Encryption as an Abstract Datatype:

an extended abstract

Dale Miller
INRIA/Futurs/Saclay and École polytechnique

Outline

1. Security protocols specified using multisets rewriting.
2. Eigenvariables for nonces and session keys.
3. Encrypted data as an abstract datatype.
4. Protocols as linear logic theories.
5. Tests, traces, and interpolants.

A Typical Protocol Specification

The following is a presentation of the Needham-Schroeder Shared Key Protocol. Alice and Bob make use of a trusted server to help them establish their own private channel for communications.

Message 1	$A \longrightarrow S$: A, B, n_A
Message 2	$S \longrightarrow A$: $\{n_A, B, k_{AB}, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$
Message 3	$A \longrightarrow B$: $\{k_{AB}, A\}_{k_{BS}}$
Message 4	$B \longrightarrow A$: $\{n_B\}_{k_{AB}}$
Message 5	$A \longrightarrow B$: $\{n_B - 1\}_{k_{AB}}$

Here, A, B, and S are agents (Alice, Bob, server), and the k's are encryption keys, and the n's are nonces.

One of our goals is to replace this specific syntax with one that is based on a direct use of logic. We will then investigate if logic's meta-theory can help in reasoning about security.

Motivating a more declarative specification

The notation $A \longrightarrow B\colon M$ seems to indicate a "three-way synchronization," but communication here is asynchronous: Alice put a message on in a network and Bob picks it up from the network. An intruder might read/delete/modify the message.

A better syntax might be:

$$\begin{array}{rcl} A & \longrightarrow & A' \mid \mathsf{N}(M) \\ B \mid \mathsf{N}(M) & \longrightarrow & B' \\ \vdots & & \\ E \mid \mathsf{N}(M) & \longrightarrow & E' \mid \mathsf{N}(M) \end{array}$$

More generally,

$$(A\ \mathit{Memory}) \mid \mathsf{N}(M_1) \mid \cdots \mid \mathsf{N}(M_p) \longrightarrow (A'\ \mathit{Memory}') \mid \mathsf{N}(P_1) \mid \cdots \mid \mathsf{N}(P_q)$$

where $p, q \geq 0$. The agent can be missing from the left (agent creation) or can be missing from the right (agent deletion).

This is essentially a specification of *multiset rewriting* of atomics formulas.

Dynamic creation of new symbols

New symbols representing nonces (used to help guarantee "freshness") and new keys for encryption and session management are needed also in protocols. We could introduced syntax such as:

$$a_1\ S \longrightarrow new\ k.\ a_2\ \langle k, S\rangle \mid \mathsf{N}(\{M\}_k)$$

This *new* operator looks a bit like a quantifier: it should support α-conversion and seems to be a bit like reasoning generically. The scope of *new* is over the body of this rule.

Static distribution of keys

Consider a protocol containing the following messages.

$$\vdots$$

Message i $\quad A \longrightarrow S\colon \{M\}_k$

Message j $\quad S \longrightarrow A\colon \{P\}_k$

$$\vdots$$

How can we declare that a key, such as k, is only built into two specific agents. This static declaration is critical for modularity and for establishing correctness later. A **local** declaration can be used (borrowed from λProlog).

$$\vdots$$

$$\mathit{local}\ k. \quad \left\{ \begin{array}{r} A \longrightarrow A' \mid\ \mathsf{N}(\{M\}_k) \\ S \mid\ \mathsf{N}(\{P\}_k) \longrightarrow S' \end{array} \right\}$$

$$\vdots$$

This declarations also appears to be similar to a quantifier.

Are these specifications logical expressions?

Can we view the symbols we have introduced as logical connectives?

	$\mid$	$\longrightarrow$	*new*	*local*	*empty*
disjunctive (Forum)	$\parr$	$\circ\!\!-$	$\forall$	$\exists$	$\bot$
conjunctive (MSR)	$\otimes$	$-\!\!\circ$	$\exists$	$\forall$	**1**

The disjunctive approach allows protocols to be seen as **abstract logic programs**: that is, it fits into the "logic programming as goal-directed search" paradigm.

Note: Logic is not used here to form judgments *about* protocol. Rather, elements of logic are elements of the protocol.

For MSR, see Cervesato, Durgin, Lincoln, Mitchell, Scedrov. "A Meta-Notation for Protocol Analysis," Proceedings of the 12th IEEE Computer Security Foundations Workshop IEEE Computer Society Press, 1999.

Encrypted data as an abstract data type

Encryption keys are encoded as symbolic functions on data of type $data \rightarrow data$.

Replace $\{M\}_k$ with $(k\ M)$.

By providing scope to such keys, encrypted data forms an abstract datatype.

To insert an encryption key into data, we will use the postfix coercion constructor $(\cdot)^\circ$ of type $(data \rightarrow data) \rightarrow data$.

The use of higher-order types means that we will also use the equations of $\alpha\beta\eta$-conversion (a well studied extension to logic programming with robust implementations).

$$\exists\ k.\ \begin{bmatrix} a_1\ S \multimap \forall n.\ a_2\ \langle k^\circ, S\rangle \parr \mathsf{N}(k\ n) \\ a_2\ \langle k^\circ, S\rangle \parr \mathsf{N}(k\ M) \multimap \ldots \end{bmatrix}$$

A Linear Logic Specification of Needham-Schroeder

$$
\begin{array}{l}
\exists k_{as} \exists k_{bs} \{ \\
\quad a\ S \qquad\qquad\qquad\qquad \multimap\ \forall na. \quad a_1\ \langle na, S\rangle \parr \mathsf{N}(\langle a, b, na\rangle). \\
\quad a_1\ \langle N, S\rangle \parr \mathsf{N}(k_{as}\langle N, b, K, En\rangle) \multimap \qquad a_2\ \langle N, K, S\rangle \parr \mathsf{N}(En). \\
\quad a_2\ \langle Na, Key^{\circ}, S\rangle \parr \mathsf{N}(Key\ Nb) \multimap \qquad a_3\ \langle\rangle \parr \mathsf{N}(Key\ \langle Nb, S\rangle). \\
\quad b\ \langle\rangle \parr \mathsf{N}(k_{bs}\ \langle Key^{\circ}, a\rangle) \multimap\ \forall nb. \quad b_1\ \langle nb, Key^{\circ}\rangle \parr \mathsf{N}(Key\ nb). \\
\quad b_1\ \langle Nb, Key\rangle \parr \mathsf{N}(Key\langle Nb, S\rangle) \multimap \qquad b_2\ S. \\
\quad s\ \langle\rangle \parr \mathsf{N}(\langle a, b, N\rangle) \multimap\ \forall k. \quad s\ \langle\rangle \parr \mathsf{N}(k_{as}\langle N, b, k^{\circ}, k_{bs}\langle k^{\circ}, a\rangle\rangle). \\
\}
\end{array}
$$

Outermost universal quantifiers around individual clauses have not been written but are assumed for variables (tokens starting with a capital letter).

Relating implementation and specification

A property of NS should be that Alice can communicate to Bob a secret with the help of a server. That is, the clause

$$\forall x\ (a\ \langle x\rangle \parr b\ \langle\rangle \parr s\ \langle\rangle \multimap a_3\ \langle\rangle \parr b_2\ \langle x\rangle \parr s\ \langle\rangle)$$

can be seen as part of the specification of this protocol.

If we call the above clause *SPEC* and the formula for Needham-Schroeder *NS*, then it is a simple calculation to prove that $NS \vdash SPEC$ in linear logic.

Of course, a kind of converse is more interesting and harder. At least a trivial thing is proved trivially.

Should not logical entailment be a center piece of logical specifications?

Scheme for reasoning about logic programs

Higher-order quantification in this talk will be featured in two ways.

- During computation (proof search) higher-order quantification will be "easy": e.g., generate a new symbol at higher-order type.
- When reasoning about computations, the dual operation of instantiating new symbols with clever substitutions might be necessary.

One approach to reasoning about logic programs is the following:

$P \vdash G$ proof search (cut-free, automated)

$\underline{P' \vdash P}$ reasoning about programs: involves rich substitutions and lemmas

$P' \vdash G$ after cut-elimination (lemma removal), we have a computation again

Notice that P has appeared both positively and negatively in these examples. What corresponded to "generate a new predicate" dualizes to "find a logical expression to substitution" when the polarity is shifted.

Can't we compile away higher-order quantification?

If we simply execute security protocols, then the expressions $\{M\}_k$ and $(k\ M)$ can be compiled as first-order expressions such as

$$(\mathit{apply}\ k\ M), \quad \text{or more appropriately, as} \quad (\mathit{encrypt}\ k\ M).$$

In order to reason about such a protocol, we need to explain the meaning of this new non-logical constant. This complicates the reasoning process somewhat.

Lesson: Do not leave the paradise of Church too soon.

A simple logical equivalence

Consider the following two clauses:

$$a \circ\!\!- \forall k.\mathsf{N}(k\ m) \quad \text{and} \quad a \circ\!\!- \forall k.\mathsf{N}(k\ m').$$

These two clauses show that Alice can take a step that generates a new encryption key and then outputs either the message m or m' in encrypted form. These two clauses seem "observationally similar".

More surprisingly

$$a \circ\!\!- \forall k.\mathsf{N}(k\ m) \dashv\vdash a \circ\!\!- \forall k.\mathsf{N}(k\ m').$$

That is, they are logically equivalent! In particular, the sequent

$$\forall k.\mathsf{N}(k\ m) \longrightarrow \forall k.\mathsf{N}(k\ m')$$

is proved by using the eigenvariable c on the right and the term $\lambda w.(c\ m')$ on the left.

More logical equivalences

If we allow local ($\exists$) abstractions of predicates, then other more interesting logical equivalences are possible.

For example, 3-way synchronization can be implemented using 2-way synchronization with a hidden intermediary.

$$\exists\, x. \left\{ \begin{array}{l} a \parr b \circ\!\!- x \\ x \parr c \circ\!\!- d \parr e \end{array} \right\} \quad \dashv\vdash \quad a \parr b \parr c \circ\!\!- d \parr e$$

Intermediate states of an agent can be taken out entirely.

$$\exists\, a_2, a_3. \left\{ \begin{array}{l} a_1 \parr \mathsf{N}(m_0) \circ\!\!- a_2 \parr \mathsf{N}(m_1) \\ a_2 \parr \mathsf{N}(m_2) \circ\!\!- a_3 \parr \mathsf{N}(m_3) \\ a_3 \parr \mathsf{N}(m_4) \circ\!\!- a_4 \parr \mathsf{N}(m_5) \end{array} \right\} \quad \dashv\vdash$$

$$a_1 \parr \mathsf{N}(m_0) \circ\!\!- (\mathsf{N}(m_1) \circ\!\!- (\mathsf{N}(m_2) \circ\!\!- (\mathsf{N}(m_3) \circ\!\!- (\mathsf{N}(m_4) \circ\!\!- (\mathsf{N}(m_5) \parr a_4)))))$$

This suggests an alternative syntax for agents.

Needham-Schroeder revisited

$$
\begin{array}{ll}
\exists k_{as} \exists k_{bs}.[& \\
\text{(Out)} & \forall na.\mathsf{N}(\langle alice,\ bob,\ na\rangle) \circ\!\!- \\
\text{(In)} & \quad (\forall Kab \forall En.\mathsf{N}(kas\langle na,\ bob,\ Kab^{\circ},\ En\rangle) \circ\!\!- \\
\text{(Out)} & \quad\quad (\mathsf{N}(En) \circ\!\!- \\
\text{(In)} & \quad\quad\quad (\forall Nb.\mathsf{N}(Kab\ Nb) \circ\!\!- \\
\text{(Out)} & \quad\quad\quad\quad \mathsf{N}(Kab(Nb,\ secret))))). \\
 & \\
\text{(Out)} & \quad \bot \circ\!\!- \\
\text{(In)} & \quad\quad (\forall Kab.\mathsf{N}(kbs(Kab^{\circ},\ alice)) \circ\!\!- \\
\text{(Out)} & \quad\quad\quad (\forall nb.\mathsf{N}(Kab\ nb) \circ\!\!- \\
\text{(In)} & \quad\quad\quad\quad (\forall S.\mathsf{N}(Kab(nb,\ S)) \circ\!\!- \\
\text{(Cont)} & \quad\quad\quad\quad\quad b\ S))). \\
 & \\
\text{(Out)} & \bot \circ\!\!- \\
\text{(In)} & \quad\quad (\forall N.\mathsf{N}(\langle alice,\ bob,\ N\rangle) \circ\!\!- \\
\text{(Out)} & \quad\quad\quad (\forall key.\mathsf{N}(kas\langle N,\ bob,\ key^{\circ},\ kbs(key^{\circ},\ alice)\rangle)))). \\
] &
\end{array}
$$

The general setting for specifying agents

Let A denote atomic formulas. Consider

$$H = A \mid \bot \mid H \parr H \mid \forall x.\ H \qquad \text{(heads)}$$

$$K = H \mid H \circ\!\!- K \mid \forall x.\ K \qquad \text{(agents)}$$

Let $\mathcal{A}$ denote a multiset of atoms (ie, network messages). Let Γ and Δ be a multiset of "agents" (K-formulas), such that those in Γ are in output mode and those in Δ are in output mode.

The sequent $\Delta \longrightarrow \Gamma, \mathcal{A}$ captures the relationship between these three elements (nework messages are degenerated output processes).

The rules for implication introduction provide the basic dynamics:

$$\frac{H \longrightarrow \mathcal{A}_1 \qquad \Delta \longrightarrow K, \mathcal{A}_2}{\Delta, H \circ\!\!- K \longrightarrow \mathcal{A}_1, \mathcal{A}_2} \qquad \frac{\Delta, K \longrightarrow \Gamma, H, \mathcal{A}}{\Delta \longrightarrow H \circ\!\!- K, \Gamma, \mathcal{A}}$$

Left-introduction can be limited to sequents with atomic left-hand sides.

If in the definition of K-formulas above we write $H \circ\!\!- H$ instead of $H \circ\!\!- K$, we are restricting our selves to MSR (bipolars) again.

Intruders, Testing, and Interpolants

One approach to characterizer *intruders*, called *tests* here, is to say that they are essentially the same things as principles, except that they can halt a computation, using with the $\top$ logical connective:

$$W ::= \top \mid H \mid H \circ\!\!- W \mid \forall x.W.$$

P and Q are *testing equivalent* if for every multiset Γ of testers, $\vdash P, \Gamma$ iff $\vdash Q, \Gamma$.

Interpolants can be used to monitor communications across a boundary.

Classically, if $A \vdash B$ then an interpolant is a formula R such that $A \vdash R$ and $R \vdash B$ and all the non-logical constants in R occur in both A *and* B.

Interpolation Theorem. Let Γ be a set of role formulas (principals) and let Δ be a set of tests (intruders) such that $\vdash \Gamma, \Delta$. There is a formula R (the interpolant) such that the non-logical constants in R occur in both Γ and in Δ and is such that $\vdash \Gamma, R$ and $R \vdash \Delta$.

Tracing Communications

The interpolants needed in this theorem have the following structure:

$$M ::= \bot \mid A \mid M \parr M$$

$$R^+ ::= \top \mid \forall x.R^+ \mid M \multimap R^- \qquad R^- ::= M \multimap R^+ \mid \forall x.R^-.$$

Formulas in the R^+ syntactic category are called *traces*. Clearly, traces are in fact simple tests.

Two processes are *trace equivalent* if for every trace R, $\vdash P, R$ iff $\vdash Q, R$.

Theorem. Trace equivalent and testing equivalent coincide.

Proof. If P and Q are testing equivalent, it follows immediate that they are trace equivalent (every trace is a test).

Conversely, assume that P and Q are trace equivalent and that Γ is a set of testers such that $\vdash P, \Gamma$. By the Interpolation Theorem, we know that there is a trace R such that $\vdash P, R$ and $R \vdash \Gamma$. Since P and Q are trace equivalent, we know that $\vdash Q, R$ and by cut-elimination, we know that $\vdash Q, \Gamma$.

Conclusions

1. Linear logic can be used to specify the *execution* of security protocols.
2. Seeing encryption as an abstract datatype seems a powerful logical device to help reason about hiding information.
3. Abstraction of "continuation predicates" can transform bipolar (MSR) expressions into non-bipolar (process calculus expression) expressions.
4. Proof theoretical techniques have a use in reasoning about protocol correctness.
   (a) Cut-elimination is a basic tool.
   (b) Higher-type quantification makes protocols more declarative and offer new avenues for reasoning about protocols.
   (c) Interpolants can be used to characterize the interaction between agents and environments.
5. Related work: Sumii & Pierce, Logical relations for encryption, CSFW 2001.
6. Also: *strand spaces* seem to be simple graph-like structures definable via cut-free proofs.