Justifying Finite Resources for Adversaries in Automated Analysis of Authentication Protocols

Scott D. Stoller*
Computer Science Dept., Indiana University, Bloomington, IN 47405 USA

February 13, 1999

Abstract

Authentication protocols (including protocols that provide key establishment) are designed to work correctly in the presence of an adversary that can (1) perform an unbounded number of encryptions (and other operations) while fabricating messages, and (2) prompt honest principals to engage in an unbounded number of concurrent runs of the protocol. The amount of local state maintained by a single run of an authentication protocol is bounded. Intuitively, this suggests that there is a bound on the resources needed to attack the protocol. Such bounds clarify the nature of attacks on these protocols. They also provide a rigorous basis for automated verification of authentication protocols. However, few such bounds are known. This paper defines a language for authentication protocols and establishes two bounds on the resources needed to attack protocols expressible in that language: an upper bound on the worst-case number of encryptions by the adversary, and an exponential lower bound on the worst-case number of concurrent runs of the protocol. The upper bound on encryptions is relative to an upper bound on the number of runs; on-going work on proving such a bound is briefly described.

1 Introduction

Many protocols are intended to work correctly in the presence of an adversary that can (1) perform an unbounded number of encryptions (and other operations) while fabricating messages, and (2) prompt honest principals to engage in an unbounded number of concurrent runs of the protocol. This includes some protocols for Byzantine Agreement [GLR95], secure reliable multicast [Rei96, MR97], authentication, and electronic payment [OPT97]. In this paper, we focus on protocols for authentication, including key establishment. Such protocols play a fundamental role in many distributed systems, and their correctness is essential to the correctness of those systems. Informally, authentication protocols should satisfy (at least) two kinds of correctness requirements: *secrecy*, *i.e.*, certain values (such as cryptographic keys) are not obtained by the adversary, and *correspondence*, *i.e.*, a principal's conclusion about the identity of a principal with whom it is communicating is never incorrect.

The amount of local state maintained by a single run of an authentication protocol is bounded. Intuitively, this suggests that there is a bound on the resources needed to attack the protocol. Such bounds provide insight into the possible kinds of attacks on these protocols. They also provide a rigorous basis for automated verification of authentication protocols. Authentication protocols are short and look deceptively simple, but numerous flawed or weak protocols have been published; some examples are described in [DS81, BAN90, WL94, AN95, AN96, Low96, Aba97, LR97, FHG98]. This attests to the importance of rigorous verification of such protocols. Theorem proving requires considerable expertise from the user. Systematic state-space exploration, including temporal-logic model checking and process-algebraic equivalence checking, is emerging as a practical approach to automated verification [CES86, Hol91, DDHY92, Kur94, CS96].

*Email: stoller@cs.indiana.edu. Web: http://www.cs.indiana.edu/~stoller/ .

Systems containing adversaries of the kind described above have an unbounded number of reachable states, so state-space exploration is not directly possible. The case studies in [MCF87, Ros95, HTWW96, DK97, LR97, MMS97, MCJ97, MSS98, Bol98] offer strong evidence that state-space exploration of authentication protocols and other similar kinds of protocols is feasible when small upper bounds are imposed on the size of messages and the number of runs. However, the bounds used in most of those case studies have not yet been rigorously justified. Reduction theorems are needed, which show that if a protocol is correct in a system with certain finite bounds on these parameters, then the protocol is correct in the unbounded system as well. This paper defines languages for authentication protocols and requirements and establishes two bounds on the resources needed to attack protocols expressible in that language: an upper bound on the worst-case number of encryptions by the adversary, and a lower bound on the worst-case number of concurrent runs of the protocol. The upper bound on encryptions is relative to an upper bound on the number of runs; we sketch an approach to proving such a bound.

Few other reductions of this kind are known. Authentication protocols are a good first target for such reductions, because they have extremely simple control flow and a small repertoire of operations. Dolev and Yao developed impressive analysis algorithms that directly verify secrecy requirements [DY83]; however, their algorithms do not consider correspondence properties and apply to a rather limited class of protocols, which excludes several well-known protocols, such as the Otway-Rees and Yahalom protocols [BAN90], and is strictly included in the class of protocols we consider. Roscoe [Ros98] has done some interesting preliminary work on using data-independence techniques to prove reductions for authentication protocols; this has not yet led to specific reductions (*i.e.*, specific bounds). Lowe proved specific bounds for a corrected version of the Needham-Schroeder public-key protocol [Low96] and subsequently generalized that proof to show for a class of protocols that no violations of secrecy properties are missed when small bounds are used [Low98a, Low98b]. However, that result does not extend to correspondence requirements [Low98a, p. 61] and does not consider known-key attacks, and the class of protocols considered [Low98a, Sections 2.1 and 2.2] excludes some well-known protocols, such as the Otway-Rees [OR87], Yahalom [BAN90], and Kerberos (V5) [KN93] protocols.

Section 2 defines our languages for authentication protocols and requirements. These languages are based closely on Woo and Lam's model of authentication [WL93a, WL93b]; that model also underlies [MCJ97, CMJ98]. Section 3 characterizes encryptions by the adversary that are not useful, *i.e.*, do not affect the behavior of the other principals in an execution. Section 4 shows that in executions in which the adversary performs many nested encryptions, some of those encryptions must be useless. Section 5 presents upper bounds on the number of operations by the adversary, specifically, on the number of encryptions and concatenation (pairing) operations. The bound on encryptions assumes a bound on the number of protocol runs. Section 6 presents an exponential lower bound on the number of runs.

There are two main directions for future work. The first is to broaden the scope of our results by considering hash functions, timestamps, recency requirements, and key confirmation requirements [MvOV97, Ch. 12]. The second is to prove an upper bound on the number of runs. This is closely related to techniques for automated analysis of systems with unbounded numbers of similar processes, such as [CGJ95, KM95, EN96, AJ98], but those techniques are neither aimed at nor applicable to authentication protocols. Proving an exponential upper bound is of some theoretical interest but would be of little value for automated verification. Thus, the lower bound in this paper implies that additional restrictions on the class of protocols are needed in order to obtain a useful upper bound. Finding syntactic restrictions that do not exclude interesting protocols seems difficult, so we are studying the use of "dynamic" restrictions, which (like the correctness requirements) would be checked by state-space exploration of bounded systems. The main idea is to require that in executions involving at most two protocol runs, those two runs interact in limited ways, and to prove that in arbitrary executions, runs still interact only in those same limited ways.

2 Model of Authentication Protocols

Our model of authentication is based closely on Woo and Lam's model [WL93a]. We call the language LAP (Language for Authentication Protocols).

Woo and Lam's model incorporates some common simplifying assumptions. It assumes that the symmetric and public-key cryptosystems are perfect (or ideal) [WL93a]; a reasonable approximation to this can be obtained by incorporating into the encryption and decryption functions an integrity check based on a message digest. It also assumes that messages contain explicit formatting information sufficient for the recipient to correctly divide messages into fields; a header specifying the starting offset of each field suffices.

2.1 Syntax of Authentication Protocols

Let Con be a set of constants; this includes symbols representing nonces, keys, and names (of principals). Let $Name \subset Con$ be the set of names (of principals), including a distinguished name Z for the adversary. The set Op of operators is $Op = \{encrypt, pair, key, pubkey, pvtkey\}$. The term $encrypt(t_1, t_2)$ represents t_1 encrypted with t_2 and is usually written as $\{t_1\}_{t_2}$. The term $pair(t_1, t_2)$ represents t_1 paired with t_2 and is usually written as $t_1 \cdot t_2$; similarly, $pair(t_1, pair(t_2, t_3))$ is usually written as $t_1 \cdot t_2 \cdot t_3$; and so on. For $x_1 \in Name$ and $x_2 \in Name$, the term $key(x_1, x_2)$ represents a symmetric key shared by principals x_1 and x_2, and the terms $pubkey(x_1)$ and $pvtkey(x_1)$ represent x_1's public and private keys respectively. Let Key_{sym} denote the set containing constants representing symmetric keys and terms of the form $key(x_1, x_2)$. Let Key_{asym} denote the set containing terms of the form $pubkey(x)$ and $pvtkey(x)$. Distinct elements of $Key_{sym} \cup Key_{asym}$ are assumed to represent distinct keys. Let Var be a set of variables. A *term* is an expression composed of constants, variables, and operators. A *ground term* is a term not containing variables. Let $Term$ and $Term_G$ denote the sets of terms and ground terms, respectively.

The kinds of statements are:

BeginInit: "Begin initiator protocol". This and the following 3 statements are included to facilitate expression of correspondence requirements (see Section 2.3).

BeginRespond: "Begin responder protocol".

EndInit: "Successful completion of initiator protocol

EndRespond: "Successful completion of responder protocol".

NewValue(ns, v): This generates an unguessable value t (*e.g.*, a nonce or session key) and binds variable v to t. A value generated by NewValue is called a *genval*. Informally, if ns is non-empty, then t is a secret intended to be shared only by the principals named in ns; if ns is empty, then t is not intended to be kept secret. The genval t becomes *old* when every principal (ignoring Z) in the set ns has executed Old(t); thus, if ns is empty, t is old as soon as it is generated. When a generated value becomes old, it is automatically revealed to Z; this models known-key attacks. The principal executing the NewValue statement is not necessarily included in ns; for example, a server S might generate a session key for A and B by executing NewValue($\{A, B\}, v$), because by the usual definition of known-key attacks, this key is considered vulnerable as soon as A and B have accepted it as a session key.

Send(x, t): This sends a message t to x. The message might not reach x; the adversary can intercept it.

Receive(t): This receives a message t' and binds the unbound variables in t to the corresponding subterms of t'. This statement attempts pattern-matching between a candidate message t' and the term t. If there exist bindings for the unbound variables of t such that t with those bindings equals t', then the Receive statement executes and establishes those bindings. The Receive statement blocks until this condition is satisfied. Variables bound by previous statements are not treated as pattern variables in this Receive statement; in other words, occurrences of those variables in this Receive statement are uses, not defining occurrences. Note that occurrences of the *encrypt* operator in Receive statements actually represent decryptions, not encryptions. We could extend the Receive statement with another argument indicating the expected sender of the message, but this has little benefit (mainly because Z can forge that information).

Old(t): This indicates that a principal has finished its part in set-up involving t, where t is a term not containing the *encrypt* operator. The primary motivation for introducing this statement is to facilitate modeling of known-key attacks.

A *local protocol* is a finite sequence of statements satisfying the well-formedness requirements given below. There are 3 kinds of local protocols. *Initiator (local) protocols* may contain up to one occurrence each of BeginInit and EndInit and do not contain BeginRespond or EndRespond. *Responder (local) protocols* contain up to one occurrence each of BeginRespond and EndRespond and do not contain BeginInit or EndInit. *Fixed (local) protocols* do not contain any of these four kinds of statements. When a principal x starts executing a local protocol, the variable μ is automatically bound to x, and in initiator and responder protocols, the variable p is automatically bound to an arbitrary element of $Name \setminus \{x\}$, identifying the *partner*, i.e., the principal expected to act as the responder or initiator, respectively. A *defining occurrence* of a variable v other than μ or p in a local protocol is an occurrence of v that (1) appears in the first statement containing v and (2) appears in Receive or the second argument of NewValue. There are no defining occurrences of μ and p. All non-defining occurrences of variables are called *uses*. For the reader's convenience, defining occurrences of variables are underlined in local protocols. The well-formedness requirements are: (1) Variables are bound before they are used, *i.e.*, for each variable v except μ and p, each statement containing uses of v is preceded by a statement containing defining occurrences of v. (2) Variables are single-assignment, *i.e.*, for each variable v, uses of v do not occur in the second argument of NewValue statements. (3) Keys are parameterized by names or variables, *i.e.*, for each occurrence of *key*, *pubkey*, or *pvtkey*, the arguments are in $Name \cup Var$.

A *protocol* is a pair $\langle IK, PS \rangle$, where the *initial knowledge* IK is a set of ground terms and PS is a set of pairs of the form $\langle ns, P \rangle$, where $ns \subseteq (Name \setminus \{Z\})$ and P is a local protocol. IK is the set of terms initially known to Z. A pair $\langle ns, P \rangle$ in PS means that local protocol P can be executed by any principal in ns. Note that each run of a local protocol has its own variable bindings.

Example. In LAP, the Yahalom protocol [BAN90] is

$$\langle \{key(Z, S)\}, \{\langle \{A, B\}, P_I \rangle, \langle \{A, B\}, P_R \rangle, \langle \{S\}, P_S \rangle\} \rangle, \tag{1}$$

where

P_I:

0. NewValue($\emptyset, \underline{ni}$)
1. Send($p, \mu \cdot ni$)
2. Receive($\{p \cdot \underline{k} \cdot ni \cdot \underline{nr}\}_{key(\mu,S)} \cdot \underline{x}$)
3. BeginInit
4. Send($p, x \cdot \{nr\}_k$)
5. EndInit
6. Old(k) 7. Old(nr)

P_R:

0. Receive($p \cdot \underline{ni}$)
1. NewValue($\{\mu, p\}, \underline{nr}$)
2. BeginRespond
3. Send($S, \mu \cdot \{p \cdot ni \cdot nr\}_{key(\mu,S)}$)
4. Receive($\{p \cdot \underline{k}\}_{key(\mu,S)} \cdot \{nr\}_{\underline{k}}$)
5. EndRespond
6. Old(k) 7. Old(nr)

P_S:

0. Receive($\underline{r} \cdot \{i \cdot \underline{ni} \cdot \underline{nr}\}_{key(r,\mu)}$)
1. NewValue($\{i, r\}, \underline{k}$)
2. Send($i, \{r \cdot k \cdot ni \cdot nr\}_{key(i,\mu)}$
$\cdot \{i \cdot k\}_{key(r,\mu)}$)

Variables μ and p have type Name (as always); x in P_I has type All; the remaining variables have type Prim. For comparison, in the concise but informal and sometimes ambiguous notation commonly found in the security literature, the Yahalom protocol might be written as

$$\begin{array}{ll} 1.\ A \rightarrow B: & A \cdot N_a \\ 2.\ B \rightarrow S: & B \cdot \{A \cdot N_a \cdot N_b\}_{K_{bs}} \\ 3.\ S \rightarrow A: & \{B \cdot K_{ab} \cdot N_a \cdot N_b\}_{K_{as}} \cdot \{A \cdot K_{ab}\}_{K_{bs}} \\ 4.\ A \rightarrow B: & \{A \cdot K_{ab}\}_{K_{bs}} \cdot \{N_b\}_{K_{ab}} \end{array}$$

Example. In LAP, the Otway-Rees protocol [OR87] is

$$\langle \{key(Z, S)\}, \{\langle \{A, B\}, P_I \rangle, \langle \{A, B\}, P_R \rangle, \langle \{S\}, P_S \rangle\} \rangle, \tag{2}$$

where

P_I:

0. NewValue($\emptyset, \underline{m}$)
1. NewValue($\emptyset, \underline{n}$)
2. BeginInit
3. Send(r, $m \cdot \mu \cdot p$
$\cdot \{n \cdot m \cdot \mu \cdot p\}_{key(\mu,S)}$)
4. Receive($m \cdot \{n \cdot \underline{k}\}_{key(\mu,S)}$)
5. Old(k)
6. EndInit

P_R:

0. Receive($\underline{m} \cdot p \cdot \mu \cdot \underline{x}$)
1. BeginRespond
2. NewValue($\emptyset, \underline{n}$)
3. Send($S, m \cdot p \cdot \mu \cdot x \cdot \{n \cdot m \cdot p \cdot \mu\}_{key(\mu,S)}$)
4. Receive($m \cdot \underline{y} \cdot \{n \cdot \underline{k}\}_{key(\mu,S)}$)
5. Send($p, m \cdot y$)
6. Old(k)
7. EndRespond

P_S:

0. Receive($\underline{m} \cdot \underline{i} \cdot \underline{r} \cdot \{\underline{x} \cdot \underline{m} \cdot \underline{i} \cdot \underline{r}\}_{key(\underline{i},\mu)}$
$\cdot \{\underline{y} \cdot \underline{m} \cdot \underline{i} \cdot \underline{r}\}_{key(\underline{r},\mu)}$)
1. NewValue($\{i, r\}, \underline{k}$)
2. Send($r, m \cdot \{x \cdot k\}_{key(i,\mu)}$
$\cdot \{y \cdot k\}_{key(r,\mu)}$)

Example. In LAP, the Needham-Schroeder shared-key protocol [BAN90], slightly modified, is

$$\langle \{key(Z, S)\}, \{\langle \{A, B\}, P_I \rangle, \langle \{A, B\}, P_R \rangle, \langle \{S\}, P_S \rangle\} \rangle, \tag{3}$$

where

P_I:

0. NewValue($\emptyset, \underline{ni}$)
1. Send($S, \mu \cdot p \cdot ni$)
2. Receive($\{ni \cdot p \cdot \underline{k}\}_{key(\mu,S)} \cdot \underline{x}$)
3. Send(p, x)
4. Receive($\{\underline{nr}\}_k$)
5. BeginInit
6. Send($\{nr \cdot p\}_k$)
7. EndInit
8. Old(k)

P_R:

0. Receive($\{\underline{k} \cdot p\}_{key(\mu,S)}$)
1. NewValue($\emptyset, \underline{nr}$)
2. BeginRespond
3. Send($p, \{nr\}_k$)
4. Receive($\{nr \cdot \mu\}_k$)
5. EndRespond
6. Old(k)

P_S:

0. Receive($\underline{i} \cdot \underline{r} \cdot \underline{ni}$)
1. NewValue($\{i, r\}, \underline{k}$)
2. Send($i, \{ni \cdot r \cdot k\}_{key(i,\mu)} \cdot \{k \cdot i\}_{key(r,\mu)}$)

The original Needham-Schroeder protocol is obtained by changing nr in line 6 of P_I and line 4 of P_R to $nr - 1$, and by changing line 2 of P_I to Receive($\{na \cdot p \cdot \underline{k} \cdot \underline{x}\}_{key(\mu,S)}$) and line 2 of P_S to Send($i, \{na \cdot r \cdot k \cdot \{k \cdot i\}_{key(r,\mu)}\}_{key(i,\mu)}$). A straightforward argument shows that correctness of the above protocol implies correctness of the original Needham-Schroeder protocol.

2.2 Semantics of LAP

Sequences. Sequences are represented as functions from the natural numbers or a prefix of the natural numbers to elements. Thus, the initial element of a sequence σ is $\sigma(0)$; the next element is $\sigma(1)$; and so on. The *domain* of a sequence σ is defined by $\mathrm{dom}(\sigma) = \{0, 1, \ldots, |\sigma| - 1\}$, where $|\sigma|$ is the length of σ. For $j < |\sigma|$, $\sigma(0..j)$ denotes the prefix of σ of length $j + 1$; for $j \geq |\sigma|$, $\sigma(0..j)$ denotes σ.

Run-ids, Substitutions, and Events. A *run-id* is a number identifying a particular run of a local protocol. Let ID denote the set of run-ids. For a set V of variables, let $Subst(V)$ denote the set of *bindings* for the variables in V, *i.e.*, the set of functions from V to ground terms. We use "binding" and "substitution" interchangeably. The application of a substitution θ to a term t is denoted $t[\theta]$. An *event* is a tuple $\langle id, l, s\rangle$, where id is a run-id or Z, l is a natural number or Z, and s is a statement. When $id = Z$ and $l = Z$, this event indicates that statement s is executed by Z. Otherwise, id indicates the run of which this event is part, and l is the line number (in a local protocol) of the statement s being executed in this event.

Executions. Let $\langle IK, PS\rangle$ be a protocol. Let $\{\langle ns_1, P_1\rangle, \langle ns_2, P_2\rangle, \ldots, \langle ns_n, P_n\rangle\} = PS$.[1] An *execution* of Π is a tuple $\langle \sigma, subst, prin, lprot\rangle$, where σ is a sequence of events, and for each run of a local protocol, $lprot \in (ID \to PS)$ indicates the local protocol being run, $prin \in (ID \to (Name \setminus \{Z\}))$ indicates the principal running the local protocol, and $subst$ indicates the variable bindings. For $id \in ID$, $subst(id) \in Subst(vars(lprot(id)))$, where for a local protocol P, $vars(P)$ is the set of variables occurring in P. For convenience, we define $subst(Z)$ to be the empty substitution, *i.e.*, $subst(Z) \in Subst(\emptyset)$, and we define $prin(Z) = Z$. An execution must satisfy the following requirements (E1)–(E8).

E1. For each $id \in ID$, letting $\langle ns, P\rangle = lprot(id)$, $subst(id)(\mu) = prin(id)$ and $prin(id) \in ns$.

E2. For each event $\langle id, l, s\rangle$ in σ, if $id \neq Z$, then s is the statement in line l of $lprot(id)$.

E3. For each $id \in ID$, the line numbers in the events in the subsequence of σ containing events of run id are a prefix of the natural numbers. In other words, execution of a local protocol starts at line 0 and proceeds line-by-line.

E4. Every Send event $\langle id, l, \mathrm{Send}(x, t)\rangle$ is immediately followed by a Receive event $\langle id', l', \mathrm{Receive}(t')\rangle$ (called the *corresponding* Receive event)[2] such that

$$t[subst(id)] = t'[subst(id')] \ \wedge\ (id \neq id') \ \wedge\ (id' = Z \vee prin(id') = x[subst(id)]).$$

This allows Z to intercept messages and send messages that appear to be from others principals. Furthermore, every Receive event is immediately preceded by a Send event.

E5. For each NewValue event $\langle id, l, \mathrm{NewValue}(ns, t)\rangle$ in σ, $t[subst(id)]$ is a fresh genval, *i.e.*, does not appear in $\sigma(0..j-1)$ or in initial condition IK, and $id \neq Z$.[3]

E6. For each $id \in ID$, for each $v \in vars(lprot(id))$, if v appears as an argument of *key*, *pubkey*, or *pvtkey* in some statement in $lprot(id)$, then $subst(id)(v) \in Name$.

[1] We use phrases like "let $\langle t_1, t_2\rangle = t$" to indicate that meta-variables t_1 and t_2 are being introduced to denote components of t.

[2] Allowing the corresponding Receive event to be separated from the Send event would lead to an equivalent model, because message delay is already modeled by the possibility of Z intercepting and later re-sending a message.

[3] Allowing Z to generate fresh values would not change any of our results, because inequality tests cannot be expressed in LAP. The Receive statement can express equality tests, *i.e.*, a local protocol can test whether two subterms of messages it received are equal and execute a Receive statement only if they are. However, LAP cannot express inequality tests, *i.e.*, there is no LAP protocol that executes some statement only if two subterms of messages it received are *not* equal.

E7. For each $j \in \mathrm{dom}(\sigma)$, if $\sigma(j)$ has Z as the run-id, *i.e.*, $\sigma(j)$ is of the form $\langle Z, l, s\rangle$, then $l = Z$ and

$$(\exists t \in \mathit{Term}_G, x \in \mathit{Name} : (s = \mathrm{Send}(x,t) \wedge t \in \mathit{known}_Z(IK, \sigma(0..j-1), \mathit{subst})) \vee (s = \mathrm{Receive}(t))),$$

where $\mathit{known}_Z(IK, \sigma, \mathit{subst})$ is the set of ground terms known to Z after the events in σ with initial knowledge IK and bindings subst. Informally, Z knows t iff Z can obtain t by the following procedure: starting with the terms in IK and that Z learned during σ, Z first crumbles these terms into smaller terms by un-doing pairings and encryptions and then constructs larger terms by applying pairing and encryption operators. Let $\mathit{rcvd}_Z(\sigma, \mathit{subst})$ be the set of terms received by Z in σ. Let $\mathit{genvals}_Z(\sigma, \mathit{subst})$ be the set of genvals n such that n is generated in σ by an event $\langle id, l, NewValue(ns, v)\rangle$ and either: (1) Z generated n, *i.e.*, $id = Z$; (2) n is intended to be shared with Z, *i.e.*, $Z \in ns[\mathit{subst}(id)]$; or (3) n is old, *i.e.*, for each principal x in $ns[\mathit{subst}(id)] \setminus \{Z\}$, σ contains an event of the form $\langle id', l', \mathrm{Old}(v')\rangle$ with $\mathit{prin}(id') = x$ and $\mathit{subst}(v') = n$. Let $\mathit{learned}_Z(IK, \sigma, \mathit{subst}) = IK \cup \mathit{rcvd}_Z(\sigma, \mathit{subst}) \cup \mathit{genvals}_Z(\sigma, \mathit{subst})$. Then $\mathit{known}_Z(IK, \sigma, \mathit{subst}) = \mathit{closure}(\mathit{crumble}(\mathit{learned}_Z(IK, \sigma, \mathit{subst})))$, where for a set S of ground terms, $\mathit{crumble}(S)$ is the least set C satisfying[4]

$$\begin{aligned} C = S &\cup \{t_1 \mid (\exists t_2 : \mathit{pair}(t_1,t_2) \in C \vee \mathit{pair}(t_2,t_1) \in C)\} \cup \{t \mid \{t\}_K \in C \wedge K \in C \cap \mathit{Key}_{sym}\} \\ &\cup \{t \mid \{t\}_{\mathit{pubkey}(x)} \in C \wedge \mathit{pvtkey}(x) \in C\} \cup \{t \mid \{t\}_{\mathit{pvtkey}(x)} \in C \wedge \mathit{pubkey}(x) \in C\} \end{aligned} \tag{4}$$

and where $\mathit{closure}(S)$, the set of terms that can be constructed from the terms in S, is the least set C satisfying

$$C = S \cup \{\mathit{pair}(t_1,t_2) \mid t_1 \in C \wedge t_2 \in C\} \cup \{\{t_1\}_K \mid t_1 \in C \wedge K \in C \cap (\mathit{Key}_{sym} \cup \mathit{Key}_{asym})\}. \tag{5}$$

These definitions assume that the symmetric and public-key cryptosystems are perfect (sometimes called "ideal") [WL93a].

There are several minor differences between Woo and Lam's language and semantics and ours. Our NewValue statement is essentially the same as Woo and Lam's NewSecret and NewNonce statements; we changed the name because we sometimes use this statement to generate values that are not secrets (*e.g.*, m in the Otway-Rees protocol). We introduced the special variables μ and p and eliminated the arguments of BeginInit, EndInit, BeginRespond, and EndRespond. Woo and Lam's definition of execution allows Z to execute BeginInit, EndInit, BeginRespond, and EndRespond statements; our definition of execution does not. Letting Z execute these statements is harmless but unnecessary. First, note that Z executing these statements does not change the set of possible future behaviors of any principal, including Z. Furthermore, modifying an execution by inserting or removing events in which these statements are executed by Z cannot affect whether the execution satisfies a correspondence or secrecy property. Woo and Lam's definition of execution also allows Z to execute the Accept statement, which is equivalent to our Old statement. This, too, is harmless but unnecessary, because the only effect of an Accept is to contribute to a secret becoming old and therefore obtainable by Z *via* GetValue, but Z can only Accept a secret that it already knows. Accordingly, our definition of execution does not allow Z to execute the Old statement. In Woo and Lam's semantics, Z executes a GetValue statement to obtain an old generated value; we simplify the definition of execution slightly by making old generated values immediately available to Z.

---

[4] If the public-key cryptosystem is not reversible, the last set comprehension in the definition of *crumble* should be omitted.

2.3 Syntax and Semantics of Requirements

We consider two kinds of correctness requirements: correspondence and secrecy. The semantics of both kinds are given by defining the set of executions that satisfy a requirement. A protocol satisfies a requirement iff every execution of the protocol satisfies the requirement.

A correspondence requirement is specified by a pair, which must be $\langle \text{EndInit}, \text{BeginRespond} \rangle$ or $\langle \text{EndRespond}, \text{BeginInit} \rangle$. An execution $\langle \sigma, subst, prin, lprot \rangle$ satisfies a correspondence requirement $\langle a, a' \rangle$ if, for all distinct pairs $\langle x, y \rangle$ of honest principals, for all prefixes σ' of σ, the number of events in σ' in which x executes a in a run with partner y is less than or equal to the number of events in σ' in which y executes a' in a run with partner x (this implies that every event of the former kind is preceded by a distinct corresponding event of the latter kind).

The *short-term secrecy requirement* expresses secrecy of genvals. An execution $\langle \sigma, subst, prin, lprot \rangle$ satisfies the short-term secrecy requirement iff all genvals in $known_Z(IK, \sigma, subst)$ are in $genvals_Z(\sigma, subst)$.

A *long-term secrecy requirement* expresses secrecy of long-term secrets. A term is *long-term* if it contains no genvals. A long-term secrecy requirement is specified by a set of long-term ground terms not containing the *encrypt* or *pair* operators. An execution $\langle \sigma, subst, prin, lprot \rangle$ satisfies a long-term secrecy requirement S iff $known_Z(IK, \sigma, subst) \cap S = \emptyset$.

For example, the Yahalom protocol satisfies the correspondence requirements $\langle \text{EndInit}, \text{BeginRespond} \rangle$ and $\langle \text{EndRespond}, \text{BeginInit} \rangle$, the short-term secrecy requirement, and the long-term secrecy requirement $\bigcup_{x \in Name \setminus \{Z,S\}} \{key(x, S)\}$.

3 Useless Encryptions

A *ciphertext* is a term with the *encrypt* operator at its root. An *encryption* in an execution $\langle \sigma, subst, prin, lprot \rangle$ of a protocol $\Pi = \langle IK, PS \rangle$ is a pair $\langle j, oc \rangle$ such that: $\sigma(j)$ is a Send event, $\langle id, l, \text{Send}(x, t) \rangle = \sigma(j)$; oc is a subterm of t; oc is a ciphertext $\{t'\}_K$; and if $id = Z$, then $K \in crumble(learned_Z(I, \sigma(0..j-1), subst))$ (otherwise, Z must be forwarding a ciphertext that it learned).

This section identifies encryptions that can be removed from an execution of a protocol. Informally, an encryption in an execution is "useful" (hence cannot be easily removed) only if it is

1. performed by an honest principal,[5] or
2. performed by Z and the resulting ciphertext is (2a) decrypted by an honest principal or (2b) checked for equality with ciphertext produced by a useful encryption.

Case (2b) reflects the transitive nature of usefulness. We illustrate this definition using the following execution fragment:

$$\begin{array}{l} \ldots, \langle Z, Z, \text{Send}(A, c) \rangle, \langle id_0, 0, \text{Receive}(\underline{v_a}) \rangle, \langle id_0, 1, \text{Send}(B, \{\{v_a\}_K\}_{key(A,B)}) \rangle, \\ \langle id_1, 0, \text{Receive}(\{\underline{v_b}\}_{key(A,B)}) \rangle, \langle Z, Z, \text{Send}(B, \{\{c\}_K\}_{K'}) \rangle, \langle id_1, 1, \text{Receive}(\{v_b\}_{K'}) \rangle, \ldots \end{array} \tag{6}$$

Here, c is some ciphertext, and Z is assumed to know K and K' but not $key(A, B)$. The resulting variable bindings are $subst(id_0)(v_a) = c$ and $subst(id_1)(v_b) = \{c\}_K$. Based on the events shown above, all encryptions by Z in c are useless. In the fifth shown event, Z performs two encryptions. The encryption with K' is useful, because it is decrypted by the encryption by id_1 in the next event (this is case 2a). The encryption with K is also useful, because it is checked for equality with the ciphertext produced by the encryption by id_0 in the third shown event (this is case 2b). That encryption by id_0 is useful, because it is performed by an honest principal (this is case 1).

[5] For technical convenience, we classify all encryptions by principals other than Z as useful. It might be more intuitive not to classify them, because some of them might not be genuinely useful in the functioning of the protocol.

All encryptions that are useless (*i.e.*, not useful) can be collectively removed from an execution e of a protocol Π, and the result is an execution e' of Π. Furthermore, for any correspondence or secrecy requirement ϕ, e satisfies ϕ iff e' satisfies ϕ. In general, removing a single useless encryption from e does not yield an execution of Π, because the ciphertext produced by that encryption might be checked for equality with the ciphertext produced by another useless encryption. Removing useless encryptions from Z's Send statements requires adjusting honest principals' variable bindings in a straightforward way. The following two theorems express the key features of useless encryptions.

Theorem 1. Let e be an execution of a protocol Π. Removing all useless encryptions from e yields an execution of Π.

Proof Sketch: The only potential danger in removing useless encryptions is that some local protocol might "block" prematurely, *i.e.*, pattern-matching might fail for some Send event and the corresponding Receive event. The definition of "useful" is designed exactly so that this does not occur. ∎

Theorem 2. Removing useless encryptions from an execution preserves all correspondence and secrecy properties. In other words, for all protocols Π, for all properties ϕ, for all executions e of Π, e satisfies ϕ iff the execution e' obtained by removing all useless encryptions from e satisfies ϕ.

Proof Sketch: For secrecy properties, this follows from the observation that useless encryptions are performed by Z and hence are encryptions with keys known to Z, so removing these encryptions does not affect the set of terms known to Z. For correspondence properties, this follows from the observation that removing useless encryptions from an execution does not change the sequence of BeginInit, EndInit, BeginRespond, and EndRespondevents. ∎

4 Existence of Useless Encryptions

The *encryption height* (or *height*, for short) of a ground term t, denoted $height(t)$, is the maximum number of nested encryption operators in t. The following theorem establishes the existence of useless encryptions in executions containing terms of excessive height. Recall that occurrences of the *encrypt* operator in Receive statements actually represent decryptions, not encryptions.

Theorem 3. Let $e = \langle \sigma, subst, prin, lprot \rangle$ be an execution of a protocol $\langle IK, \{\langle ns_1, P_1\rangle, \langle ns_2, P_2\rangle, \ldots, \langle ns_n, P_n\rangle\}\rangle$. For each local protocol P_k, let s_k be the number of runs of P_k in e; d_k, the number of occurrences of the *encrypt* operator in Receive statements in P_k; and e_k, the number of occurrences of the *encrypt* operator in Send statements in P_k. Let $\sigma(j)$ be a Send event by Z, and let $\langle Z, Z, \mathrm{Send}(x,t)\rangle = \sigma(j)$. If $height(t) > \sum_{k=1}^{n}(e_k + d_k)s_k$, then t contains a ciphertext whose source is a useless encryption.

Proof: The proof is a fairly straightforward counting argument. Details appear in Appendix A. ∎

We conjecture that this bound is asymptotically tight. It is natural for the bound to depend linearly on the number of decryptions by honest principals (*i.e.*, $\sum_{k=1}^{n} d_k s_k$). To see why the number of encryptions by honest principals (*i.e.*, $\sum_{k=1}^{n} e_k s_k$) also matters, consider the LAP protocol $\langle \{K\}, \{\langle\{A\}, P_I\rangle, \langle\{B\}, P_R\rangle\}\rangle$, where

P_I: 0. $\mathrm{Send}(B, \{\{\{A\}_K\}_K\}_{K_{AB}})$

P_R: 0. $\mathrm{Receive}(\{\underline{x}\}_{K_{AB}})$
1. $\mathrm{Receive}(x)$
2. $\mathrm{Send}(Z, K_{AB})$

Consider executions containing exactly one run of each local protocol. Line 2 of P_R (in which B reveals a secret) can be executed only if Z encrypts A twice with K; those encryptions are checked by A's encryptions in line 0 or P_I. Thus, such executions contain at most one decryption by honest principals, but long-term secrecy is violated only if Z performs at least two encryptions.

One idea for prohibiting such protocols is to limit the encryption height of the arguments of Send statements to 2. We conjecture that with this restriction, the bound in Theorem 3 and the corresponding bound in Theorem 4 can be decreased to $\sum_{k=1}^{n} e_k$.

5 Upper Bound on Encryption Height

Theorem 4. Let $\langle IK, \{\langle ns_1, P_1\rangle, \langle ns_2, P_2\rangle, \ldots, \langle ns_n, P_n\rangle\}\rangle$ be a protocol, and let ϕ be a correspondence or secrecy property. For each P_k, let s_k be a bound on the number of runs of P_k in an execution. If any such execution of this protocol violates ϕ, then there exists an execution violating ϕ and in which Z sends terms with encryption height at most $\sum_{k=1}^{n}(e_k + d_k)s_k$, where d_k is the number of occurrences of the *encrypt* operator in Receive statements in P_k, and e_k is the number of occurrences of the *encrypt* operator in Send statements in P_k.

Proof Sketch: This follows immediately from Theorems 1–2.

6 A Lower Bound on Number of Runs

Theorem 5. There exists a family of LAP protocols Π^ℓ and property ϕ such that the minimum number of concurrent runs in an execution of Π^ℓ that violates ϕ is $\Omega((\ell/2-4)^{(\ell/2-4)})$, where ℓ is the maximum number of Send statements in a local protocol of Π^ℓ.

Proof Sketch: We take Π^ℓ to be $\Pi_h^{\ell,\ell}$, where the family of protocols $\Pi_h^{\alpha,\beta}$ is defined in Appendix B. Intuitively, protocol $\Pi_h^{\alpha,\beta}$ performs two depth-first traversals of an α-ary tree of height β before violating ϕ. Each non-leaf node of the tree corresponds to a run of a local protocol. A run of P_I corresponds to the root. Runs of P_R correspond to the non-root non-leaf nodes. Runs of P_S correspond to the leaves. It is necessary to perform *two* depth-first traversals to force all the runs of P_R to be concurrent. During the first traversal, messages sent from a parent to a child have the form $\{v\}_k$, and messages sent from a child to its parent have the form $\{v{\cdot}w\}_k$. During the second traversal, messages sent from a parent to a child have the form $\{v{\cdot}w{\cdot}0\}_k$, and messages sent from a child to its parent have the form $\{v{\cdot}w{\cdot}1\}_k$.

The long-term secrecy requirement $\{key(A,B)\}$ is violated iff P_I runs to completion. P_I can run to completion only in executions containing $\Omega(\alpha^{\beta-1})$ concurrent runs of P_R (the runs of P_S need not be concurrent). If a run id of P_R is aborted after line $3\alpha+3$ and one tries to use a new run id' of P_R (instead of run id) in the second traversal, messages from the first traversal can be replayed to bring id' up to line $3\alpha+4$ with the correct bindings for k, k', and v, but id' would get stuck at line $3\alpha+4$, because id and id' have different bindings for v_0. Thus, the runs of P_R corresponding to nodes of the tree must be concurrent. ∎

6.1 Towards An Upper Bound on Number of Runs

Finding a natural set of restrictions that prohibits protocols like Π_h and allows useful authentication protocols (*e.g.*, those in [MvOV97]) is an open problem. After seeing Π_h, a natural idea is to impose a termination requirement, such as: in an execution including exactly one run of each local protocol, it is possible for each run id to terminate, *i.e.*, to execute the last line of local protocol $lprot(id)$. This requirement does prohibit Π_h. However, it seems not to get at the root of the problem, because Π_h can be modified to satisfy it, at the cost of increasing the number of local protocols by a small constant (*e.g.*, from 3 to 6). The idea is to break P_I and P_R into pieces, each of which starts by receiving a ciphertext encoding the "current" local state and ends by sending a ciphertext encoding the "next" local state. This is essentially the same idea already used to break up the tree traversal in Π_h. As mentioned in Section 1, we are currently studying dynamic restrictions that will lead to small upper bounds on the number of runs.

References

[Aba97] Martín Abadi. Explicit communication revisited: Two new attacks on authentication protocols. *IEEE Transactions on Software Engineering*, 23(3):185–186, March 1997.

[AJ98] Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes. In *Proc. 4th Intl. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[AN95] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In *Proc. Int'l. Conference on Advances in Cryptology (CRYPTO 95)*, volume 963 of *Lecture Notes in Computer Science*, pages 236–247. Springer-Verlag, 1995.

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[BAN90] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990. An expanded version appeared as Res. Rep. 39, Systems Research Center, Digital Equipment Corp., Palo Alto, CA, Feb. 1989.

[Bol98] Dominique Bolignano. Integrating proof-based and model-checking techniques for the formal verification of cryptographic protocols. In Alan J. Hu and Moshe Y. Vardi, editors, *Proc. Tenth Int'l. Conference on Computer-Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state Concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGJ95] Edmund M. Clarke, Orna Grumberg, and Somesh Jha. Verifying parameterized networks using abstractions and regular languages. In *Proc. Sixth Int'l. Conference on Concurrency Theory (CONCUR)*, 1995.

[CMJ98] Edmund Clarke, Will Marrero, and Somesh Jha. A machine checkable logic of knowledge for specifying security properties of electronic commerce protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, June 1998.

[CS96] Rance Cleaveland and Steve Sims. The NCSU Concurrency Workbench. In R. Alur and T. Henzinger, editors, *Proc. 8th Int'l. Conference on Computer-Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397. Springer-Verlag, 1996.

[DDHY92] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525. IEEE Computer Society, 1992.

[DK97] Z. Dang and R. A. Kemmerer. Using the ASTRAL model checker for cryptographic protocol analysis. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997.

[DS81] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):198–208, 1981.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.

[EN96] E. Allen Emerson and Kedar S. Namjoshi. Automated verification of parameterized synchronous systems. In *Proc. 8th Int'l. Conference on Computer-Aided Verification (CAV)*, 1996.

[FHG98] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *Proc. 11th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1998.

[GLR95] Li Gong, Patrick Lincoln, and John Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Dependable Computing for Critical Applications—5*, pages 79–90. IFIP WG 10.4, preliminary proceedings, 1995.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[HTWW96] Nevin Heintze, J. D. Tygar, Jeannette Wing, and Hao-Chi Wong. Model checking electronic commerce protocols. In *Proc. of the USENIX 1996 Workshop on Electronic Commerce*, November 1996.

[KM95] R. P. Kurshan and K. L. McMillan. A structural induction theorem for processes. *Information and Computation*, 117(1), 1995.

[KN93] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5), Internet Request for Comments 1510, September 1993.

[Kur94] Robert P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach*. Princeton University Press, 1994.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *Proc. Workshop on Tools and Algorithms for The Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, March 1996.

[Low98a] Gavin Lowe. Towards a completeness result for model checking of security protocols. Technical Report 1998/6, Dept. of Mathematics and Computer Science, University of Leicester, 1998.

[Low98b] Gavin Lowe. Towards a completeness result for model checking of security protocols (extended abstract). In *Proc. 11th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1998.

[LR97] Gavin Lowe and Bill Roscoe. Using csp to detect errors in the tmn protocol. *IEEE Transactions on Software Engineering*, 23(10), 1997.

[MCF87] J. Millen, S. Clark, and S. Freedman. The Interrogator: protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2), February 1987.

[MCJ97] Will Marrero, Edmund Clarke, and Somesh Jha. A model checker for authentication protocols. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997. Available via http://dimacs.rutgers.edu/Workshops/Security/program2/program.html.

[MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proc. 18th IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.

[MR97] Dahlia Malkhi and Michael Reiter. A high-throughput secure reliable multicast protocol. *The Journal of Computer Security*, 5:113–127, 1997.

[MSS98] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In *Seventh USENIX Security Symposium*, pages 201–216, 1998.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography.* CRC Press, 1997.

[OPT97] Donal O'Mahony, Michael Peirce, and Hitesh Tewari. *Electronic Payment Systems.* Artech House, Boston, 1997.

[OR87] Dave Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[Rei96] Michael K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, 1996.

[Ros95] A. W. Roscoe. Modelling and verifying key exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society Press, 1995.

[Ros98] A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *Proc. 11th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.

[WL93a] Thomas Y. C. Woo and Simon S. Lam. A semantic model for authentication protocols. In *Proc. 14th IEEE Symposium on Research in Security and Privacy*, pages 178–194. IEEE Computer Society Press, May 1993. Available via http://www.cs.utexas.edu/users/lam/NRL/network_security.html.

[WL93b] Thomas Y. C. Woo and Simon S. Lam. Verifying authentication protocols: Methodology and example. In *Proc. Int'l. Conference on Network Protocols*, October 1993.

[WL94] Thomas Y.C. Woo and Simon S. Lam. A lesson in authentication protocol design. *ACM Operating Systems Review*, 28(3):24–37, July 1994.

A Proof of Theorem 3

We formalize the necessary concepts and then prove the theorem.

A *decryption* in an execution $\langle \sigma, subst, prin, lprot \rangle$ is a pair $\langle j, oc \rangle$ such that: $\sigma(j)$ is a Receive event, $\langle id, l, \text{Receive}(x, t) \rangle = \sigma(j)$; oc is a subterm of t; oc is a ciphertext; and $id \neq Z$ (decryptions by Z are not of interest here).

An encryption or decryption $\langle j, oc \rangle$ is *performed by* the principal $prin(id)$, where we let $\langle id, l, s \rangle = \sigma(j)$.

Let t be a ground term, and let t' be a term (think of t as a message and of t' as the "pattern" in a Receive statement). Suppose there exists a substitution θ such that $t = t'[\theta]$. Then there is a natural correspondence between subterms of t and (occurrences of) subterms of t'. We say that a subterm of t' *covers* the corresponding subterms of t. For example, if $t = (N_1 \cdot (N_2 \cdot N_3))$ and $t' = N_1 \cdot v_2$, then the occurrence of N_1 in t' covers the occurrence of N_1 in t, and the occurrence of v_2 in t' covers the occurrences of N_2, N_3, and $(N_2 \cdot N_3)$ in t.

Each occurrence of each ciphertext in each variable binding and each occurrence of each ciphertext in each Send event by Z is produced by some encryption, called the *source* of that occurrence. The source of a ciphertext c is uniquely determined in a straightforward way (details omitted) by the data-flow in the execution,[6] except that the source of an occurrence of a ciphertext c might not be uniquely determined if c was sent by Z and there are multiple events through which Z learned c. For example, suppose Z doesn't know K_1 and Z receives $\{N\}_{K_1}$ from A, then receives $\{N\}_{K_1}$ from B, and then sends $\{N\}_{K_1}$ to S, who binds the received ciphertext to v. Assuming A and B each encrypted their own message, it is undetermined whether A's encryption or B's encryption should be regarded as the

[6] If $\langle j, oc \rangle$ is an encryption in a Send event by Z, then $\langle j, oc \rangle$ is its own source.

source of the ciphertext in v. For our purposes, it makes no difference which encryption is chosen as the source in such cases.

An encryption $\langle j, oc\rangle$ is *decrypted* if it matches with a decryption in the corresponding Receive event, or if some ciphertext with source $\langle j, oc\rangle$ matches with a decryption in some Send/Receive pair of events. Formally:

- Let $\langle id, l, \mathrm{Send}(x,t)\rangle = \sigma(j)$ and $\langle id', l', \mathrm{Receive}(x',t')\rangle = \sigma(j+1)$. If $id' \neq Z$ and the subterm of t' that covers oc is a ciphertext (not a variable), then $\langle j, oc\rangle$ is decrypted.
- If there exists a Send event $\sigma(j')$ such that, letting $\langle id, l, \mathrm{Send}(x,t)\rangle = \sigma(j')$ and $\langle id', l', \mathrm{Receive}(x',t')\rangle = \sigma(j'+1)$, $id' \neq Z$ and there exists an occurrence ov of a variable v in t such that there exists an occurrence oc_1 of a ciphertext in $subst(id)(v)$ such that $\langle j, oc\rangle$ is the source of oc_1 and the subterm of t' that covers the occurrence of oc_1 in t produced by instantiation of ov is a ciphertext (not a variable), then $\langle j, oc\rangle$ is decrypted.

An encryption $\langle j, oc\rangle$ is *checked by* an encryption $\langle j', oc'\rangle$ if there is a ciphertext oc_1 with source $\langle j, oc\rangle$ that is received by a principal who checks it for equality with a ciphertext oc'_1 with source $\langle j', oc'\rangle$; the equality check is represented by oc'_1 occurring (in an appropriate position) in the binding of a variable v such that v covers oc_1 in the pattern-matching for a Receive event. Formally, $\langle j, oc\rangle$ is checked by $\langle j', oc'\rangle$ if there exists a Send event $\sigma(j'')$ such that, letting $\langle id, l, \mathrm{Send}(x,t)\rangle = \sigma(j'')$ and $\langle id', l', \mathrm{Receive}(x',t')\rangle = \sigma(j''+1)$, $id' \neq Z$ and there exists an occurrence ov of a variable v in t and an occurrence ov' of a variable v' in t' such that there exists an occurrence oc_1 of a ciphertext in $subst(id)(v)$ and an occurrence oc'_1 of a ciphertext in $subst(id')(v')$ such that $\langle j, oc\rangle$ is the source of oc_1, and $\langle j', oc'\rangle$ is the source of oc'_1, and in the matching of $t[subst(id)]$ with $t'[subst(id')]$, oc_1 corresponds to oc'_1.

An encryption $\langle j, oc\rangle$ is *useful* if there exists a sequence es of encryptions such that: (1) $es(0) = \langle j, oc\rangle$; (2) $es(|es|-1)$ is decrypted or performed by a principal other than Z; and (3) if $|es| > 1$, then for all k from 0 to $|es|-2$, $es(k)$ is checked by $es(k+1)$.

Proof of Theorem 3: The number of decryptions in e is bounded by $n_d = \sum_{k=1}^{n} d_k s_k$. The number of encryptions in e performed by principals other than Z is bounded by $n_e = \sum_{k=1}^{n} e_k s_k$. By hypothesis, $height(t) > n_e + n_d$.

Consider a longest sequence ocs of nested ciphertexts in t such that $ocs(0)$ contains a single occurrence of the *encrypt* operator and for all k from 0 to $|ocs|-2$, $ocs(k)$ is an occurrence of a proper subterm of $ocs(k+1)$. Note that $|ocs| = height(t)$, so by hypothesis, $|ocs| > n_e + n_d$. How many elements of ocs have useful sources? The definition of "useful" implies that if the source of $ocs(k)$ is useful, then there is a witness $es(k)$. So, the number of elements of ocs with useful sources is bounded by $w_d + w_e$, where w_d is the number of elements of ocs with useful sources such that the witness ends with an encryption that is decrypted, and w_e is the number of elements of ocs with useful sources such that the witness ends with an encryption performed by a principal other than Z.[7] It suffices to show that $w_d \leq n_d$ and $w_e \leq n_e$, because then we can conclude that there are at least $height(t) - (n_e + n_d)$ elements of ocs having sources that are not useful.

We show first that $w_d \leq n_d$. For each decryption in e, there is at most one encryption that is decrypted by that decryption. For each encryption $\langle j, oc\rangle$ in e, there is at most one element of ocs having $\langle j, oc\rangle$ as its source, because all occurrences of ciphertexts having the same source are occurrences of the same term, while each element of ocs is an occurrence of a distinct term. It follows

[7] We say "bounded by" rather than "equal to", because these two categories of witnesses are not necessarily disjoint: an encryption can be both decrypted and performed by a principal other than Z.

from these two facts that, for each decryption, there is at most one element of *ocs* whose witness ends with that decryption, so $w_d \leq n_d$.

We now show that $w_e \leq n_e$. Observe that, for all encryptions $\langle j, oc\rangle$ and $\langle j', oc'\rangle$, if $\langle j, oc\rangle$ is checked by $\langle j', oc'\rangle$, then oc and oc' are occurrences of the same term. So, by transitivity of equality, for each element $ocs(k)$ of ocs with a witness $es(k)$ in this second category, the last element of $es(k)$ is an occurrence of the same term as $ocs(k)$. Since each element of ocs is an occurrence of a distinct term, we conclude that for each encryption in e performed by a principal other than Z, there is at most one element of ocs whose witness ends with that encryption, so $w_e \leq n_e$. ∎

B A Lower Bound on Number of Runs

We describe a family Π_h of LAP protocols that require many runs to attack. Π_h can be expressed in Woo and Lam's language [WL93a]; it is not an artifact of our modifications. The family of protocols has two positive integer parameters, α and β. To express the family of protocols compactly, we introduce a for-loop macro. For given values of α and β, the for-loop macro is unrolled (by a conceptual pre-processor), yielding a particular protocol $\Pi_h^{\alpha,\beta}$ in the family.

Let $K_0, K_1, \ldots$ be constants in Key_{sym}. Π_h can easily be modified to use session keys produced by NewValue instead of these long-term keys. In P_I^{β}, the second tree traversal starts at line $\beta + 3$; in P_R^{α}, the second traversal starts at line $3\alpha + 4$. Protocol $\Pi_h^{\alpha,\beta}$ is

$$\langle\{key(Z,S)\}, \{\langle\{A,B\}, P_I^{\alpha}\rangle, \langle\{A,B\}, P_R^{\beta}\rangle, \langle\{S\}, P_S\rangle\}\rangle, \tag{7}$$

where

P_R^{α}:
0. Receive($\{\underline{k}\cdot\underline{k'}\}_{key(A,B)}$)
1. Receive($\{\underline{v}\}_k$)
for $a := \alpha$ downto 1
  $3(\alpha-a)+2$. NewValue($\emptyset, \underline{v_a}$)
  $3(\alpha-a)+3$. Send($p, \{v_a\}_{k'}$)
  $3(\alpha-a)+4$. Receive($\{v_a\cdot\underline{w_a}\}_{k'}$)
$3\alpha+2$. NewValue($\emptyset, \underline{v_0}$)
$3\alpha+3$. Send($p, \{v\cdot v_0\}_k$)
$3\alpha+4$. Receive($\{v\cdot v_0\cdot 0\}_k$)
for $a := \alpha$ downto 1
  $5\alpha-2a+5$. Send($p, \{v_a\cdot w_a\cdot 0\}_{k'}$)
  $5\alpha-2a+6$. Receive($\{v_a\cdot w_a\cdot 1\}_{k'}$)
$5\alpha+5$. Send($p, \{v\cdot v_0\cdot 1\}_k$)

P_I^{β}:
for $b := \beta$ downto 1
  $\beta-b$. Send($p, \{K_b\cdot K_{b-1}\}_{key(A,B)}$)
β. NewValue($\emptyset, \underline{v}$)
$\beta+1$. Send($p, \{v\}_{K_\beta}$)
$\beta+2$. Receive($\{v\cdot\underline{w}\}_{K_\beta}$)
$\beta+3$. Send($p, \{v\cdot w\cdot 0\}_{K_\beta}$)
$\beta+4$. Receive($\{v\cdot w\cdot 1\}_{K_\beta}$)
$\beta+5$. Send($p, key(A,B)$)

P_S:
0. Receive($\{\underline{v}\}_{K_0}$)
1. Send($A, \{v\cdot 0\}_{K_0}$)
2. Receive($\{\underline{v'}\cdot\underline{w'}\cdot 0\}_{K_0}$)
3. Send($A, \{v\cdot w'\cdot 1\}_{K_0}$)