A Reduction for Automated Verification of Authentication Protocols

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

June 18, 1999

Abstract

Authentication protocols (including protocols that provide key establishment) are designed to work correctly in the presence of an adversary that can 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 and provide a rigorous basis for automated verification of authentication protocols. However, few such bounds are known. This paper defines a domain-specific language for authentication protocols and establishes an upper bound on the resources needed to attack a large subset of the protocols expressible in that language, including versions of the Yahalom, Otway-Rees, and Needham-Schroeder public-key protocols.

1 Introduction

Many protocols are designed to work correctly in the presence of an adversary that can 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 [DvOW92, MvOV97]. 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, THG98]. 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].

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,

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

DK97, LR97, MMS97, MCJ97, MSS98, Bol98] offer strong evidence that state-space exploration of authentication protocols and similar kinds of protocols is feasible when small upper bounds are imposed on the size of messages and the number of protocol runs per execution. However, the bounds used in most of those case studies were not 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 a domain-specific language for authentication protocols and proves a reduction for a large subset of the protocols expressible in that language. One novel idea in our reduction is the use of a *dynamic* restriction, which is really just a safety property [AS85], as well as *static* (i.e., syntactic) restrictions to characterize that subset. With static restrictions alone, it seems difficult to find restrictions that are both strong enough to enable the proof and weak enough to be satisfied by well-known protocols. Essentially, the dynamic restriction is an auxiliary safety property that would be established as part of the proof in any case.

For a reduction to be useful for automated verification, its hypotheses must be automatically checkable. So, roughly speaking, we hypothesize that an appropriate bounded (*i.e.*, finite-state) system satisfies certain static restrictions and the dynamic restriction and prove that (1) the unbounded system satisfies the dynamic restriction, and (2) the unbounded system satisfies a correctness requirement iff the bounded system does. Thus, checking whether a given protocol satisfies our restrictions and, if so, checking whether the protocol is correct are finite-state problems that can be decided by straightforward state-space exploration.

Few other reductions applicable to authentication protocols are known. Most existing techniques for automated analysis of systems with unbounded numbers of processes, such as [CGJ95, KM95, EN96, AJ98], are neither aimed at nor applicable to authentication protocols. One reason they do not apply is that authentication protocols involve the generation of fresh values (for use as nonces or session keys), so the set of values grows as the number of processes (equivalently, the number of protocol runs) grows. Dolev and Yao developed analysis algorithms that directly verify secrecy requirements [DY83] of cryptographic protocols; however, their algorithms do not consider correspondence properties, do not consider known-key attacks, and apply to a severely restricted class of protocols, which excludes almost all well-known authentication protocols (e.g., the Otway-Rees [OR87] 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].[1] However, that result does not extend to correspondence requirements [Low98a, p. 61], does not consider known-key attacks [MvOV97, p. 496], and is based on restrictions that exclude all protocols that use temporary secrets and well-known protocols such as the Otway-Rees [OR87] and Yahalom [BAN90] protocols. In contrast, our reduction considers secrecy and correspondence requirements, accommodates known-key attacks, and applies to some protocols that use temporary secrets. For example, it applies to the Yahalom protocol, the Otway-Rees protocol, and Lowe's corrected version of the Needham-Schroeder public-key protocol [Low96]. The basic idea of our reduction is to start from an event that violates of a correctness requirement or dynamic restriction in some execution, define the set of events on which the violating event depends (other events can

[1] It is not strictly a generalization: the reduction in [Low98a, Low98b] does not apply to the corrected protocol from [Low96], because message 3 violates Assumption 2.

be pruned away without affecting the violation), and then obtain a bound on the size of that set.

Work is in progress to reformulate these results within a simpler and more declarative framework, such as the elegant strand space model of Thayer, Herzog, and Guttman [THG98]. Directions for future work include broadening the scope of our results by considering hash functions, timestamps, recency requirements, and key confirmation requirements [MvOV97, p. 492], and extending our approach to handle other problems mentioned above, such as reliable multicast and electronic payment.

2 Model of Authentication Protocols

Our model of authentication is based closely on Woo and Lam's model [WL93]. We call the language LAP (Language for Authentication Protocols).

2.1 Syntax of LAP

Primitive Values. The set of *primitive values* is $\mathit{Prim} = \mathit{Name} \cup \mathit{Genval} \cup \mathit{Key}_{LT}$, where Name is the set of names (of principals), including a distinguished name Z for the adversary, Genval is a set of symbols representing genvals ("freshly generated values", which can be used as nonces or keys), and Key_{LT} is a set of symbols representing long-term encryption keys, defined by $\mathit{Key}_{LT} = \mathit{Key}_{sym} \cup \mathit{Key}_{asym}$, where the set of symmetric keys is $\mathit{Key}_{sym} = \bigcup_{x,y \in \mathit{Name}} \{\mathit{key}(x,y)\}$, and the set of asymmetric keys is $\mathit{Key}_{asym} = \bigcup_{x \in \mathit{Name}} \{\mathit{pubkey}(x), \mathit{pvtkey}(x)\}$. $\mathit{pubkey}(x)$ and $\mathit{pvtkey}(x)$ represent x's public and private keys, respectively. Distinct symbols in Prim are assumed to represent distinct values.

Terms. The set Op of operators is $\mathit{Op} = \{\mathit{encrypt}, \mathit{pair}, \mathit{key}, \mathit{pubkey}, \mathit{pvtkey}\}$. The term $\mathit{encrypt}(t_1, t_2)$ represents t_1 encrypted with key t_2 and is usually written as $\{t_1\}_{t_2}$. The term $\mathit{pair}(t_1, t_2)$ represents t_1 paired with t_2 and is usually written as $t_1 \cdot t_2$; similarly, $\mathit{pair}(t_1, \mathit{pair}(t_2, t_3))$ is usually written as $t_1 \cdot t_2 \cdot t_3$; and so on. 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. A *key term* is a term whose outermost operator is *key*, *pubkey*, or *pvtkey*.[2]

Each variable has a type, which is either Prim or All. Variables of type Prim can be bound only to primitive values. Variables of type All can be bound to all ground terms. By convention, lowercase and uppercase variables have type Prim and All, respectively. A *ciphertext* is a term whose outermost operator is *encrypt*. A subterm t' of t is *unencrypted* in t if t' is not in the scope of *encrypt* in t. Note that a ciphertext can be unencrypted in a term. The *encryption height* of a ground term t is the maximum number of nested occurrences of *encrypt* in t. For example, the encryption heights of $\{v_1\}_{K_1} \cdot \{v_2\}_{K_2}$ and $\{\{v_1\}_{K_1}\}_{K_2}$ are 1 and 2, respectively.

The kinds of statements are:

BeginInit(t): Begin initiator protocol with argument t. This statement is included to facilitate expression of correspondence requirements (see Section 2.3). The argument t is an arbitrary term used to distinguish different executions of BeginInit.

BeginRespond(t): Begin responder protocol with argument t.

[2] Key terms may contain variables; elements of Key_{LT} cannot.

EndInit(t): Successful completion of initiator protocol with argument t.

EndRespond(t): Successful completion of responder protocol with argument t.

NewValue(ns, v): Generate an unguessable value t (*e.g.*, a nonce or session key) and bind 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 in the set ns has executed Old(t). Thus, if ns is empty (or as a special case, if $Z \in ns$), t is old as soon as it is generated. In our model, when a genval becomes old, it is immediately made known to Z, allowing Z to perform known-key attacks. The adversary can generate values but does not use variables, so the adversary can execute NewValue(ns, t), where the ground term t is the fresh value generated by this statement.

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

Receive(t): Receive a message t' and bind 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. Note that 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. A subterm $\{t\}_k$ in a Receive statement represents a decryption with the inverse key of k, not an encryption. (A symmetric key is its own inverse.)

Old(t): Indicate that the executing principal's part in session set-up involving t is finished. This is used to model known-key attacks, as described above.

Local Protocols. 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. *Server (local) protocols* do not contain any of these four kinds of statements.

Reserved Variables. When an honest principal x starts executing a local protocol: (*i*) the variable μ (of type Prim) is automatically bound to x; (*ii*) in initiator and responder protocols, the variable p (of type Prim) 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; (*iii*) in server protocols, the variables i and r (of type Prim) are automatically bound to arbitrary and distinct elements of $Name$, identifying the initiator and responder that an instance of the server protocol is willing to serve.

Defining Occurrences and Uses. A *defining occurrence* of a variable v is an occurrence of v that (*i*) appears in the first statement containing v and (*ii*) appears in Receive or the second argument of NewValue; as exceptions to this, there are no defining occurrences of μ and p in initiator and responder protocols or of i and r in server protocols. All non-defining occurrences of variables are called *uses*. Note that, for a variable v and a statement s, all occurrences of v in s are defining occurrences, or all of them are uses. A variable may have multiple defining occurrences, which must occur in a single Receive statement. For the reader's convenience, defining occurrences of variables are underlined in local protocols.

Well-Formedness Requirements. The well-formedness requirements are: (WF1) Variables are bound before they are used, *i.e.*, for each variable v except μ and p in P_I and P_R and except i and r in P_S, each statement containing uses of v is preceded by a statement containing defining occurrences of v. (WF2) Variables are single-assignment, *i.e.*, for each variable v, uses of v do not occur in the second argument of NewValue statements. (WF3) Keys are parameterized by names or primitive variables, *i.e.*, for each occurrence of *key*, *pubkey*, or *pvtkey*, the arguments are names or primitive variables. (WF4) Each argument of Old is a variable of type Prim. (WF5) Server protocols do not contain Old. (WF6) A session key must be known before it can be used for decryption; more precisely,[3] in a Receive, if the second argument of an *encrypt* contains a defining occurrence of a variable v, then there is also a defining occurrence of v in the first argument of an *encrypt* whose second argument contains no defining occurrences of variables. (WF7) Honest principals do not encrypt using other principals' private keys; more precisely, if a Send contains a ciphertext of the form $\{t_1\}_{pvtkey(t_2)}$, then t_2 is μ. (WF8) Honest principals do not decrypt using other principals' private keys; more precisely, if a Receive contains a ciphertext of the form $\{t_1\}_{pubkey(t_2)}$ and t_1 contains a defining occurrence of a variable, then t_2 is μ.[4] (WF9) Honest principals don't use shared keys that they don't know; more precisely, if a Send or Receive contains a ciphertext of the form $\{t_1\}_{key(t_2,t_3)}$, then t_2 is μ or t_3 is μ.

Protocols. A *protocol* is a pair $\langle IK, PS\rangle$, where IK (mnemonic for *initial knowledge*) is a set of ground terms, and PS is a set of pairs of the form $\langle ns, P\rangle$, where $ns \subseteq (\mathit{Name} \setminus \{Z\})$ 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.

Example. Here is a simple unilateral authentication protocol that uses a temporary secret:

$$\Pi_U = \langle\{key(A,Z), key(Z,A), key(Z,B), key(B,Z)\}, \{\langle\{A,B\},P_I\rangle, \langle\{A,B\},P_R\rangle\}\rangle, \qquad (1)$$

where

P_I:	0. NewValue($\{p\}, \underline{n}$)	P_R:	0. Receive($p\cdot\{\underline{n}\cdot\mu\}_{key(p,\mu)}$)
	1. Send($p, \mu\cdot\{n\cdot p\}_{key(\mu,p)}$)		1. BeginRespond(n)
	2. Receive(n)		2. Old(n)
	3. EndInit(n)		3. Send(p, n)

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

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

where

[3] This is stronger than necessary but is relatively simple and seems adequately permissive.

[4] If t_1 contains no defining occurrences, then the pattern-matching can be implemented by encrypting a candidate message using the appropriate principal's public key.

P_I:	P_R:	P_S:
0. NewValue($\emptyset, \underline{ni}$)	0. Receive($p \cdot \underline{ni}$)	0. Receive($r \cdot \{i \cdot \underline{ni} \cdot \underline{nr}\}_{\mathit{key}(r,\mu)}$)
1. Send($p, \mu \cdot ni$)	1. NewValue($\{\mu, p\}, \underline{nr}$)	1. NewValue($\{i, r\}, \underline{k}$)
2. Receive($\{p \cdot \underline{k} \cdot ni \cdot \underline{nr}\}_{\mathit{key}(\mu,S)} \cdot \underline{X}$)	2. BeginRespond(ni)	2. Send($i, \{r \cdot k \cdot ni \cdot nr\}_{\mathit{key}(i,\mu)}$
3. BeginInit(nr)	3. Send($S, \mu \cdot \{p \cdot ni \cdot nr\}_{\mathit{key}(\mu,S)}$)	$\cdot \{i \cdot k\}_{\mathit{key}(r,\mu)}$)
4. Send($p, X \cdot \{nr\}_k$)	4. Receive($\{p \cdot \underline{k}\}_{\mathit{key}(\mu,S)} \cdot \{nr\}_{\underline{k}}$)	
5. EndInit(ni)	5. EndRespond(nr)	
6. Old(k)	6. Old(k)	
7. Old(nr)	7. Old(nr)	

Example. In LAP, Lowe's corrected version of the Needham-Schroeder public-key protocol [Low96] is (following Lowe, we omit the steps used to obtain public keys from a server)

$$\Pi_{NSPK} = \langle \{\mathit{pubkey}(A), \mathit{pubkey}(B), \mathit{pubkey}(Z), \mathit{pvtkey}(Z)\}, \{\langle \{A, B\}, P_I \rangle, \langle \{A, B\}, P_R \rangle\} \rangle, \tag{3}$$

where

P_I:	P_R:
0. NewValue($\{\mu, p\}, \underline{ni}$)	0. Receive($\{\underline{ni} \cdot p\}_{\mathit{pubkey}(\mu)}$)
1. Send($p, \{ni \cdot \mu\}_{\mathit{pubkey}(p)}$)	1. NewValue($\{\mu, p\}, \underline{nr}$)
2. Receive($\{ni \cdot \underline{nr} \cdot p\}_{\mathit{pubkey}(\mu)}$)	2. BeginRespond(ni)
3. BeginInit(nr)	3. Send($p, \{ni \cdot nr \cdot \mu\}_{\mathit{pubkey}(p)}$)
4. Send($p, \{nr\}_{\mathit{pubkey}(p)}$)	4. Receive($\{nr\}_{\mathit{pubkey}(\mu)}$)
5. EndInit(ni)	5. EndRespond(nr)
6. Old(ni)	6. Old(ni)
7. Old(nr)	7. Old(nr)

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

$$\Pi_{OR} = \langle \{\mathit{key}(Z, S)\}, \{\langle \{A, B\}, P_I \rangle, \langle \{A, B\}, P_R \rangle, \langle \{S\}, P_S \rangle\} \rangle, \tag{4}$$

where

P_I:	P_R:	P_S:
0. NewValue($\emptyset, \underline{m}$)	0. Receive($\underline{m} \cdot p \cdot \mu \cdot \underline{X}$)	0. Receive($\underline{m} \cdot i \cdot r \cdot \{\underline{x} \cdot \underline{m} \cdot i \cdot r\}_{\mathit{key}(i,\mu)}$
1. NewValue($\emptyset, \underline{n}$)	1. BeginRespond(m)	$\cdot \{\underline{y} \cdot \underline{m} \cdot i \cdot r\}_{\mathit{key}(r,\mu)}$)
2. BeginInit(m)	2. NewValue($\emptyset, \underline{n}$)	1. NewValue($\{i, r\}, \underline{k}$)
3. Send($p, m \cdot \mu \cdot p$	3. Send($S, m \cdot p \cdot \mu \cdot X \cdot \{n \cdot m \cdot p \cdot \mu\}_{\mathit{key}(\mu,S)}$)	2. Send($r, m \cdot \{x \cdot k\}_{\mathit{key}(i,\mu)}$
$\cdot \{n \cdot m \cdot \mu \cdot p\}_{\mathit{key}(\mu,S)}$)	4. Receive($m \cdot \underline{X} \cdot \{n \cdot \underline{k}\}_{\mathit{key}(\mu,S)}$)	$\cdot \{y \cdot k\}_{\mathit{key}(r,\mu)}$)
4. Receive($m \cdot \{n \cdot \underline{k}\}_{\mathit{key}(\mu,S)}$)	5. Send($p, m \cdot X$)	
5. Old(k)	6. Old(k)	
6. EndInit(m)	7. EndRespond(m)	

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

$$\Pi_{NSSK} = \langle \{\mathit{key}(Z, S)\}, \{\langle \{A, B\}, P_I \rangle, \langle \{A, B\}, P_R \rangle, \langle \{S\}, P_S \rangle\} \rangle, \tag{5}$$

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(nr)
6. Send($\{nr\cdot p\}_k$)
7. EndInit(k)
8. Old(k)

P_R:

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

P_S:

0. Receive($i\cdot r\cdot \underline{ni}$)
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 shared-key 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($\{ni\cdot p\cdot \underline{k}\cdot \underline{X}\}_{key(\mu,S)}$) and line 2 of P_S to Send($i, \{ni\cdot r\cdot k\cdot \{k\cdot i\}_{key(r,\mu)}\}_{key(i,\mu)}$). A straightforward argument shows that correctness of the above protocol implies correctness of the original version of the 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, sequences may be infinite or finite. Thus, the initial element of a sequence σ is $\sigma(0)$; the next element is $\sigma(1)$; and so on. For a function f, dom(f) denotes the domain of f. The length a sequence σ, denoted $|\sigma|$, is the size of dom(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* of a local protocol can be thought of as a thread that executes the local protocol once and then exits. A *run-id* identifies a particular run. 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 that respect the types of the variables. We use "binding" and "substitution" interchangeably. The application of a substitution θ to a term t is denoted $t[\theta]$. For a local protocol P, $vars(P)$ is the set of variables occurring in P. An *event* is a tuple of the form $\langle id, l, s\rangle$ or $\langle Z, Z, s\rangle$, where id is a run-id, l is a natural number, and s is a statement. The event $\langle id, l, s\rangle$ indicates that run id executes statement s, which is line number l of a local protocol (the local protocol associated with id is specified separately, as described in the next paragraph). The event $\langle Z, Z, s\rangle$ indicates that Z executes statement s.

Executions. Let $\langle IK, PS\rangle$ be a protocol. Let $\{\langle ns_1, P_1\rangle, \langle ns_2, P_2\rangle, \ldots, \langle ns_n, P_n\rangle\} = PS$.[5] An *execution* of Π is a tuple $\langle \sigma, subst, lprot\rangle$, where σ is a sequence of events, and for each run-id id, $lprot(id) \in \{P_1, P_2, \ldots, P_n\}$ is the local protocol being executed in that run, and $subst(id) \in Subst(vars(lprot(id)))$ is the variable bindings for that run. For convenience, we define $subst(Z)$ to be the substitution that binds μ to Z, i.e., $subst(Z) \in Subst(\{\mu\})$ and $subst(Z)(\mu) = Z$. An execution must satisfy the following requirements (E1)–(E7).

[5] 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.

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

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

E3. For each $id \in ID$, letting σ' be the subsequence of σ containing the events of run id, the line numbers in σ' 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. For each Send event $e = \langle id, l, \mathrm{Send}(x, t)\rangle$, e is the last event in the execution or is immediately followed by a Receive event $\langle id', l', \mathrm{Receive}(t')\rangle$, called the *corresponding* Receive event,[6] such that

$$t[\mathit{subst}(id)] = t'[\mathit{subst}(id')] \;\land\; (id \neq id') \;\land\; (id' = Z \lor \mathit{subst}(id')(\mu) = x[\mathit{subst}(id)]).$$

This allows Z to intercept messages and send messages that appear to be from other principals. 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, $t[\mathit{subst}(id)]$ is a fresh genval, *i.e.*, $t[\mathit{subst}(id)]$ does not appear in IK or in preceding events in σ.

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

E7. For each $j \in \mathrm{dom}(\sigma)$, if $\sigma(j)$ is of the form $\langle Z, Z, s\rangle$, then (using Lamport's bullet-style notation for lists of conjuncts or disjuncts [Lam93])

$$\begin{aligned}
&\lor\ (\exists t \in \mathit{Prim}, ns \subseteq \mathit{Name} : s = \mathrm{NewValue}(ns, t))\\
&\lor\ (\exists t \in \mathit{Term}_G, x \in \mathit{Name} : s = \mathrm{Send}(x, t) \land t \in \mathit{known}_Z(IK, \sigma(0..j-1), \mathit{subst}))\\
&\lor\ (\exists t \in \mathit{Term}_G : s = \mathrm{Receive}(t))
\end{aligned}$$

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 crumbles these terms into smaller terms by un-doing pairings and encryptions and then constructs larger terms using pairing and encryption. Let $\mathit{rcvd}_Z(\sigma)$ 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 $\langle id, l, \mathrm{NewValue}(ns, v)\rangle$ and either: (*i*) Z generated n, *i.e.*, $id = Z$; (*ii*) n is intended to be shared with Z, *i.e.*, $Z \in ns[\mathit{subst}(id)]$; or (*iii*) n is *old*, *i.e.*, for each principal x in $ns[\mathit{subst}(id)]$, σ contains an event of the form $\langle id', l', \mathrm{Old}(v')\rangle$ with $\mathit{subst}(id')(\mu) = x$ and $\mathit{subst}(v') = n$. Let $\mathit{learned}_Z(IK, \sigma, \mathit{subst}) = IK \cup \mathit{rcvd}_Z(\sigma) \cup \mathit{genvals}_Z(\sigma, \mathit{subst})$. 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[7]

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

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

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

[6] 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.

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

These definitions assume that the symmetric and public-key cryptosystems are perfect, *i.e.*, invulnerable to cryptanalytic attacks. Several researchers, using similar models of perfect cryptography, have defined similar functions to express the adversary's knowledge.

2.3 Syntax and Semantics of Correctness Requirements

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

Correspondence 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 satisfies a correspondence requirement $\langle a, a' \rangle$ iff, for all distinct pairs $\langle x, y \rangle$ of honest principals, if x executes a with argument t in a run with partner y, then previously y executed a' with argument t in a run with partner x.

Short-Term Secrecy Requirement. The Short-Term Secrecy Requirement expresses secrecy of genvals. An execution $\langle \sigma, \mathit{subst}, \mathit{lprot} \rangle$ satisfies the Short-Term Secrecy Requirement iff all genvals in $\mathit{known}_Z(IK, \sigma, \mathit{subst})$ are in $\mathit{genvals}_Z(\sigma, \mathit{subst})$.

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

Example. The Yahalom protocol satisfies 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 $\bigcup_{x \in \mathit{Name} \setminus \{Z, S\}} \{\mathit{key}(x, S)\}$.

3 Reduction

3.1 Static Restrictions

Primitive Variable Restriction. All variables in the protocol are of type Prim.

Shallow Ciphertext Restriction. The encryption height of the second argument of each Send and the argument of each Receive is at most 1.

Secrecy of Long-Term Keys Restriction. Long-term keys are not sent in messages, *i.e.*, in each Send statement, the operators *key*, *pubkey*, and *pvtkey* occur only in the second argument of the *encrypt* operator. Initially, Z does not know other principals' keys, *i.e.*, for $x \in \mathit{Name} \setminus \{Z\}$, IK does not contain $\mathit{key}(x, S)$ or $\mathit{pvtkey}(x)$.

Known Name Restriction. Z knows all names, *i.e.*, $\mathit{Name} \subseteq IK$.

Width of End$*$ Restriction. For each EndRespond and EndInit statement, the sum of the numbers of variables and primitive values in its argument is less than $|Prim \cap IK|$.

Lemma 1. Let Π be a protocol satisfying the Primitive Variable and Shallow Ciphertext Restrictions. For all executions of Π, every ciphertext sent or received by an honest principal has an encryption height of 1.

Proof: Straightforward.

Lemma 2. Let $\langle IK, PS\rangle$ be a protocol satisfying the Secrecy of Long-Term Keys Restriction. For all executions $\langle \sigma, subst, lprot\rangle$ of Π, $(known_Z(IK, \sigma, subst) \cap Key_{LT}) \subseteq IK$.

Proof: Straightforward.

3.2 Dependence Relations

Source of Ciphertext. Given an execution $\langle \sigma, subst, lprot\rangle$ of a protocol $\Pi = \langle IK, PS\rangle$ that satisfies the Primitive Variable Requirement, the *source* of each ciphertext c' in each message received by an honest principal is an index j such that $\sigma(j)$ is a Send event $\langle id, l, \text{Send}(x, t)\rangle$ containing the encryption that created c'. Let $\langle id', l', \text{Receive}(t')\rangle = \sigma(j)$ be a Receive event by an honest principal. Let $\langle id, l, \text{Send}(x, t)\rangle = \sigma(j-1)$ be the corresponding Send event. Let c' be a subterm of $t'[subst(id')]$ that is a ciphertext. Let $\{d\}_k = c'$. Suppose $id \neq Z$. Then the Primitive Variable Requirement and E4 in the definition of execution imply that the source of $\langle j, c'\rangle$ is $j-1$. Suppose $id = Z$. If $k \in known_Z(IK, \sigma(0..j-1, subst)$, then Z can perform the encryption to create c', so the source of $\langle j, c'\rangle$ is $j-1$. Otherwise, there must be a previous Send event $\langle id'', l'', \text{Send}(x'', t'')\rangle = \sigma(j'')$ such that $t''[subst(id'')]$ contains a subterm equal to c' (there might be multiple such Send events; for our purposes, it does not matter which one is chosen), and the source of $\langle j, c'\rangle$ is j''.

Dependence Between Events. Given an execution ex, we define dependence relations on events and runs in ex. These relations and their auxiliary functions are implicitly parameterized by the execution being considered. Informally, $\overset{lcl}{\rightarrow}$ captures local dependencies, *i.e.*, dependencies within a single run; $\overset{ct}{\rightarrow}$ captures dependencies induced by one run receiving a ciphertext sent by another run; and $\overset{rvl,n}{\rightarrow}$ captures dependencies induced by one run helping reveal to Z a genval n that Z uses in a message received by another run. For a genval n generated by an event $\langle id, l, \text{NewValue}(ns, v)\rangle$, the set $prins(n)$ of principals associated with n is defined by $prins(n) = ns[subst(id)]$. We say "helps reveal n" rather than "reveals n", because an Old event can contribute to revealing a genval n without actually revealing n (if $|prins(n)| > 1$).

$$\begin{array}{lcl}
e \overset{lcl}{\rightarrow} e' & \triangleq & e \text{ and } e' \text{ are events with the same run-id and } e \text{ precedes } e' \\
e \overset{ct}{\rightarrow} e' & \triangleq & e \text{ is a Send event by an honest principal and } e' \text{ is a Receive of a term containing a ciphertext} \\
 & & \text{whose source is } e \\
e \overset{rvl,n}{\rightarrow} e' & \triangleq & (e \text{ helps reveal } n) \wedge (e' \text{ rcvs } n \text{ from } Z)
\end{array} \tag{6}$$

where for a genval n,

$$\begin{array}{lcl} e \text{ helps reveal } n & \triangleq & \vee\, e \text{ is the NewValue event that generates } n \\ & & \vee\, Z \notin \mathit{prins}(n) \wedge (\exists x \in \mathit{prins}(n) : e \text{ is the first Old event by } x \text{ with argument } n) \\ e \text{ rcvs } n \text{ from } Z & \triangleq & \vee\, e \text{ is a Receive of a term containing } n \text{ unencrypted} \\ & & \vee\, e \text{ is a Receive of a term containing a ciphertext containing } n \\ & & \quad \text{and whose source is an event by } Z \end{array} \tag{7}$$

In the definition of $\xrightarrow{lcl}$, e need not immediately precede e'. The definition of "e helps reveal n" considers only NewValue and Old events, because the Short-Term Secrecy Requirement implies that only those two kinds of events help Z learn genvals,[8] and we use this relation only in contexts where the Short-Term Secrecy Requirement holds. Regarding the second disjunct in the definition of "e rcvs n from Z", note that a ciphertext contains terms used for the encryption key or in the data; for example, $\{A\}_{key(B,S)}$ contains the primitive values A and $key(B, S)$.

Replaceable Genvals. Roughly, a set evs of events depends on an event e if there exists an event e' in evs such that e precedes e' in some run, e is the source of a ciphertext received by e', e generates a genval received by e', or e helps reveal to Z a genval used by Z in a message received by e'. To avoid spurious dependencies, only genvals that are "irreplaceable" in R are considered in the last of these cases. Let evs be a set of events (implicitly, a subset of the events in the execution being considered) that is backwards-closed with respect to $\xrightarrow{lcl} \cup \xrightarrow{ct}$, *i.e.*, $e' \in evs \wedge e \; (\xrightarrow{lcl} \cup \xrightarrow{ct}) \; e' \Rightarrow e \in evs$. In other words, evs contains prefixes of some set R of runs. A genval n is *irreplaceable* in evs iff evs contains the NewValue event that generated n. If n is replaceable (*i.e.*, not irreplaceable) in evs, then n can be replaced with any primitive value in evs, and the events in evs would occur exactly as before, except for the effect of this substitution on the variable bindings of the runs in R. To see this, note that in evs, n is received only (*i*) unencrypted, (*ii*) in ciphertexts produced by Z, or (*iii*) in ciphertexts produced by other events in evs. Regarding (*i*) and (*ii*), Z can substitute any other primitive value for n in those messages (this might require inserting into the execution a Receive event by Z to intercept the message and an immediately following Send event by Z to send the modified message). Now we show that (*iii*) does not prevent replacement of n in evs. The proof is by induction on evs ordered by $(\xrightarrow{lcl} \cup \xrightarrow{ct})^*$, where for a binary relation $\rightarrow$, $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$. For the base case, let e be a minimal (with respect to $(\xrightarrow{lcl} \cup \xrightarrow{ct})^*$) event in evs that is the source of a ciphertext c containing n and received by some Receive event $e' \in evs$. Let id and id' be the runs containing e and e', respectively. In the prefix of id up to and including e, n is received only in contexts (*i*) and (*ii*), so n can be replaced in that prefix and hence in ciphertext c and hence in the prefix of id' up to (not including) the earliest Receive event in id' after e'. The induction hypothesis is that n can be replaced in a subset S of evs that is backwards-closed with respect to $\xrightarrow{lcl} \cup \xrightarrow{ct}$. For the step case, let event e be a minimal event in $evs \setminus S$ that is the source of a ciphertext c containing n and received by a Receive event $e' \in evs$. Let id and id' be the runs containing e and e', respectively. By similar reasoning as in the base case, n can be replaced in c and hence in the prefix of id' up to (not including) the earliest Receive event in id' after e'.

[8] After NewValue and Old statements reveal a genval n to Z, Send statements can send n unencrypted or encrypted with a key known to Z without violating the Short-Term Secrecy Requirement. Such Send events can be ignored here, because Z needs to learn a genval only once to use it in subsequent messages.

One can define simpler dependence relations on runs that implicitly consider all genvals to be irreplaceable. In that case, there would be more dependencies between runs, and fewer protocols would satisfy the cluster restriction for a given cluster size (see Section 3.3). Thus, the notion of irreplaceability makes our reduction applicable to more protocols and sometimes reduces the size of the state space that needs to be explored.

Support. Let $\mathit{irrep}(\mathit{evs})$ denote the set of genvals that are irreplaceable in evs. For an execution ex and a set evs of events, the set of events on which events in evs depend is denoted $\mathit{support}(\mathit{ex}, \mathit{evs})$ and defined by

$$\begin{aligned}\mathit{support}(\mathit{ex}, \mathit{evs}) \triangleq{} & \text{the least set } \mathit{evs}' \text{ such that} \\ & \mathit{evs} \subseteq \mathit{evs}' \;\wedge\; (\forall e \in \mathit{ex}, e' \in \mathit{evs}', n \in \mathit{irrep}(\mathit{evs}') : e\ (\xrightarrow{lcl} \cup \xrightarrow{ct} \cup \xrightarrow{rvl,n})\ e' \Rightarrow e \in \mathit{evs}')\end{aligned} \tag{8}$$

To see that this is well-defined, note that if evs_1 and evs_2 satisfy the predicate in (8), then so does $\mathit{evs}_1 \cap \mathit{evs}_2$. Let $\mathit{events}(\mathit{ex}, \mathit{id})$ denote the set of events of run id in ex. For convenience, we overload $\mathit{support}$ by defining

$$\mathit{support}(\mathit{ex}, \mathit{id}) \triangleq \mathit{support}(\mathit{ex}, \mathit{events}(\mathit{ex}, \mathit{id})). \tag{9}$$

3.3 Dynamic Restriction

Clusters. A *cluster size* is a function from the set of local protocols to $\{1, 2, \ldots\}$. For a cluster size f, a set evs of events is an f*-cluster* in an execution ex iff for each local protocol P, evs contains events from at most $f(P)$ runs of P in ex.

Cluster Restriction. An execution ex satisfies the *cluster restriction* for cluster size f, abbreviated Clus(f), iff for every run id in ex, $\mathit{support}(\mathit{ex}, \mathit{id})$ is an f-cluster.

3.4 Protocol Transformation

Most protocols of interest do not directly satisfy the Primitive Variable Restriction, but most of them do satisfy it after a simple correctness-preserving transformation is applied.

Transformation ElimFwdVar. Eliminate all occurrences of forwarding variables. A *forwarding variable* is a variable whose defining occurrences are in a Receive and all of whose occurrences are unencrypted. If this transformation reduces the second argument of a Send or the argument of a Receive to the "empty term", then that statement can be deleted.

Lemma 3. Let Π be a protocol. Let Π' be a protocol obtained from Π by applying transformation ElimFwdVar. Let ϕ be a correspondence or secrecy requirement. Π satisfies ϕ iff Π' satisfies ϕ.

Proof: Straightforward.

Example. Applying transformation ElimFwdVar to the Yahalom protocol (2) eliminates variable X from lines 2 and 4 of P_I. Let $\Pi_{Y'}$ denote the resulting protocol.

3.5 Dependence Width

Let s be a Receive statement Receive(t) executed during a run id. Informally, the *dependence width* of s is the number of other runs on which direct (*i.e.*, not transitive) dependencies can be created by executing s. This concept is used in Theorem 1 to bound the number of runs involved in a violation of the Cluster Restriction. We assume that the protocol satisfies the static restrictions in Section 3.1.

Consider a ciphertext $\{t'\}_k$ in t. Let c be the received ciphertext that matches $\{t'\}_k$. If k is $key(\mu, S)$, then the Secrecy of Long-Term Keys Restriction implies that Z does not know k, so source of c is an honest principal, and receiving c creates a dependence on the source of c. Otherwise, the source of c might be Z, so receiving it creates dependencies on events (if any) that revealed primitive values in c to Z. The Known Name and Secrecy of Long-Term Keys Restrictions imply that Z does not learn names or long-term keys during an execution, so occurrences of these names and key terms in t do not (by themselves) create dependencies. Genvals in c must match variables in t. The number of such genvals and the number of events that help reveal them to Z are bounded as described below.

Consider a use of a variable v. Suppose this occurrence of v is unencrypted or is encrypted and the received ciphertext is from Z. If v is bound to a genval n, then (when s is executed) this occurrence of v creates direct dependencies on the events that help reveal n to Z. If v is defined in a statement of the form NewValue(ns, v), then the number of such events not in id is at most $|ns|$. If v is defined in a previous Receive statement, then the number of such events is at most $|ns| + 1$, because the NewValue statement that generated n might be in a run other than id. If v is not bound to a genval, then the Primitive Variable Restriction implies that v is bound to a name or long-term key. The Known Name and Secrecy of Long-Term Keys Restrictions imply that Z does not learn names or long-term keys during an execution. Thus, this occurrence of v does not (by itself) create any dependencies. Note that reserved variables are always bound to names. Suppose v appears encrypted and the received ciphertext is from an honest principal. This occurrence of v creates a direct dependence on the source of the ciphertext; this dependence are "charged" to the *encrypt* operator enclosing v.

Consider a defining occurrence of a variable v. Note that all occurrences of v is s are uses, or all occurrences of v is s are defining occurrences. Suppose all defining occurrences of v are in ciphertexts with key $key(\mu, S)$. These occurrences must match parts of ciphertexts whose sources are events by honest principals; the resulting dependences on the sources of those ciphertexts are "charged" to the *encrypt* operator enclosing v. Suppose some defining occurrence of v is unencrypted or is in a ciphertext with key not equal to $key(\mu, S)$. If v gets bound to a genval, then these occurrences of v create direct dependencies on the events that help reveal n to Z; there are at most $maxss(\Pi) + 1$ such events, where $maxss(\Pi)$ ("maximum sharing size") is the maximum cardinality of the first argument of NewValue statements in Π. If v does not get bound to a genval, then it must be bound to a name or long-term key, so it creates no dependencies.

Based on the above arguments, the dependence width of the argument t of a Receive statement in a protocol Π satisfying the static restrictions in Section 3.1 is defined as follows. Intuitively, for each ciphertext c in t, we take the maximum of one dependence (on the source of c, if c's source is an honest principal) and the number of events needed to help reveal to Z the genvals bound to variables in c. However, if a variable v possibly bound to a genval n occurs in multiple ciphertexts t that might match messages from Z, the events that help reveal n are counted only once; this is why dependence width is not defined by a simple structural recursion. Let $ctexts(t)$ be the set of ciphertexts appearing in t. Let $vars(t)$ be the set of variables appearing in t.

$$
\begin{aligned}
dw(t) &\triangleq \sum_{c \in ctexts(t)} dw_C(c,t) \;+ \sum_{v \in vars(t)} dw_V(v,t) \qquad (10)\\
dw_C(\{t'\}_k, t) &\triangleq \begin{cases} 0 & \text{if } k \text{ is not } key(\mu,S)\text{, and } \{t'\}_k \text{ contains a variable } v \text{ such that } dw_V(v,t) > 0. \\ 1 & \text{otherwise} \end{cases}\\
dw_V(v,t) &\triangleq \begin{cases} |ns| & \text{if there is a use of } v \text{ in } t \text{ not in a ciphertext with key } key(\mu,S)\text{,} \\ & \text{and } v \text{ is defined in a statement NewValue}(ns,v) \\ maxss(\Pi)+1 & \text{if there is a use of } v \text{ in } t \text{ not in a ciphertext with key } key(\mu,S)\text{,} \\ & \text{and } v \text{ is defined in a previous Receive statement} \\ maxss(\Pi)+1 & \text{if there is a defining occurrence of } v \text{ in } t \text{ in a ciphertext} \\ & \text{with a key not equal to } key(\mu,S) \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

For a protocol Π, define the dependence width $dw(\Pi)$ of Π to be the maximum dependence width of the Receive statements in Π.

Examples. Consider the transformed Yahalom protocol. The dependence width of P_I, line 2, is 1; of P_R, line 0, is 0; of P_R, line 4, is 6; of P_S, line 0, is 6. Thus, $dw(\Pi_{Y'}) = 6$. For the simple protocol Π_U, $dw(\Pi_U) = 2$. The dependence width of the transformed Otway-Rees protocol is 6, because the Receive in line 0 of P_S has a dependence width of 6. The dependence width of the Needham-Schroeder public-key protocol is 5, because the Receive in line 2 of P_I has a dependence width of 5.

3.6 Reduction for Cluster Restriction and Short-Term Secrecy Requirement

Run-Bounds. A run-bound β bounds the number of runs of each local protocol in an execution. For a protocol Π, for each local protocol P of Π, $\beta(\Pi)(P)$ is a bound on the number of runs of P in an execution. $\Pi|_\beta$ denotes a protocol whose executions are the executions of Π that satisfy the bounds imposed by β. For example, if $\beta(\Pi)(P_I) = 3$, and ex is an execution of Π containing at most 3 run-ids id such that $lprot(id) = P_I$, then ex is an execution of $\Pi|_\beta$.

For a cluster size f, define a run-bound β_f by

$$\beta_f(\Pi)(P) \triangleq (dw(\Pi)+1)f(P). \qquad (11)$$

Theorem 1. Let Π be a protocol satisfying the first four static restrictions in Section 3.1. Let f be a cluster size. Π satisfies Clus(f) and the Short-Term Secrecy Requirement iff $\Pi|_{\beta_f}$ satisfies Clus(f) and the Short-Term Secrecy Requirement.

Proof: The forward direction ($\Rightarrow$) of the "iff" follows immediately from the fact that the set of executions of $\Pi|_{\beta_f}$ is a subset of the set of executions of Π. For the reverse direction ($\Leftarrow$), we prove the contrapositive, *i.e.*, we suppose there exists an execution $ex = \langle \sigma, subst, lprot \rangle$ of Π violating Clus(f) or the Short-Term Secrecy Requirement and show that $\Pi|_{\beta_f}$ violates Clus(f) or the Short-Term Secrecy Requirement. We abuse notation and write $ex(0..j)$ to denote $\langle \sigma(0..j), subst, lprot \rangle$. Let d be the largest d such that $ex(0..d)$ satisfies Clus(f) and the Short-Term Secrecy Requirement. Such a d exists because an execution of length 0 satisfies these properties. Let id be the run containing the event that causes the violation, *i.e.*, $\langle id, l, s\rangle = \sigma(d+1)$. $ex(0..d)$ satisfies Clus(f), so $support(ex(0..d), id)$ is an f-cluster.

The event $\sigma(d+1)$ may create direct dependencies of id on other runs. Let R_1 be the set of such runs. Only Receive events create dependencies on other runs. As argued in Section 3.5, the dependence width of a Receive statement bounds the number of direct dependencies on other runs created by that Receive statement. Thus, $|R_1| \leq dw(\Pi)$.

$ex(0..d)$ satisfies Clus(f), so for each $id' \in R_1$, $support(ex(0..d), id')$ is an f-cluster. Let

$$R = \bigcup_{id' \in R_1 \cup \{id\}} support(ex(0..d), id').$$

Note that R contains events from at most $dw(\Pi) + 1$ f-clusters.

Consider the effect of removing from $ex(0..d+1)$ the events not in R. By definition of R and *support*, events in R do not receive ciphertexts whose sources are events not in R. Consider ciphertexts c received by events in R and whose sources are events by Z; specifically, consider when Z learned the terms used in such a ciphertext c. Lemma 1 implies that Z does not use (learned) ciphertexts to construct c. Tuples do not need to be considered explicitly here, because Z knows a tuple exactly when Z knows the components of the tuple. So, it suffices to consider when Z learned the primitive values in c. Π satisfies the Secrecy of Long-Term Keys Restriction, so Lemma 2 implies that Z does not learn long-term keys. Π satisfies the Known Name Restriction, so Z does not learn names. Thus, all primitive values learned by Z during an execution are genvals. Removal of events not in R can potentially affect events in R by changing when Z learns genvals used in messages received by events in R. By definition of $\overset{rvl,n}{\to}$ and *support*, all events that help Z learn irreplaceable genvals are included in R, so removing events not in R does not affect when Z learns those genvals. Events that help Z learn genvals that are replaceable in R might be removed. As argued in Section 3.2, replaceable genvals can be replaced with any primitive value in the events in R, and the result is an execution in which events in evs occur exactly as before, except for the effect of this substitution on the variable bindings of the affected runs. This replacement eliminates the dependence of events in R on the events that helped reveal the replaceable genvals to Z.

Consider primitive values received unencrypted by events in R. Π satisfies the Secrecy of Long-Term Keys Restriction, so Lemma 2 implies that long-term keys are never received unencrypted. Π satisfies the Known Name Restriction, so receiving an unencrypted name does not create dependencies on other events. Consider a genval n received unencrypted by an event e in R. If n is irreplaceable in R, then removing events not in R does not affect when Z learns n and hence does not affect e. If n is replaceable in R, then events that help reveal n to Z might be removed, but by the same argument as in the previous paragraph, n can be replaced with any primitive value in R, thereby eliminating the dependence of R on the events that help reveal n to Z.

Let ex' be $ex(0..d+1)$ with events not in R removed and with genvals that are replaceable in R replaced with some primitive value known to Z (*e.g.*, some name). It follows from the above that ex' is an execution of $\Pi|_{\beta_f}$. It remains to show that ex' violates Clus(f) or the Short-Term Secrecy Requirement.

Suppose $ex(0..d+1)$ violates Clus(f). By construction, every event in $support(ex(0..d+1), id)$ is included in ex'. By definition, calculation of *support* is not affected by replacement of replaceable genvals with other primitive values. Thus, the support of $events(id)$ is the same in $ex(0..d+1)$ and ex', so ex' violates Clus(f).

Suppose $ex(0..d+1)$ violates the Short-Term Secrecy Requirement; thus, $\sigma(d+1)$ reveals to Z a genval n

not previously known to Z. Let e be the event that generates n. We show that $e \in R$, *i.e.*, n is irreplaceable in R; this and the fact that removing events not in R cannot make n old imply that ex' violates the Short-Term Secrecy Requirement. To see that $e \in R$, consider some sequence of messages by which n flowed from the run containing e to id. Every message in the sequence must be a ciphertext, because if n were sent unencrypted, Z would know n before $\sigma(d+1)$ occurred, a contradiction. Thus, $e' \ (\xrightarrow{lcl} \cup \xrightarrow{ct})^* \ \sigma(d)$. From the definition of *support*, it follows that $e \in R$. ∎

3.7 Reduction for Long-Term Secrecy and Correspondence Requirements

For a cluster size f, define a run-bound β_f^1 by: $\beta_f^1(\Pi)(P) = f(P)$. Thus, an execution of $\Pi|_{\beta_f^1}$ contains at most one f-cluster.

Theorem 2. Let Π be a protocol satisfying the static restrictions in Section 3.1. Let f be a cluster size. Let ϕ be a long-term secrecy or correspondence requirement . Suppose $\Pi|_{\beta_f}$ satisfies Clus(f) and the Short-Term Secrecy Requirement. Π satisfies ϕ iff $\Pi|_{\beta_f^1}$ satisfies ϕ.

Proof: The forward direction ($\Rightarrow$) of the "iff" follows immediately from the fact that the set of executions of $\Pi|_{\beta_f}$ is a subset of the set of executions of Π. For the reverse direction ($\Leftarrow$), we prove the contrapositive, *i.e.*, we suppose there exists an execution $ex = \langle \sigma, subst, lprot \rangle$ of Π violating ϕ and show that $\Pi|_{\beta_f}$ violates ϕ. Let d be the largest d such that $ex(0..d)$ satisfies ϕ. Let $\langle id, l, s \rangle = \sigma(d+1)$. Let $R = support(ex(0..d+1), id)$. Theorem 1 implies that Π satisfies Clus(f) and the Short-Term Secrecy Requirement, so R is an f-cluster. Let ex' be $ex(0..d+1)$ with events not in R removed and with genvals that are replaceable in R replaced with a primitive value known to Z. By the same reasoning as in the proof of Theorem 1, it follows that ex' is an execution of $\Pi|_{\beta_f}$. It remains to show that ex' violates ϕ.

Suppose ϕ is a Long-Term Secrecy Requirement. Let $t \in \phi$ be a ground term revealed to Z by $\sigma(d+1)$. Removing events not in R does not eliminate this event and therefore does not destroy the violation of ϕ. t does not contain genvals, so replacing replaceable genvals with other primitive values does not destroy the violation of ϕ. Thus, ex' violates ϕ.

Suppose ϕ is a Correspondence Requirement. Let $\langle a, a' \rangle = \phi$. $\sigma(d+1)$ must have the form $\langle id, l, a(t) \rangle$. Removing events not in R does not destroy the violation of ϕ. We need to argue that there exists a substitution of primitive values known to Z for replaceable genvals of R that does not destroy the violation of ϕ; some substitutions might destroy the violation of ϕ by causing the argument of some a' event in $\sigma(0..d)$ to become equal to $t[subst(id)]$. A substitution that replaces all replaceable genvals with some primitive value that does not appear in $t[subst(id)]$ cannot produce such equalities. Thus, it suffices to argue that Z knows such a primitive value. The Primitive Variable and Width of End$*$ Restrictions imply that the number of primitive values in $t[subst(id)]$ is less than $|Prim \cap IK|$, so Z knows such a primitive value. ∎

4 Discussion

Applicability. Applying Transformation ElimFwdVar to the Otway-Rees, Needham-Schroeder shared-key, corrected Needham-Schroeder public-key, and Yahalom protocols yields protocols satisfying the static restrictions in Section 3.1. It seems that some interesting protocols satisfy Clus(f) for cluster sizes f with 1 or 2 runs per local protocol. (Many incorrect or inefficient protocols do not satisfy this restriction, but that is no cause for concern.) Define cluster size f_1 by: $f_1(P) = 1$ for all P. A tool that conveniently checks the Cluster Requirement is not yet available, and those checks might require considerable CPU cycles. If such a tool and sufficient CPU cycles were available, then the appropriate cluster size for a given (correct) protocol could be determined by starting with a cluster size f equal to f_1 and gradually increasing the cluster size until the tool reports that $\Pi|_{\beta_f}$ satisfies $\mathrm{Clus}(f)$. In the meantime, I proved that the transformed Yahalom protocol satisfies $Clus(f_Y)$, where $f_Y(P_I) = 1$, $f_Y(P_R) = 2$, and $f_Y(P_S) = 1$, that the transformed Otway-Rees protocol satisfies $Clus(f_{OR})$, where $f_{OR}(P_I) = 1$, $f_{OR}(P_R) = 1$, and $f_{OR}(P_S) = 2$,[9] and that the corrected Needham-Schroeder public-key protocol and the simple unilateral protocol (1) satisfy $\mathrm{Clus}(f_1)$.

Bound on Number of Operations Performed by Z. Z builds messages using encryption and tupling. Recall from Lemma 1 that all ciphertexts sent or received by honest principals have an encryption height of 1. The Primitive Variable Restriction implies that the number of *pair* operators in messages sent or received by honest principals is bounded by the number of concatenation operators in a Send or Receive statement in the protocol. Thus, we can easily obtain a bound on the number of useful message-building operations that Z can perform in an execution of $\Pi|_\beta$. We also need to bound the number of NewValue events performed by Z. Provided the Width of End$*$ Restriction holds and $|Prim \cap IK| \geq 1$, we can prohibit Z from executing NewValue events. This restriction on Z preserves all correctness requirements and dynamic restrictions. This follows from an argument similar to the argument in the proof of Theorem 2 that replaceable genvals can be replaced with other primitive values without destroying violations of Correspondence Requirements.

Implementing LAP. To correctly implement the LAP semantics, the run-time system needs to ensure that variables of type Prim get bound only to primitive values. In an implementation, how can ciphertexts and tuples be distinguished from primitive values? For example, does it suffice to adopt a data format such that the size (in bytes) of a ciphertext or tuple never equals the size of a primitive value? If so, this requirement seems fairly easy to satisfy, since there are only a few kinds of primitive values, and they typically have fixed sizes. In an implementation, the encryption and decryption functions must incorporate an integrity check based on a message digest. Also, messages must contain formatting information that enables a recipient to correctly split a tuple into its components; a header specifying the starting offset of each field suffices. Proving a (probabilistic) refinement relationship between such an implementation and the high-level semantics in Section 2.2 is a non-trivial undertaking.

Reducing the Bound by Protocol Transformation. Consider the transformation: Replace each Receive statement of the form $\mathrm{Receive}(t_1 \cdot t_2 \cdot \ldots \cdot t_n)$ for $n > 2$ with the sequence of statements $\mathrm{Receive}(t_1)$; $\mathrm{Receive}(t_2); \cdots; \mathrm{Receive}(t_n)$. This transformation preserves correctness requirements and static restrictions and sometimes reduces the dependence width and therefore β_f. However, it can introduce violations of the Cluster Requirement. Thus, if the transformed protocol violates the Cluster Requirement (for some cluster size), then the untransformed protocol should be considered instead.

Feasibility of Model-Checking. We have succeeded in making the problem finite-state. However, for most protocols, the bound on the number of runs is probably too large for state-space exploration to be feasible with current technology. For the simple unilateral protocol Π_U, executions with up to 6 runs (3 of each local protocol) need to be checked; this is feasible. For the corrected Needham-Schroeder public-key

[9] The fact that $f_{OR}(P_S) > 1$ reflects the fact that the Otway-Rees protocol does not ensure key agreement [THG98].

protocol, the dependence width is 5, so executions with up to 12 runs (6 of each local protocol) need to be checked; I believe this is currently infeasible. For the transformed Yahalom protocol $\Pi_{Y'}$, executions with up to 28 runs (7 each of P_I and P_S, and 14 of P_R) need to be checked; this is currently infeasible. Symmetry reductions and partial-order reductions can reduce the cost of this. Future work includes developing a more careful analysis or stronger restrictions, in order to justify smaller bounds.

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.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 24(4):181–185, October 1985.

[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.

[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.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[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.

[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.

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

[Lam93] Leslie Lamport. How to write a long formula. Technical Report SRC-119, Digital Equipment Corporation, Systems Research Center, 1993. Available via http://www.research.digital.com/SRC/personal/Leslie_Lamport/proofs/proofs.html.

[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.

[THG98] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 18th IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1998.

[WL93] 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.

[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.