with k separate leak points will have $k + 2$ distinct ranks. It is not clear whether, in practice, more than three ranks will ever be required. If three ranks are indeed sufficient, then the theory presented above should almost certainly be refined to reflect this fact. In seeking to present a general theory we have had to introduce some concepts (such as the idea of a family of functions) which would be redundant in a three-rank theory.

A temporal rank function can be viewed as establishing a hierarchy of secrecy, where a message m_1 is *less secret* than a message m_2 if $\rho(m_1) < \rho(m_2)$. One message is therefore more secret than another if it must remain secret for longer. Such a fine-grained notion of secrecy has proven useful in the present work, and it would be interesting to consider whether temporal ranks can be applied in a more general setting.

At a higher level of abstraction, our treatment of forward secrecy allows us to form a rather natural hierarchy of secrecy specifications, since, in our model, secrecy is implied by the presence of partial forward secrecy and, in turn, partial forward secrecy is implied by the presence of full forward secrecy. This invites us to verify security protocols in a similarly hierarchical way; either by working upwards from secrecy or downwards from some stronger condition. The top-down approach (starting with, say, full forward secrecy) would tend to involve a more difficult proof but would have the pleasing side-effect of guaranteeing all weaker secrecy goals by implication. Developing useful notions of secrecy that extend this hierarchy, in either direction, seems an interesting research challenge.

7. Acknowledgements

Thanks to Colin Boyd for responding to our questions and to Gavin Lowe, whose critique of our earlier approach [6] prompted the present work. Martin Green, Neil Evans and the programme committees of WITS'05 and CSFW-18 made many valuable comments on various incarnations of this paper.

References

[1] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *Proceedings of the 5th ACM Conference on Computer and Communication Security*. ACM Press, 2000.

[2] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1992.

[3] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[4] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions (Ext. abstract). In *Advances in Cryptology: Proceedings of EUROCRYPT '02*, volume 2332 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[5] R. Delicata and S. A. Schneider. A formal model of Diffie–Hellman using CSP and rank functions. Technical Report CSD-TR-03-05, Department of Computer Science, Royal Holloway, University of London, 2003.

[6] R. Delicata and S. A. Schneider. Towards the rank function verification of protocols that use temporary secrets. In *Proceedings of the Workshop on Issues in the Theory of Security: WITS '04*, 2004.

[7] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2), 1992.

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

[9] N. Evans. Investigating security through proof. *Ph.D Thesis, Royal Holloway, University of London*, 2003.

[10] C. G. Günther. An identity-based key-exchange protocol. In *Advances in Cryptology: Proceedings of EUROCRYPT '89*, volume 0434 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[11] J. D. Guttman. Key compromise, strand spaces, and the authentication tests. In *Proceedings of Mathematical Foundations of Programming Semantics: MFPS 17*, volume 47 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2001.

[12] J. A. Heather. Strand spaces and rank functions: More than distant cousins. In *Proceedings of The 15th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2002.

[13] J. A. Heather and S. A. Schneider. Towards automatic verification of security protocols on an unbounded network. In *Proceedings of The 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[14] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Advances in Cryptology: Proceedings of ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[15] H. Krawczyk. SIGMA: The 'SIGn and MAc' approach to authenticated Diffie–Hellman and its use in the IKE protocols. In *Advances in Cryptology: Proceedings of CRYPTO '03*, volume 2729 of *Lectures Notes in Computer Science*. Springer-Verlag, 1996.

[16] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *Symposium on Network and Distributed System Security*. IEEE Computer Society Press, 1996.

[17] A. J. Menezes, M. Qu, and S. A. Vanstone. Some new key agreement protocols providing mutual implicit authentication. In *Workshop on Selected Areas in Cryptography: SAC '95*, 1995.

[18] D. Park, C. Boyd, and S.-J. Moon. Forward secrecy and its application to future mobile communications security. In *Proceedings of Public Key Cryptography: Third International Workshop on Practice and Theory in Public Key Cryptosystems: PKC 2000*, volume 1751 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[19] O. Pereira. Modelling and security analysis of authenticated group key agreement protocols. *Ph.D Thesis, Université Catholique de Louvain*, 2003.

[20] O. Pereira and J.-J. Quisquater. Security analysis of the Cliques protocols suites. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.

[21] O. Pereira and J.-J. Quisquater. Generic insecurity of Cliques-type authenticated group key agreement protocols. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2004.

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

[23] S. A. Schneider. Verifying authentication protocols with CSP. In *Proceedings of The 10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[24] S. A. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. John Wiley and Sons, Ltd, 2000.

[25] S. A. Schneider. Verifying authentication protocol implementations. In *Proceedings of the 5th international conference on Formal Methods for Open Object-Based Distributed Systems*, 2002.

[26] M. J. Wiener. Performance comparison of public-key cryptosystems. *Cryptobytes*, 4(1), 1998.

A. CSP

CSP is a notation for describing concurrent systems in terms of *processes* which perform *events*. Communication is achieved by *prefixing*. The communication $a \rightarrow P$ is the process that performs the event a and then behaves like P. Events may be structured in terms of channels where the event $c.v$ denotes the communication of value v on channel c. Compound events can be used to model input and output where, by convention, input is represented using '?' and output using '!': so, for example, $c?v : T \rightarrow P(v)$ is the process that inputs on channel c any value v of type T and then behaves as $P(v)$. The concept of choice is modelled with the $\Box$ operator such that the process $P = a \rightarrow P \;\Box\; b \rightarrow P$ is prepared to perform either a or b, and the choice is determined by the environment. The generalised choice operator $\Box_{i \in I} P_i$ allows the environment to choose between a family of processes $\{P_i \mid i \in I\}$. Conditional choice can be written in the form — if G then P else Q —- where G is a boolean guard.

Processes may be composed in parallel to perform handshaking synchronisation. The process $P \,|[\, X \,]|\, Q$ forces the processes P and Q to agree on all events in the set X. When an event $x \in X$ is performed by $P \,|[\, X \,]|\, Q$, *both* processes perform the event. Events outside of X may proceed independently. The communication set X will often be written in the form $\{|\; x_1, x_2 \;|\}$; the $\{|\;\;|\}$ syntax denotes the closure of events on channels x_1 and x_2. An interleaving of two processes $P \;|||\; Q$ allows P and Q to proceed independently of one another, without synchronising on any events. If an event is performed that both P and Q are willing to communicate, only one process actually performs the event, and is chosen non-deterministically. The generalised interleaving operator $\left|||_{i \in I} P_i\right.$ models the interleaving of a family of processes $\{P_i \mid i \in I\}$. Σ is the set of all events. Two particularly important processes are *Stop* and RUN_A. *Stop* is the process that does nothing at all. RUN_A:

$$RUN_A = ?x : A \rightarrow RUN_A$$

is the process which, for a set of events $A \subseteq \Sigma$, is always willing to communicate any member of A that the environment desires.

Several semantic models of CSP exist, and in this paper we make use of the simplest: the traces model. The traces model describes each process P in terms of its observable events, $traces(P)$, the set of all possible sequences of events that P can perform. $traces(P)$ is always non-empty, since every process can perform the empty trace $\langle\rangle$, and $traces(P)$ is prefix closed, meaning that if $s \frown t$ is a trace, then so is s. For example, $traces(a \rightarrow b \rightarrow Stop) = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle\}$. We write $\#tr$ to mean the length of a trace tr. Each CSP construct has an associated rule for the calculation of traces. The traces of a process define the possible behaviour of that process and are sufficient for reasoning about safety properties of a system.

If an event e occurs in a trace tr_0 we write e in tr_0. Conversely, if e does not occur in tr_0 we write $\neg(e \text{ in } tr_0)$.

$tr_0 \upharpoonright T$ is the sequence tr_0 restricted to T: the sequence whose members are those of tr_0 which are in T. For example, if $tr_0 = \langle a, b, a, c\rangle$ then $tr_0 \upharpoonright \{a, c\} = \langle a, a, c\rangle$. If C is a set of channels then $tr_0 \Downarrow C$ is the sequence of values that have been communicated along channels in C in tr_0. If $tr_0 = \langle input.a, output.b, input.c\rangle$ then $tr_0 \Downarrow \{output\} = \langle b\rangle$. When C is understood as the singleton set $\{c\}$ we may omit the braces and write $tr_0 \Downarrow c$ in preference to $tr_0 \Downarrow \{c\}$.

A modern presentation of the CSP language and its semantic models can be found in [22, 24].

B. Proof of Theorem 3

To prove that NET' **sat secret** T we need to show that:

$$\forall\, tr \in traces(NET').\textbf{secret}\ T(tr)$$

We prove, for an arbitrary trace tr_0, that the existence of a rank function ρ_n is sufficient to conclude **secret** $T(tr)$ when $\tau(k, tr) = n$.

For a contradiction assume that **C1**–**C5** hold, but also that $\neg(\textbf{secret}\ T(tr_0))$. Then there exists an occurrence in tr_0 of some message $t \in T$. Since **C3** tells us that $\rho_n(t) = \infty$ for any $t \in T$ we have that there are some messages with a rank of ∞. Let tr_1 be the prefix of tr_0 whose last message is the first message of tr_0 with rank ∞. The sequence tr_1 is the trace up to the point where the first rank ∞ message occurs.

The prefix-closure of traces in processes tells us that $tr_1 \in traces(NET')$. The last message of tr_1 can take one of three forms: *leak.m*, *trans.u.v.m*, or *rec.u.v.m* for some *u*, *v* and *m*, where $\rho_n(m) = \infty$.

Case *leak.m***:** from **C5** we have that $\rho_n(leak.m) < \infty$, forcing a contradiction.

Case *rec.u.v.m*: We have that tr_1 is a trace of *ENEMY* and, by Theorem 1, that $(IIK \cup (tr \Downarrow \{trans, leak\})) \vdash tr \Downarrow rec$ and so $(IIK \cup (tr_1 \Downarrow \{trans, leak\})) \vdash m$. But, by definition of tr_1 we have that $\rho_n(tr_1 \Downarrow \{trans, leak\}) \neq \infty$ since all messages in tr_1 apart from the last have a finite rank. Therefore **C1** and **C2** yield that $\rho_n(m) \neq \infty$, forcing a contradiction.

Case *trans.u.v.m*: Let $tr_u = tr_1 \upharpoonright \{trans.u, rec.u\}$. This is the portion of tr_1 in which $USER_u$ participates, so $tr_u \in traces(USER_u)$. Hence, by **C4** we have that **holds** $\rho_n(tr_u)$. Expanding the definition we have that:

$$\rho_n(tr_u \Downarrow rec) \neq \infty \Rightarrow \rho_n(tr_u \Downarrow trans) \neq \infty$$

from which it follows that $\rho_n(tr_u \Downarrow rec) \neq \infty \Rightarrow \rho_n(m) \neq \infty$. However, by definition of tr_0 and hence tr_u we have that $\rho_n(m) = \infty$, forcing a contradiction. In either case we find a contradiction, which establishes the theorem. □