# Knowledge Flow Analysis for Security Protocols

Emina Torlak, Marten van Dijk, Blaise Gassend, Daniel Jackson, and Srinivas Devadas
{emina, marten, gassend, dnj, devadas}@mit.edu

August 8, 2005

**Abstract**

Knowledge flow analysis offers a simple and flexible way to find flaws in security protocols. A protocol is described by a collection of rules constraining the propagation of knowledge amongst principals. Because this characterization corresponds closely to informal descriptions of protocols, it allows a succinct and natural formalization; because it abstracts away message ordering, and handles communications between principals and applications of cryptographic primitives uniformly, it is readily represented in a standard logic. A generic framework in the Alloy modelling language is presented, and instantiated for two standard protocols, and a new key management scheme.

## 1 Introduction

One area of major successes for formal methods has been the verification of security protocols. A number of specialized tools have been developed in the last decade that have exposed subtle flaws in existing protocols (see, e.g. [12; 29]). For the most part, however, these tools have been used by the researchers that developed them, and less attention has been paid to usability issues.

This paper presents a new approach to formulating and checking cryptographic protocols. It does not enable any new form of analysis. Instead, it makes verification more accessible to the designers of protocols. Its key contribution is a new characterization of these protocols that is both closer to how designers conceive them, and amenable to a more direct encoding in standard first-order logic. This more direct encoding allows existing tools to be applied as black boxes without modification; it requires no tweaking of parameters or issuing of special directives by the user. Moreover, because the semantic gap between informal descriptions of protocols and their formalization is smaller, there are fewer opportunities for errors to creep in.

In this paper, the Alloy modeling language is used to record the details of the protocol and its security goals, and the Alloy Analyzer is used to find flaws. The approach, however, requires no special features of Alloy or its analysis, and could be applied in the context of any formal method based on first-order logic. Its simplicity suggests that it may be useful in teaching; indeed, using the approach, we have explained cryptographic protocols to undergraduates who have had only a few weeks of experience in formal methods.

Our approach, which we call *knowledge flow analysis*, gives a uniform framework for expressing the actions of principals, assumptions on intruders, and properties of cryptographic primitives. The dynamic behaviour of the protocol is described by an initial state of knowledge, and a collection of rules that dictate how knowledge may flow amongst principals. A state is given by a relation mapping principals to the values they know; the allowable knowledge flows can thus be succinctly described as a standard transition relation on knowledge states, written as a constraint.

This simple setup allows us to model a range of intruder capabilities and to detect replay, parallel session, type flaw, and binding attacks. We have applied it to both symmetric and public-key cryptography under the Dolev-Yao [16] approach. The modeling framework itself is more general, however, and can be extended to include the properties of cryptographic primitives [10; 14; 33; 42] and an unbounded number of sessions with bounded messages [11].

This approach grew out of an effort to check a new cryptographic scheme [20; 21]. Knowledge flow analysis described here was the final result of a series of incremental attempts at formalizing and checking the protocol using the Alloy language and tool. This process helped crystallize our intuitions, and drew out a number of important assumptions. The final analysis, although only performed over a finite domain, actually establishes the correctness of the protocol for unbounded instantiations because of a special property of this protocol. The Alloy models developed for this case study were generalized into a simple framework that was subsequently applied to some standard protocols, such as Needham-Schroeder [36] and Otway-Rees [40].

The contributions of this paper are:

1. the knowledge flow formalism, which characterizes the dynamic behaviour of a cryptographic protocol in terms of the increasing knowledge of the principals, avoiding the need to impose an explicit ordering on messages;

2. a realization in the Alloy modelling language as a generic framework with a library of primitives that can be easily instantiated for a variety of protocols;

3. soundness and completeness results that guarantee that (1) any counterexample generated by the analyzer to a security theorem is legitimate, and not an artifact of the modelling framework, formalism or analysis; and (2) that if a counterexample exists involving any number of message exchanges and any number of steps, it will be found, so long as the number of parallel sessions is within a prescribed bound;

4. case study applications of the approach to two well-known protocols, one of which (Needham-Schroeder) is explained in detail, and to a new key management scheme based on controlled physical random functions [20; 21].

Section 2 explains the key intuitions underlying the approach, using Needham Schroeder as an example. Section 3 shows the complete formalization of this example, including the statement of the security goal, and a discussion of the counterexample corresponding to the well-known attack. Section 4 gives a mathematical summary of the approach without reference to any particular modeling language that might serve as a basis for implementations in other tools, and which makes precise the assumptions underlying the model. The paper closes with an evaluation and a discussion of related work.

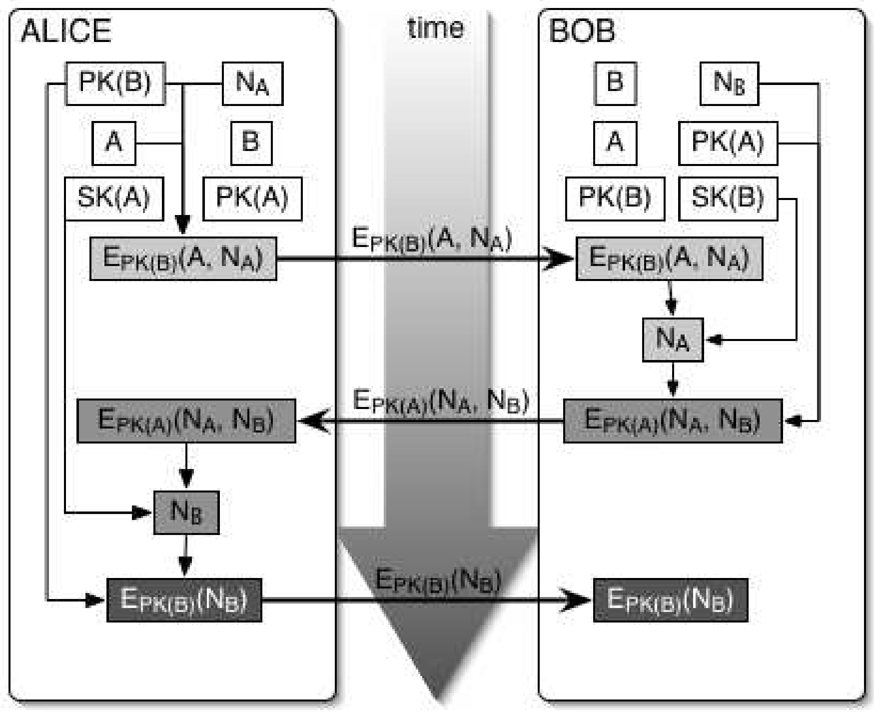


Figure 1: Knowledge Flow in Needham-Schroeder Protocol

## 2 Knowledge Flow Basics

The key idea behind knowledge flow analysis is the observation that, at the most basic level, the purpose of a security protocol is to distribute knowledge among its legitimate participants. A protocol is flawed if it allows an intruder to learn a value that is intended to remain strictly within the legitimate principals' pool of knowledge. To gain more intuition about knowledge flows in security applications, consider the Needham-Schroeder Public Key Protocol [36] shown in Figure 1.

We have two principals, Alice and Bob, each of whom has an initial pool of knowledge represented with white boxes. Alice's initial knowledge, for example, consists of her own public/private key pair $PK(A)$/$SK(A)$, identity $A$, nonce $N_A$, and Bob's public key $PK(B)$ and identity $B$. The purpose of the protocol is to distribute the nonces between Alice and Bob in such a way that the following conditions hold at the end: (1) Alice and Bob both know $N_A$ and $N_B$, and (2) no other principal knows the nonces.

To initiate the protocol, Alice first expands her pool of knowledge to include $E_{PK(B)}(A, N_A)$, an encryption of her identity and nonce with Bob's public key. She then sends the cipher to Bob who decrypts it using his private key, $SK(B)$. At the end of the first step of the protocol, each principal's knowledge has increased to include the values in light gray boxes. Bob performs the second step of the protocol by adding $E_{PK(A)}(N_A, N_B)$ to his current knowledge and sending the cipher to Alice. She uses her private key to decrypt Bob's message and extract $N_B$. By using $N_B$ and $PK(B)$, Alice can set up an authenticated and private channel with Bob as is done during the final step of the protocol in which Alice creates $E_{PK(B)}(N_B)$ and forwards it to Bob. Both Alice and Bob now know the two nonces and share all other knowledge except their secret keys.

Following the flow of knowledge in the Needham-Schroeder protocol provides a crucial insight underlying our analysis method. Namely, a principal can learn a value in one of three ways; he can

- *draw* the value at the start,
- *compute* it using his current knowledge, or
- *learn* it by communication.

Our analysis treats the latter two ways of obtaining knowledge as equivalent. Specifically, we can think of Alice's computing $E_{PK(B)}(A, N_A)$ as her learning it from a principal called *Encryptor* whose initial pool of values includes all possible ciphers: Alice sends the tuple $(PK(B), (A, N_A))$ to *Encryptor* who responds by sending back the encryption of $(A, N_A)$ with $PK(B)$.

Treating cryptographic primitives as principals allows us to consider the total pool of knowledge to be *fixed*. That is, the set of all values before and after the execution of a security protocol is the same; the only difference is the distribution of those values among the principals. Since we assume that principals never forget values, the set of principals who know a value at the end of a protocol session subsumes the set of principals who drew the value at the beginning.

The goal of analyzing knowledge flows in a protocol is to verify that particular values never leak out of the honest participants' pool of knowledge. In other words, *we are interested in analyzing the flow of knowledge from an intruder's perspective*. This observation allows us to make sound simplifying assumptions that drastically reduce the effort needed to formalize a protocol in terms of knowledge flows:

- We need not encode the flows of knowledge among the honest principals, such as the flow which allows Alice to learn $E_{PK(A)}(N_A, N_B)$ from $Encryptor$. Rather, we may assume that each honest principal draws all values in the total knowledge pool and specify protocols solely in terms of the intruders' knowledge flows (sections 4.1 and 4.2).
- We may model all adversaries, including the untrusted public network, with a single opponent whom we call $Oscar$. The soundness of this approach is formally proved in section 4.3. Intuitively, the approach makes sense if we note that the potential adversaries will be most effective when they collaborate and share knowledge among themselves. Hence, we can replace the (collaboration of) adversaries with a single principal who possesses all their knowledge, without excluding any intrusion scenarios.

In our example, the flow of knowledge from the intruder's perspective starts with the protocol initialization message $E_{PK(B)}(A, N_A)$, since Oscar needs no prior knowledge to learn the first cipher that Alice sends to Bob. In general, because Oscar includes the untrusted public network, he learns the first message of the protocol for free, regardless of who its intended recipient and sender are:

$$\forall_{p \in \{a,b\}, p' \in \{a,b\} \cup O} \, [\emptyset \rightarrow E_{PK(p')}(I(p), N(\epsilon, I(p)))]. \tag{1}$$

The variables $a$ and $b$ denote the honest principals (Alice and Bob), and the set $O$ stands for Oscar. The notation $N(\epsilon, I(p))$ represents the nonce that the nonce primitive $N$ generated for the principal identified by $I(p)$ using the random value $\epsilon$ as the seed. For example, Alice's identity is $I(a) = A$ and Alice's nonce is $N(\epsilon, I(a)) = N_A$. The empty set means that Oscar does not need prior knowledge to learn $E_{PK(p')}(I(p), N(\epsilon, I(p)))$.

Once his pool of knowledge includes $E_{PK(B)}(A, N_A)$, Oscar learns the corresponding response, $E_{PK(A)}(N_A, N_B)$. More generally[1],

$$\forall_{p' \in \{a,b\}, p \in \{a,b\} \cup O, v \in V} [c \rightarrow E_{PK(p)}(v, N(c, I(p')))] \quad \text{where } c = E_{PK(p')}(I(p), v). \tag{2}$$

The variable $V$ denotes the set of all values, or the fixed pool of knowledge. Note that our formalization constrains the seed of Bob's nonce to be Alice's initialization message. This is needed to establish that Bob's nonce was generated in the context of the protocol session started by Alice with $E_{PK(B)}(A, N_A)$. The resulting correspondence between the nonces prevents our analysis from sounding false alarms when Oscar legitimately obtains two nonces from Alice and Bob by running a valid protocol session with each.

Oscar learns the final message, $E_{PK(B)}(N_B)$, as a consequence of knowing $E_{PK(A)}(N_A, N_B)$. Formally,

$$\forall_{p \in \{a,b\}, p' \in \{a,b\} \cup O, v \in V} \left[ \{E_{PK(p)}(N(\epsilon, I(p)), v))\} \rightarrow E_{PK(p')}(v) \right]. \tag{3}$$

# 3 Example

The Needham-Schroeder protocol is vulnerable to a parallel session attack discovered by Gavin Lowe [28]. This section presents a knowledge flow analysis of the protocol that reproduces Lowe's results, and gives a flavor of the expressiveness and simplicity of our method. We have encoded the knowledge flows in the Alloy modelling language [26] and used the Alloy Analyzer [25] to find the attack. However, the modelling pattern presented here is applicable to any first-order logic with relations and transitive closure.

## 3.1 Encoding Basic Entities and Relations

The basic components of a knowledge flow model are the sets $Principal$ and $Value$, and the relations $draws$, $learns$, and $knows$ (Model Excerpt 1).

The set $Principal$ includes all principals in a protocol – the legitimate protocol participants, represented by the subset $HonestUser$, and the intruders, represented by $Oscar$. The set $Value$ models the fixed pool of knowledge on which a protocol operates. We distinguish between $AtomicValue$s, which are uninterpreted, and $CompositeValue$s, which may consist of other values and are learned by communicating with cryptographic primitives. In the example from Figure 1, Alice and Bob are members of $HonestUser$;

[1] We use the parameter $v$ in $c$ instead of $N(\epsilon, I(p))$ because $p'$, the recipient of $c$, cannot conclusively determine that $v$ is, in fact, the nonce $N(\epsilon, I(p))$.

**Model Excerpt 1** Generic Model of Principals and Values

```
1  module kf/basicdeclarations
2
3  abstract sig Value {}
4  sig CompositeValue extends Value {}
5  sig AtomicValue extends Value {}
6
7  abstract sig Principal {
8   draws: set Value,
9   owns: set draws
10 }{ no owns & (Principal - this).@owns }
11
12 sig HonestUser extends Principal {
13 }{ draws = Value }
14
15 one sig Oscar extends Principal {
16  knows: set Value,
17  learns: knows->knows
18 }{ no ^learns & iden }
19
20 pred InitialKnowledge() {
21  no CompositeValue & Oscar.draws }
22
23 pred FinalKnowledge() {
24  all v: Value |
25   v in (Oscar.draws).*(Oscar.learns) iff
26   v in Oscar.knows }
```

$Value$ consists of the union of values enclosed in the boxes ‘Alice’ and ‘Bob’; the identifiers $A$ and $B$ are $AtomicValue$s, and the ciphers are $CompositeValue$s.

The relation $draws$ (line 8) maps each principal to the set of values known by that principal at the beginning of the protocol. For example, both Alice and Bob draw Alice’s identity $A$ at the start of the protocol session shown in Figure 1. The declaration of $owns$ (line 9) together with the constraint on line 10 relate a principal to the set of drawn values which uniquely identify him. Bob, for instance, $owns$ his identity, $B$, even though both he and Alice draw it.

The field $knows$ (line 16) defines the set of all values that Oscar can learn by using the knowledge flows available to him; this includes the knowledge obtainable from both the protocol rules and the cryptographic primitives. The acyclic relation $learns$ (lines 17-18) encodes the partial ordering on Oscar’s maximal knowledge, enforced by the flows from which the knowledge was acquired. For example, the protocol rule 2 specifies that Oscar learns $E_{PK(A)}(N_A, N_B)$ from $E_{PK(B)}(A, N_A)$. Hence, `Oscar.knows` contains both ciphers and `Oscar.learns` includes the mapping

$$\langle E_{PK(B)}(A, N_A), E_{PK(A)}(N_A, N_B)\rangle.$$

The predicate `InitialKnowledge` states that Oscar may not draw any composite values. Rather, he must learn them from the protocol rules or the primitives. The predicate `FinalKnowledge` specifies that Oscar's maximal knowledge contains a value $v$ if and only if Oscar draws $v$ or he learns it from a knowledge flow originating in his initial knowledge.

### 3.2 Modelling Cryptographic Primitives

The Needham-Schroeder protocol requires the use of cryptographic primitives to encrypt/decrypt messages and generate nonces. Our encoding of the knowledge flows and values associated with these primitives is shown in Model Excerpt 2. Note that we do not explicitly model primitives as principals. Instead, we define the pools of values drawn by the primitives as signatures and encode their input/output behavior as predicates. For example, the initial knowledge of $Encryptor$ is given by the set $Ciphertext$, and $Encryptor$'s operation is encoded in the predicates `Encryptor` and `Decryptor`.

A $Ciphertext$ represents an encryption of a non-empty $plaintext$ (line 31) with a given $key$ (line 32). The predicate `Encryptor` formalizes the encryption knowledge flow from Oscar's perspective. It states that, in order to learn the cipher $v$ from the Encryptor, Oscar must provide the input $x$ consisting of the plaintext and the key associated with $v$. Similarly, the predicate `Decryptor` stipulates that Oscar can learn the plaintext $v$ after he presents the input $x$ consisting of an encryption of $v$ and the corresponding decryption key.

Note that this model of ciphers accommodates both public and symmetric key encryption. Symmetric key encryption is the default; invoking the predicate `PublicKeyCryptography` switches on public key encryption. Any atomic value owned by a principal can serve as his public/private key pair. The public portion of any principal's key is accessible to Oscar through the $draws$ relation. The decryption constraint on line 42 ensures that Oscar can decrypt a message only if he $owns$ the value representing the public/private key pair.

Nonces are encoded as composites with two fields, $seed$ and $id$. The field $id$ stores the identity of the principal to whom the nonce was issued. The predicate `NonceGenerator` says that, from Oscar's point of view, the generator will issue a nonce labeled with Oscar's identifier when presented with the input seed $x$.

### 3.3 Modelling Protocol Rules

The models presented so far are a part of a generic Alloy framework developed for analyzing knowledge flows. This section describes the values and rules specific to the Needham-Schroeder protocol.

Principals' identifiers are modelled as atomic values contained in the set $Identity$ (Model Excerpt 3, line 64). Each principal $owns$ an $Identity$ (67), which also doubles as its owners' public/private key pairs (68).

**Model Excerpt 2** Cryptographic Values and Primitives

```
module kf/primitives/encryption
open kf/basicdeclarations

sig Ciphertext extends CompositeValue {
 plaintext: some Value,
 key: Value }

pred PublicKeyCryptography() {
 Ciphertext.key in Principal.owns & AtomicValue }

pred Encryptor(x: set Value, v : Value) {
 v in Ciphertext && x = v.key + v.plaintext }

pred Decryptor(x: set Value, v : Value) {
 some c : plaintext.v | x = (c.key + c) &&
  (PublicKeyCryptography() =>
  c.key in Oscar.owns) }

pred PerfectCryptography() {
 (all disj c1,c2: Ciphertext | c1.plaintext !=
  c2.plaintext || c1.key != c2.key)
 (all c : Ciphertext | c != c.key &&
  c != c.plaintext) }
⋮
module kf/primitives/nonces
open kf/basicdeclarations

sig Nonce extends CompositeValue {
 seed : Value,
 id : Value }

pred NonceGenerator(x: set Value, v : Value) {
 v in Nonce && v.id in Oscar.owns && x = v.seed }
```

The `ProtocolRules` predicate (line 74) embeds the knowledge flow rules given by equations 1-3 into first-order logic. The predicate `ApplyRules` states that the $learns$ relation may map the set of values $x$ to the value $v$ if and only if the protocol or primitive rules define a knowledge flow from $x$ to $v$.

### 3.4 Checking Security

The predicate `SecurityAssumptions` in Model Excerpt 4 models our assumptions about the properties of cryptographic primitives and principals. We assume perfect public key cryptography (line 94) and the use

**Model Excerpt 3** Needham-Schroeder Protocol

```
59 module kf/needham_schroeder
60 open kf/basicdeclarations
61 open kf/primitives/encryption
62 open kf/primitives/nonces
63
64 sig Identity extends AtomicValue {}
65
66 pred IdentitiesAreKeys() {
67  all p : Principal | some p.owns & Identity &&
68  Ciphertext.key in Identity }
69
70 pred PrimitiveRules(x : set Value, v : Value) {
71  Encryptor(x,v) || Decryptor(x,v) ||
72  NonceGenerator(x,v) }
73
74 pred ProtocolRules(x : set Value, v : Value) {
75  v in Ciphertext && {
76   (x : some Oscar.draws &&
77    let text = v.plaintext, n = text & Nonce |
78    #text = 2 && one n && n.seed in AtomicValue &&
79    n.id = text & Identity) ||
80   (x : one Ciphertext && (some n : seed.x |
81    #x.plaintext = 2 && v.key in x.plaintext &&
82    n.id = x.key &&
83    v.plaintext = (x.plaintext - v.key) + n)) ||
84   (x : one Ciphertext &&
85    (some n : id.(x.key) & Nonce |
86    #x.plaintext = 2 && n in x.plaintext &&
87    v.plaintext = x.plaintext - n)) }}
88
89 pred ApplyRules() {
90  all v : Value | let x = Oscar.learns.v |
91   some x <=> PrimitiveRules(x, v) ||
92    ProtocolRules(x, v) }
```

of identifiers as public/private key pairs (line 95).

The security property that the protocol should satisfy is given by the predicate `SecurityTheorem`. It states that Oscar's maximal knowledge never contains two nonces, $nA$ and $nB$, such that $nB$ is generated by Bob in response to a protocol initialization message sent by Alice (a cipher containing Alice's identity and one of her nonces). The assertion `Security` stitches the model together to stipulate that the security property should hold if Oscar obtains his maximal knowledge by applying the knowledge flow rules to the values he draws.

**Model Excerpt 4** Security Assumptions and Theorem

```
pred SecurityAssumptions() {
 PerfectCryptography() && PublicKeyCryptography()
 IdentitiesAreKeys() }

pred SecurityTheorem() {
 no disj Alice, Bob : HonestUser,
 nA, nB : Oscar.knows & Nonce |
  nA.id in Alice.owns && nB.id in Bob.owns &&
  (some c : Ciphertext | nB.seed = c &&
   c.key = nB.id &&
   c.plaintext = nA.id + nA) }

assert Security {
 InitialKnowledge() && FinalKnowledge() &&
 SecurityAssumptions() && ApplyRules() =>
 SecurityTheorem() }
```

The Alloy Analyzer generates a counterexample to the `Security` assertion (Figure 2) that is a knowledge flow representation of the parallel session attack discovered by Lowe [28]. Alice uses $cipher0$ to initiate the protocol with Oscar, who extracts $nA$ and forwards it to Bob in $cipher1$. Thinking that he is authenticating with Alice, Bob responds with $cipher2$ which Oscar simply forwards to Alice. She completes the session with Oscar by sending him $nB$, which she believes is his nonce, in $cipher3$. Oscar now knows both $nA$ and $nB$, contrary to our claim.

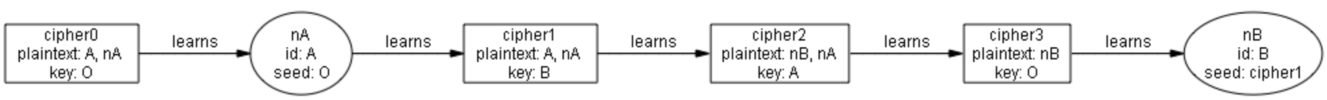


Figure 2: Parallel Session Attack on the Needham-Schroeder Protocol

# 4 Knowledge Flow Analysis

Knowledge flow analysis is based on a simple mathematical foundation. This section formalizes the ideas outlined in the discussion of knowledge flow basics. We describe how *communication rules* direct knowledge flows (4.1), show that our treatment of primitives ensures a fixed pool of values (4.2), formulate the analysis problem in terms of Oscar's knowledge flows (4.3), and present a small-model theorem which makes our analysis complete for a bounded number of parallel protocol sessions (4.4).

## 4.1 Communicating Knowledge

We denote the sets of all *principals* and *values* by $P$ and $V$. A subset of $P \times V$ is a *state of knowledge* drawn from $K = 2^{P \times V}$, the set of all possible states of knowledge. For a given state of knowledge $k \in K$, we say that "$p$ knows $v$" if $(p, v) \in k$.

**Definition 1** *A tuple* $(R, k_0)$ *is a* knowledge flow *for* $(P, V)$ *directed by the* communication rules $R \subseteq P \times V \times P \times K$ *and originating from the state* $k_0 \in K$.

A communication rule describes the conditions under which one principal may gain knowledge from another. For example, the rule $(e, E_{PK(p_b)}(v), p_a, \{(p_a, PK(p_b)), (p_a, v)\})$ states that the encryptor $e$ will tell the cipher $E_{PK(b)}(v))$ to the principal $p_a$ if $p_a$ knows $p_b$'s public key and the plaintext $v$.

Note that our definition of a communication rule limits the class of protocols expressible in the knowledge flow framework. In particular, our rules cannot be used to specify conditions under which information is *withheld* from a principal, such as "$a$ will *not* tell $v$ to $b$ if $b$ knows $x$". Although many practical protocols do not require this form of expressiveness, withholding of knowledge is an essential concept in systems that use certificates: revoking a certificate requires withholding of information. We are working on reformulating the certificate revocation problem using valid and invalid certificate sets, which should allow us to circumvent this limitation.

Given a set of communication rules $R$, we say that $k' \in K$ is reachable from $k \in K$ via $R$ if $k'$ is the result of applying all rules in $R$ to $k$ at most once; i.e. $k' = f_R(k)$ where

**Definition 2** $f_R : K \longrightarrow K$ *such that*

$$f_R(k) = k \cup \left\{ (p_a, v) : \begin{array}{l} (p_b, v) \in k, k_a \subseteq k, \textit{ and} \\ (p_b, v, p_a, k_a) \in R, \\ \textit{for some } p_b \in P \textit{ and } k_a \in K \end{array} \right\}.$$

A state of knowledge $k_n$ is reachable in the context of a knowledge flow $(R, k_0)$ if $k_n = f_R^n(k_0)$. The *maximal state of knowledge* $f_R^*(k_0)$ is the limit of $k_n = f_R^n(k_0)$ as $n \to \infty$. A state of knowledge $f_{R_\kappa}^*(\kappa)$ is *valid* for a knowledge flow $(R, k_0)$ if $R_\kappa \subseteq R$ and $\kappa \subseteq k_0$. Since $f_R(k_0)$ is monotonically increasing[2] in $R$ and $k_0$, any valid state of knowledge is a subset of the maximal state of knowledge. Hence, the maximal state of knowledge is also the smallest fixed point of $f_R$ which subsumes $k_0$.

## 4.2 Initial Knowledge

For each value $v$, $Source(v) = \{p : (p, v) \in k_0\}$ defines the set of principals who draw $v$. In the knowledge flow framwork, a principal $p$ outside of $Source(v)$ can learn $v$ only by communicating with principals who know $v$. We therefore treat cryptographic primitives, and other computationally feasible algorithms, as principals. For example, suppose that, in practice, $p$ can compute $v$ by applying the algorithm $\mathcal{A}$ to

[2]It is evident from Definition 2 that self-rules such as $r = (p, v, p, k_p) \in R$ do not affect the flow of knowledge: $f_R(k) = f_{R-\{r\}}(k)$. We therefore assume that $R$ does not contain any self-rules.

inputs $i_1, i_2, \ldots, i_n$. We model $\mathcal{A}$ by adding the principal $A$ to $P$, the tuple $(A, v)$ to $k_0$, and the rule $(A, v, p, \{(p, i_1), (p, i_2), \ldots (p, i_n)\})$ to $R$.

Our treatment of primitives ensures that $Knowledge(k_0) = \{v : (p, v) \in k_0 \text{ for some } p \in P\}$ consists of *all* learnable values. Hence, $V$ is the same in the initial and the maximal state of knowledge,

$$Knowledge(k_0) = Knowledge(f_R^*(k_0)), \tag{4}$$

which implies that we can safely restrict our analysis to the subset of $R$ applicable to $k_0$. Formally,

$$(4) \implies f_R(k_0) = f_{R(k_0)}(k_0) \text{ and } f_R^*(k_0) = f_{R(k_0)}^*(k_0),$$

where $R(k_0) = \left\{ (p_b, x, p_a, k_a) \in R : \begin{array}{l} \{x\} \cup \{v : (p_a, v) \in k_a\} \\ \subseteq Knowledge(k_0) \end{array} \right\}$.

### 4.3 Adversaries' Knowledge

Let $O \subseteq P$ be a group of collaborating adversaries. We collapse $O$ into a single principal $o$ using the following merging function:

$$\begin{array}{rcl} Merge(p) & = & \begin{cases} o \text{ if } p \in O, \\ p \text{ if } p \notin O \end{cases} \\ Merge(k) & = & \{(Merge(p), v) : (p, v) \in k\} \\ Merge(r) & = & (Merge(p_b), v, Merge(p_a), Merge(k_a)) \\ & & \text{where } r = (p_b, v, p_a, k_a) \in R \end{array}$$

The merging of adversaries does not rule out any attacks because $Merge(f_R^*(k_0)) \subseteq f_{Merge(R)}^*(Merge(k_0))$. We subsequently assume that $Merge$ is implied and use $P$, $R$, and $k_0$ to refer to $Merge(P)$, $Merge(R)$, and $Merge(k_0)$.

Security properties of protocols are expressed as predicates on the values known to Oscar in the maximal state of knowledge. We therefore restrict our analysis of knowledge flows to finding all the values in the projection of $f_{R(k_0)}^*(k_0)$ on Oscar. Specifically, we introduce the projection function $g_{R,k_0}$ and show that its smallest fixed point is the image of Oscar under $f_{R(k_0)}^*(k_0)$.

**Definition 3** *Let $X \rightarrow x$ denote the existence of a rule $(p, x, o, k_\sigma) \in R(k_0)$ for some $p \in P - \{o\}$ and $k_\sigma \in K$ with $X = \{v : (o, v) \in k_\sigma\}$. We define $g_{R,k} : 2^V \longrightarrow 2^V$ as*

$$g_{R,k_0}(X) = X \cup \{x : X_\sigma \rightarrow x \text{ for some } X_\sigma \subseteq X\}.$$

*The set of values reachable from $X$ is given by $g_{R,k_0}^*(X)$, which is the limit of $g_{R,k_0}^n(X)$ as $n \rightarrow \infty$.*

Since $f_R(k_0)$ is monotonically increasing in $R$ and $k_0$, Oscar's pool of values under $f_R^*(k_0)$ is maximized if (a) Oscar tells everything he knows to the honest principals and (b) the honest principals tell everything they know to each other. Therefore, $(P - \{o\}) \times Knowledge(k_0)$ should be included in the maximal state of knowledge. This is equivalent to assuming that each honest principal draws $Knowledge(k_0)$ because $k \subseteq f_R^*(k_0)$ implies that $f_R^*(k_0) = f_R^*(k_0 \cup k)$.

**Lemma 4** *Let $[(P - \{o\}) \times V_0] \subseteq k_0$ with $V_0 = Knowledge(k_0)$ and let $k_n = f_R^n(k_0)$. Then there exists a unique set $X_n \subseteq V$ such that*

$$k_n = [(P - \{o\}) \times V_0] \cup [\{o\} \times X_n]. \tag{5}$$

*The set $X_n$ has the property that $X_n = g_{R,k_0}^n(X_0)$.*

*Proof.*

We use induction on $n$. For $n = 0$, $X_n = X_0 = g_{R,k_0}^n(X_0)$. Since $(P - \{o\}) \times V_0 \subseteq k_0$ and $V_0 = Knowledge(k_0)$, there exists a unique $X_0$ such that $k_0$ satisfies (5).

Let $X_n = g_R^n(X_0)$ be a unique solution to (5) and $Knowledge(k_n) = V_0$ (our induction hypothesis). We know that $k_{n+1} = f_R(k_n)$ and, therefore, $Knowledge(k_{n+1}) = Knowledge(k_n) = V_0$. Together with $[(P - \{o\}) \times V_0] \subseteq k_n \subseteq k_{n+1}$, this implies the existence a unique $X_{n+1}$ for which $k_{n+1}$ satisfies (5). We now need to prove that $X_{n+1} = g_{R,k_0}^{n+1}(X_0)$.

Definition (5) lets us infer that $x \in X_{n+1} \iff (o, x) \in k_{n+1} = f_R(k_n)$. According to Definition (2), $(o, x) \in f_R(k_n)$ if and only if i) $(o, x) \in k_n$, which is, by (5), equivalent to $x \in X_n$, or ii) there exists a $p \in P$ and $k_\sigma \in K$ such that $(p, x) \in k_n$, $k_\sigma \subseteq k_n$, and $(p, x, o, k_\sigma) \in R$. Since there are no self-rules $(o, v, o, k_\sigma) \in R$, we know that $p \in P - \{o\}$. This, together with $x \in X_{n+1} \subseteq V_0$, implies that $(p, x) \in [(P - \{o\}) \times V_0] \subseteq k_n$. Given $[(P - \{o\}) \times V_0] \subseteq k_n$ and $V_0 = Knowledge(k_n)$, the condition $k_\sigma \subseteq k_n$ is equivalent to

$$X_\sigma = \{v : (o, v) \in k_\sigma\} \subseteq \{v : (o, v) \in k_n\} = X_n \text{ and}$$
$$\{v : (o, v) \in k_\sigma\} \subseteq V_0.$$

Since $x \in X_{n+1} \subseteq V_0$, $\{v : (o, v) \in k_\sigma\} \subseteq V_0$ gives us $(p, x, o, k_\sigma) \in R(k_0)$. Therefore, case ii) holds if and only if there exists a set $X_\sigma \subseteq X_n$ such that $X_\sigma \to x$. By Definition (3), case i) or case ii) holds if and only if $x \in g_{R,k_0}(X_n)$. Hence, $X_{n+1} = g_{R,k_0}(X_n)$ and the lemma follows by induction on $n$.

□

### 4.4 Detecting Intruders

Let $m$ be the total number of values used in a single protocol session, including the subterms of each composite value. Suppose that Oscar can use only the primitives which *compose* or *decompose* inputs and for which the composition rules have no collisions (e.g encryptor/decryptor). Then, the theory in [41] implies the following: if there exists an attack in which Oscar uses $w$ parallel protocol sessions, then such an attack need not involve more than $w \cdot m$ values. From (4) we infer that this corresponds to a valid state of knowledge $f_R^*(k_\sigma)$ derived from the set $k_\sigma \subseteq k_0$ of cardinality $|Knowledge(k_\sigma)| \leq wm$. By Lemma 4, we can conclusively *decide* whether there is an attack which uses $w$ parallel protocol sessions by computing

$$\left\{ v \in g^*_{R,k_\sigma}(X_\sigma) : \begin{array}{l} \text{for } [\{o\} \times X_\sigma] \subseteq k_\sigma \subseteq k_0 \\ \text{with } |Knowledge(k_\sigma)| \leq wm \end{array} \right\}. \tag{6}$$

# 5 Evaluation

We have applied the theory developed in the previous section to check the security of the original [36] and modified [28] Needham-Schroeder Public Key Protocol, the Otway-Rees Mutual Authentication Protocol [40], and the bootstrapping and renewal protocols based on Controlled Physical Random Functions (CPUFs) [20; 21].

The knowledge flows of the protocols were embedded into Alloy using the pattern presented in section 3. The pattern is embodied in a general Alloy framework for knowledge flow analysis which includes definitions of basic concepts (Model Excerpt 1), a library of primitives, and a model outline for specifying protocol rules and security theorems. For example, Model Excerpt 2 shows portions of Alloy modules that encode generic encryption/decryption and nonce generator primitives, and Model Excerpts 3 and 4 comprise an instantiation of the modelling outline for the Needham-Schroeder protocol.

We have found that the Alloy framework and its associated tool support make the process of knowledge flow modelling fast, simple, and accurate. Our analysis is sound and, since most cryptographic primitives used in practice are composing/decomposing, we can make it complete for a bounded number of parallel sessions by applying the results from section 4.4. In the case of the modified Needham-Schroeder protocol, for example, we have proved that it is secure against all attacks that use two parallel sessions. The analysis of the Otway-Rees protocol (Appendix A) produced the type flaw attack described in [8]. We found the CUPFs protocols (Appendix B) to be secure for a single protocol session and, therefore, for an unlimited number of sessions.

The main limitation of our approach is that it is not fully general. As pointed out in section 4.1, protocols that *withhold* information under certain conditions cannot be formulated as knowledge flows. However, this limitation does not significantly detract from practical usefulness of knowledge flow analysis: as far as we know, few practical protocols contain information-withholding rules.

# 6 Related Work

The first formalisms designed for reasoning about cryptographic protocols are belief logics such as BAN logic [8], used by the Convince tool [27] with the HOL theorem prover [24], and its generalizations (GNY [23], AT [3], and SVO logic [44] which the C3PO tool [15] employs with the Isabelle theorem prover [39]). Belief logics are difficult to use since the logical form of a protocol does not correspond to the protocol itself in an obvious way. Almost indistinguishable formulations of the same problem lead to different results. It is also hard to know if a formulation is over constrained or if any important assumptions are missing. BAN logic and its derivatives cannot deal with security flaws resulting from interleaving of protocol steps [7] and cannot express any properties of protocols other than authentication [30]. To overcome these limitations, the knowledge flow formalism has, like other approaches [12; 29; 32; 35; 43], a concrete operational model of protocol execution. Our model also includes a description of how the honest participants in the protocol behave and a description of how an adversary can interfere with the execution of the protocol.

Specialized model checkers such as Casper [29], Mur$\phi$ [35], Brutus [12], TAPS [13], and ProVerif [1] have been successfully used to analyze security protocols. Like knowledge flow analysis in Alloy, these tools are based on state space exploration which leads to an exponential complexity. Athena [43] is based on a modification of the strand space model [18]. Even though it reduces the state space explosion problem, it remains exponential. Multiset rewriting [17] in combination with tree automata is used in Timbuk [19]. The relation between multiset rewriting and strand spaces is analyzed in [9]. The relation between multiset rewriting and process algebras [2; 34] is analyzed in [5].

Proof building tools such as NRL, based on Prolog [32], have also been helpful for analyzing security protocols. However, they are not fully automatic and often require extensive user intervention. Model checkers lead to completely automated tools which generate counterexamples if a protocol is flawed. For theorem-proving-based approaches, counterexamples are hard to produce.

For completeness, we note that if the initial knowledge of the intruder consists of a finite number of explicit (non-parameterized, non-symbolic) values, then a polynomial time intruder detection algorithm can be shown to exist using a generalization of the proof normalization arguments [4; 22; 31], which were employed in [6; 37] and have been implemented in the framework [38]. However, in practice, the initial knowledge of an intruder is unbounded and represented by a finite number of parameterized sets, each having an infinite number of elements.

The key advantage of the knowledge flow approach over other formalisms is its simplicity and flexibility. It is simple in the sense that the underlying mathematics is straightforward and elementary; it does not require any specialized background (in logic). It is flexible in the sense that the same library of cryptographic primitives can be used to model different protocols and that the security of a complex scheme involving multiple protocols can be verified. Knowledge Flow Analysys allows modeling of confidentiality and authenticity via a wide range of primitives such as pairing, union, hashing, symmetric key encryption, public key encryption, MACs and digital signatures.

Our formalism derives its simplicity from being just sufficiently expressive to enable modelling of practical cryptographic protocols. In particular, existentials [17] cannot be encoded as knowledge flows; existentials are implicitly modeled in Oscar's initial knowledge. As mentioned in Section (4.1), NP-hardness proofs which use (existential) Horn clause reduction [17] or SAT3 reduction [41] are not applicable to Knowledge Flow Analysis.

# 7 Conclusion

This paper introduces a new method for formalizing and checking security protocols. Our approach enables natural encoding of protocol rules, simple treatment of primitives, direct embedding into first order logic, and sound analysis that is also complete for many practical protocols.

We have developed a general framework for analyzing knowledge flows using the Alloy Analyzer. The framework has been used to generate easily understandable knowledge flow representations of parallel session and type flaw attacks on the Needham-Schroeder and Otway-Rees protocols. We have also instantiated it with the rules for CPUFs key management protocols and verified the protocols' correctness for an unlimited number of parallel sessions.

We believe that knowledge flow analysis may be polynomial-time decidable for some protocols. Future work will involve identifying the class of protocols whose knowledge flows are analyzable in polynomial time and developing a specialized tool for checking them.

## Acknowledgments

We would like to thank Viktor Kuncak, Ishan Sachdev, and Ilya Shlyakhter for their contributions to and comments on earlier versions of this work.

## References

[1] M. Ababi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM*, 52(1):102–146, 2005.

[2] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In *Proc. of CONCUR '97: Concurrency Theory, 8th International Conference, LNCS 1243*, pages 59–73, 1997.

[3] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, 1991.

[4] D. Basin and H. Ganzinger. Automated complexity analysis based on ordered resolution. *JACM*, 48(1):70–109, 2001.

[5] S. Bistarelli, I. Cervesato, G. Lenzini, and F. Martinelli. Relating process algebras and multiset rewriting for immediate decryption protocols. In *2nd Int. Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS), LNCS 2776*, pages 86–99, 2003.

[6] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Flow logic for dolev-yao secrecy in cryptographic processes. *Future Gener. Comput. Syst.*, 18(6):747–756, 2002.

[7] C. Boyd and W. Mao. On a limitation of ban logic. In *Advances in Cryptology: Eurocrypt '93, Springer-Verlag*, pages 240–247, 1993.

[8] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[9] I. Cervesato, N. Durgin, J.Mitchell, P. Lincoln, and A. Scedrov. A comparison between strand spaces and multiset rewriting for security protocol analysis. In *Software Security Theories and Systems, Mext-NSF-JSPS International Symposium, ISSS 2002, LLNCS 426*, 2003.

[10] Y. Chevalier, R. Kuesters, M. Rusinowitch, and M. Turuani. An np decision procedure for protocol insecurity with xor. In *LICS'03*, 2003.

[11] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the dolev-yao intruder for analyzing an unbounded number of sessions. Technical Report RR-4869, Rapport de recherche de l'INRIA-Lorraine, Equipe : CASSIS, July 2003.

[12] E. Clarke, S. Jha, and W. Marrero. Verifying security protocols with brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, 2000.

[13] E. Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Computer Security Foundations Workshop*, 2004.

[14] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 271–280, 2003.

[15] A. H. Dekker. C3po: A tool for automatic sound cryptographic protocol analysis. In *13th IEEE Computer Security Foundations Workshop (CSFW'00)*, 2000.

[16] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[17] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 1:677–722, 2004.

[18] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, 1998.

[19] G. Feuillade, T. Genet, and V. V. T. Tong. Reachability analysis of term rewriting systems. *Technical Report RR-4970, INRIA, 2003, to be published in the Journal of Automated Reasoning*, 2004.

[20] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Controlled physical random functions. In *Proceedings of the 18th Annual Computer Security Applications Conference*, 2002.

[21] B. L. P. Gassend. Physical random functions. Master's thesis, MIT, 2003.

[22] R. Givan and D. Mcallester. Polynomial-time computation via local inference relations. *ACM Trans. Comput. Logic*, 3(4):521–541, 2002.

[23] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 234–248, 1990.

[24] M. J. C. Gordon and T. F. Melham. *Introduction to HOL, a theorem proving environment for higher-order logic*. Cambridge University Press, Cambridge, England, 1993.

[25] D. Jackson. Automating first-order relational logic. In *Proc. ACM SIGSOFT Conf. Foundations of Software Engineering / European Software Engineering Conference (FSE/ESEC '00)*, 2000.

[26] D. Jackson. Alloy: a lightweight object modelling notation. *ACM TOSEM*, 11(2):256–290, 2002.

[27] R. W. Lichota, G. L. Hammonds, and S. H. Brackin. Verifying cryptographic protocols for electronic commerce. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 53–65, 1996.

[28] G. Lowe. Breaking and fixing the needham-schröder public-key protocol using csp and fdr. In *2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, 1996.

[29] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 18–30, 1997.

[30] W. Mao and C. Boyd. Towards formal analysis of security protocols. In *Proceedings of the Computer Security Foundation Workshop VI*, pages 147–158, 1993.

[31] D. McAllester. Automatic recognition of tractability in inference relations. *Journal of ACM*, 40(2), 1993.

[32] C. A. Meadows. The nrl protocol analyzer: An overview. In *Proceedings of the 2nd Conference on the Practical Applications of Prolog*, 1994.

[33] J. Millen and V. Shmatikov. Symbolic protocol analysis with an abelian group operator or diffie-hellman exponentiation. *Journal of Computer Security*, 2004.

[34] R. Milner. *Communicating and Mobile Systems: the $\pi$-Calculus*. Cambridge University Press, 2000.

[35] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using mur$\phi$. In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153, 1997.

[36] R. Needham and M. Schröder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[37] F. Nielson, H. R. Nielson, and H. Seidl. Cryptographic analysis in cubic time. In *TOSCA'01*, volume 62 of *ENTCS*, 2001.

[38] F. Nielson, H. R. Nielson, H. Sun, M. Buchholtz, R. R. Hansen, H. Pilegaard, and H. Seidl. The succinct solver suite. In *10th TACAS*, volume 2988 of *LNCS*, 2004.

[39] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL Tutorial Draft*, March 8 2002.

[40] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21:8–10, January 1987.

[41] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In *Proceedings of the 14th Computer Security Foundations Workshop*, pages 174–187, 2001.

[42] V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *ESOP'04*, volume 2986 of *LNCS*, 2004.

[43] D. Song, S. Berezin, and A. Perrig. Athena, a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1), 2001.

[44] P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 13th Symposium on Security and Privacy*, 1994.

# Appendix A The Otway-Rees Protocol

```
module kf/otwayreese
open kf/basicdeclarations
open kf/primitives/encryption
open kf/primitives/nonces

sig Message extends CompositeValue {
 contents: some Value }

sig Identity extends AtomicValue {}

pred PrimitiveRules(x : set Value, v : Value) {
 Encryptor(x,v) || Decryptor(x,v) || NonceGenerator(x,v) ||
 (x : Message && v in x.contents) }

pred idCipher(cipher: Value) {
 cipher : Ciphertext &&
 some cipher.key.id & cipher.plaintext &&
 cipher.plaintext in Identity &&
 one cipher.plaintext - cipher.key.id }

pred keyCipher(cipher: Value) {
 cipher : Ciphertext &&
 some cipher.key.id }

pred message1(m: Value) {
```

```
 m : Message &&
 let cipher = m.contents & Ciphertext | {
  idCipher(cipher) &&
  m.contents = cipher + cipher.plaintext }}

pred message2(m: Value) {
 m : Message &&
 some cipher1 : Ciphertext |
  let cipher2 = m.contents & Ciphertext - cipher1 | {
   idCipher(cipher1) &&
   idCipher(cipher2) &&
   cipher1.plaintext = cipher2.plaintext &&
   m.contents = cipher1 + cipher2 + cipher1.plaintext }}

pred message3(m: Value) {
 m : Message &&
 some cipher1 : Ciphertext |
  let cipher2 = m.contents & Ciphertext - cipher1 | {
   keyCipher(cipher1) &&
   keyCipher(cipher2) &&
   cipher1.plaintext = cipher2.plaintext &&
   m.contents = cipher1 + cipher2 }}

pred message4(m: Value) {
 m : Message &&
 keyCipher(m.contents) }

pred ProtocolRules(x : set Value, v : Value) {
 (x : some Oscar.draws && message1(v)) ||
 (message1(x) && message2(v) && x.contents in v.contents) ||
 (message2(x) && message3(v) && x.contents.key = v.contents.key) ||
 (message3(x) && message4(v) && v.contents in x.contents) }

pred ApplyRules() {
 all v : Value | let x = Oscar.learns.v |
  some x <=> PrimitiveRules(x, v) || ProtocolRules(x, v) }

pred SecurityAssumptions(){
 PerfectCryptography() &&
```

```
 all p : Principal | some p.owns & Identity }

pred SecurityTheorem() {
 no oldResp, newResp : PUFResponse,
 renew : param.(oldResp.isRespTo) & HonestUser,
 cipher : Ciphertext |
  let oldChall = oldResp.isRespTo, newChall = newResp.isRespTo |
   oldChall.isHashOf : some (AtomicValue - Oscar.draws) &&
   cipher.key.isHashOf = oldResp + renew.hash &&
   cipher.plaintext = newResp &&
   newChall.isHashOf = oldChall + renew.hash &&
   newResp in Oscar.knows }

assert Security {
 InitialKnowledge() && FinalKnowledge() &&
 SecurityAssumptions() && ApplyRules() => SecurityTheorem() }

pred SecurityTheorem() {
 no m1, m2, m3, m4: Oscar.knows & Message,
 A, B: HonestUser.owns & Identity | {
  message1(m1) && message2(m2) &&
  message3(m3) && message4(m4) &&
  m1.contents.key.id = A &&
  m2.contents.key.id = A + B &&
  m3.contents.key.id = A + B &&
  m4.contents.key.id = A &&
  m4.contents.plaintext in
   (HonestUser.draws - Oscar.draws) & AtomicValue &&
  m4.contents.plaintext in Oscar.knows }}

assert Security {
 InitialKnowledge() && FinalKnowledge() &&
 SecurityAssumptions() && ApplyRules() => SecurityTheorem() }
```

## Appendix B The CUPF Renewal Protocol

```
module kf/primitives/hashing
open kf/basicdeclarations

```

```
sig Hash extends CompositeValue {
 isHashOf: some Value }

pred CollisionFreeHashing() {
 all disj h1, h2: Hash | h1.isHashOf != h2.isHashOf }

pred Hasher(x : set Value, v : Value) {
 v in Hash && x = v.isHashOf }
```

```
module kf/cpufs
open kf/basicdeclarations
open kf/primitives/encryption
open kf/primitives/hashing

sig PUFResponse extends CompositeValue {
 isRespTo: Value }

pred UniquePUFResponses() {
 all r: PUFResponse | r.isRespTo !in (PUFResponse - r).isRespTo }

sig RenewProg in Principal {
 param : Value,
 hash : AtomicValue & owns
}{ param + hash in draws + knows }

pred SecretsNotLeaked() {
 no (RenewProg & HonestUser).param.isHashOf & PUFResponse &&
 (RenewProg & HonestUser).param in Hash }

pred GetResponsePrimitive(x : set Value, v : Value) {
 v in PUFResponse &&
 v.isRespTo.isHashOf = Oscar.hash + Oscar.param &&
 x = v.isRespTo }

pred GetSecretPrimitive(x : set Value, v : Value) {
 v in Hash &&
 v.isHashOf = isRespTo.(Oscar.param) + Oscar.hash &&
 x = v.isHashOf }

```

```
pred PrimitiveRules(x : set Value, v : Value) {
 Encryptor(x,v) || Decryptor(x,v) ||
 GetResponsePrimitive(x,v) || GetSecretPrimitive(x,v) }

pred ProtocolRules(x : set Value, v : Value) {
 x : some Oscar.draws && {
  (v in (RenewProg & HonestUser).(param + hash)) ||
  (v in Ciphertext &&
   some renew: RenewProg & HonestUser |
    let renewHash = renew.hash |
     v.key.isHashOf = isRespTo.(renew.param) + renewHash &&
     v.plaintext.isRespTo.isHashOf = renewHash + renew.param) }}

pred ApplyRules() {
 all v : Value | let x = Oscar.learns.v |
  some x <=> PrimitiveRules(x, v) || ProtocolRules(x, v) }

pred SecurityAssumptions(){
 UniquePUFResponses() && PerfectCryptography() &&
 SingleValueEncryption() && CollisionFreeHashing() &&
 SecretsNotLeaked() }

pred SecurityTheorem() {
 no disj oldResp, newResp : PUFResponse,
 renew : param.(oldResp.isRespTo) & HonestUser,
 cipher : Ciphertext |
  let oldChall = oldResp.isRespTo, newChall = newResp.isRespTo |
   oldChall.isHashOf : some (AtomicValue - Oscar.draws) &&
   cipher.key.isHashOf = oldResp + renew.hash &&
   cipher.plaintext = newResp &&
   newChall.isHashOf = oldChall + renew.hash &&
   newResp in Oscar.knows }

assert Security {
 InitialKnowledge() && FinalKnowledge() &&
 SecurityAssumptions() && ApplyRules() => SecurityTheorem() }
```