Encryption as an Abstract Datatype:

observations about relating security protocols and proof search in linear logic

Dale Miller
Inria 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.

Reference: "Higher-order quantification and proof search," AMAST 2002. Also: Chapter 9 of lecture notes.

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 is:

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

More generally,

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

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

We formalize this using *multiset rewriting* of atomic 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 introduce the syntax:

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

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.

$$\begin{array}{lll} \vdots & & \\ \text{Message } i & \text{A} \longrightarrow \text{S:} & \{M\}_k \\ \text{Message } j & \text{S} \longrightarrow \text{A:} & \{P\}_k \\ \vdots & & \end{array}$$

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.

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

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)	$\otimes$	$\multimap$	$\exists$	$\forall$	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,” Computer Security Foundations Workshop, 1999.

Encrypted data as an abstract data type

Encryption keys are encoded as symbolic functions on data of type $\mathit{data} \to \mathit{data}$. Replace $\{M\}_k$ with $(k\ M)$.

Since such keys have scope, encrypted data is an abstract datatype.

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

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).

$$\exists\, k. \left[\begin{array}{r} a_1\ S \multimapinv \forall n.\ a_2\ \langle k^\circ, S\rangle \parr \mathrm{N}(k\ n) \\ a_2\ \langle k^\circ, S\rangle \parr \mathrm{N}(k\ M) \multimapinv \ldots \qquad\qquad\qquad\qquad \end{array} \right.$$

A Linear Logic Specification of Needham-Schroeder

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

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 Needham-Schroeder should be that Alice can communicate to Bob a secret with the help of a server. That is, the clause

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

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?

Automation of proof search

Automation of proof search must not include "invention". The subformula property is a good guide.

- Lemmas should not be automated: i.e., consider only cut-free proofs.
- Higher-order predicates substitutions must be "tame"; no automation of *invariants*.

Cuts can be avoided since linear logics satisfies the *cut elimination property*.

Higher-order substitutions have traditionally been avoided by restricting to first-order. But this is too draconian! It makes impossible admitting rich forms of abstractions.

Not all higher-order quantification is hard to automate and even the most simple forms can be a great asset when *reasoning about* logic programs.

Quantification rules

There are two ways $\forall$ is used in a proof: To prove a $\forall$, prove a generic instance of it. To use a $\forall$ assumption, make any instance of it.

$$\frac{\Sigma \vdash t : \tau \qquad \Sigma : \Gamma, B[t/x] \longrightarrow \Delta}{\Sigma : \Gamma, \forall_\tau x.B \longrightarrow \Delta}\ \forall\mathcal{L} \qquad \frac{y : \tau, \Sigma : \Gamma \longrightarrow B[y/x], \Delta}{\Sigma : \Gamma \longrightarrow \forall_\tau x.B, \Delta}\ \forall\mathcal{R}$$

If a $\forall$ appears on the right (positively), then replace it with a new constant. Even in the higher-order setting, this is a trivial operation.

If a $\forall$ appears on the left (negatively), then replace with some substitution term. The choice of substitution term is generally determined using unification.

Dual statements can be made for the $\exists$ quantifier.

Scheme for reasoning about logic programs

Here, higher-order quantification will be featured in two ways.

- During computation (proof search) higher-order quantification will be "easy": instantiate with new symbols.
- When reasoning about computations, higher-order quantification can be hard and require clever substitutions.

One approach to reasoning about logic programs is the following:

$P \vdash G$	proof search (cut-free)
$P' \vdash P$	reasoning about programs: involves rich substitutions and lemmas
$P' \vdash G$	after cut-elimination, we have a computation again

Note that P appears both positively and negatively. What corresponded to "generate a new predicate" corresponds of "find a logical expression for 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

$(apply\ k\ M)$, or more appropriately, as $(encrypt\ k\ M)$.

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:

$$a \multimap \forall k.\mathrm{N}(k\ m) \quad \text{and} \quad a \multimap \forall k.\mathrm{N}(k\ m').$$

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

$$a \multimap \forall k.\mathrm{N}(k\ m) \dashv\vdash a \multimap \forall k.\mathrm{N}(k\ m').$$

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

$$\forall k.\mathrm{N}(k\ m) \vdash \forall k.\mathrm{N}(k\ m')$$

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.

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

Intermediate states of an agent can be taken out entirely.

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

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

This suggests an alternative syntax for agents.

Needham-Schroeder revisited

$$
\begin{array}{ll}
\multicolumn{2}{l}{\exists k_{as}\exists k_{bs}.[} \\
(\mathrm{Out}) & \forall na.\mathrm{N}(\langle alice,\ bob,\ na\rangle)\multimapinv \\
(\ \mathrm{In}\ ) & \quad(\forall Kab\forall En.\mathrm{N}(kas\langle na,\ bob,\ Kab^{\circ},\ En\rangle)\multimapinv \\
(\mathrm{Out}) & \qquad(\mathrm{N}(En)\multimapinv \\
(\ \mathrm{In}\ ) & \qquad\quad(\forall Nb.\mathrm{N}(Kab\ Nb)\multimapinv \\
(\mathrm{Out}) & \qquad\qquad\mathrm{N}(Kab(Nb,\ secret))))). \\
 & \\
(\mathrm{Out}) & \bot\multimapinv \\
(\ \mathrm{In}\ ) & \quad(\forall Kab.\mathrm{N}(kbs(Kab^{\circ},\ alice))\multimapinv \\
(\mathrm{Out}) & \qquad(\forall nb.\mathrm{N}(Kab\ nb)\multimapinv \\
(\ \mathrm{In}\ ) & \qquad\quad(\forall S.\mathrm{N}(Kab(nb,\ S))\multimapinv \\
(\mathrm{Cont}) & \qquad\qquad b\ S))). \\
 & \\
(\mathrm{Out}) & \bot\multimapinv \\
(\ \mathrm{In}\ ) & \quad(\forall N.\mathrm{N}(\langle alice,\ bob,\ N\rangle)\multimapinv \\
(\mathrm{Out}) & \qquad(\forall key.\mathrm{N}(kas\langle N,\ bob,\ key^{\circ},\ kbs(key^{\circ},\ alice)\rangle)))). \\
 &]
\end{array}
$$

Two classes of connectives

The logical connectives of linear logic can be classified as

- asynchronous $\bot$, $\invamp$, $\forall$, ... The right introduction rules for these are invertible. These rules yield structural equivalences.
- synchronous 1, $\otimes$, $\exists$, ... The right introduction rules for these are not invertible. These rules yield interaction with the environment.

These connectives are De Morgan duals of each other.

We shall only write asynchronous connectives but write them on both sides of the sequent arrow (yielding both behaviors). We also use implications:

$$B \multimap C \equiv B^{\perp} \invamp C \qquad \text{and} \qquad B \Rightarrow C \equiv\, !\, B \multimap C$$

Alternation of synchronous and asynchronous connectives

A *bipolar* formula is a formula in which no asynchronous connectives is in the scope of a synchronous connective. That is, there is an outer layer of asynchronous connectives followed by an inner layer of synchronous connectives.

The multiset rewriting clauses are bipolars, for example,

$$a \parr b \multimapinv c \parr d \equiv a \parr b \parr (c^{\perp} \otimes d^{\perp}).$$

Andreoli showed how to compile arbitrary alternation of syn/asyn connectives into bipolars by introducing new predicate symbols. He also argued for only using bipolars for proof search.

Avoiding bipolars has some advantages

Only one predicate is need, namely, N(·). The other predicates (used as “line numbers” in a protocol) are not needed.

Agents now resemble process calculus expressions. The formula $a \multimapinv (b \multimapinv (c \multimapinv (d \multimapinv k)))$ can denote either

$$\bar{a} \,||\, (b.\ (\bar{c} \,||\, (d.\ \ldots))) \quad \text{or} \quad a.\ (\bar{b} \,||\, (c.\ (\bar{d} \,||\, \ldots)))$$

depending on if it appears on the right or the left of a sequent. Writing this expression without the implication:

$$a \parr (b^\perp \otimes (c \parr (d^\perp \otimes \ldots))) \quad \text{resp,} \quad a^\perp \otimes (b \parr (c^\perp \otimes (d \parr \ldots)))$$

There is a strict alternation of input and output phases. If an agent skips a phase, the adjacent phases can be merged:

$$a \multimapinv (\perp \multimapinv (b \multimapinv k)) \equiv (a \parr b) \multimapinv k.$$

The general setting for specifying agents

$$A = \text{ atomic formulas}$$

$$H = A \mid \bot \mid H \parr H \mid \forall x.\ H$$

$$K = H \mid H \multimapinv K \mid \forall x.\ K$$

Let $\mathcal{A}$ denote a multiset of atoms (ie, network messages). Let Δ and Γ be a multiset of "agents" (K-formulas). Since Δ will appear on the right, it contains outputting agents and since Γ will appear on the left, it contains inputting agents.

Two rules related to agents are then given as follows:

$$\frac{\Gamma, K \vdash \Delta, H, \mathcal{A}}{\Gamma \vdash H \multimapinv K, \Delta, \mathcal{A}} \qquad \frac{H \vdash \mathcal{A}_1 \qquad \Gamma \vdash K, \mathcal{A}_2}{\Gamma, H \multimapinv K \vdash \mathcal{A}_1, \mathcal{A}_2}$$

The left rule can be limited to sequents with atomic rhs.

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

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.
   - 4.1 Cut-elimination is a basic tool.
   - 4.2 Higher-type quantification makes protocols more declarative and offer new avenues for reasoning about protocols.
5. Related work: Sumii & Pierce, Logical relations for encryption, CSFW 2001.