

Formal methods for security protocols

Design and analysis methods

(reminder)

There are now several methods for describing and verifying security protocols, based on:

- rigorous but informal frameworks,
- temporal logics,
- process algebras (CSP, the pi calculus),
- theorem-proving tools,
- special-purpose formalisms.

Protocol descriptions

We may specify a protocol as a process, or as the corresponding set of behaviors.

In addition to the expected participants, we should model an attacker, who:

- may participate in some protocol runs,
- knows some data in advance,
- may intercept messages on the public network,
- may inject any messages that it can produce into the public network.

The origin of formal cryptology

What messages can the attacker produce?

- The attacker can be non-deterministic.
- But it cannot get too lucky in guessing keys.

A simple solution:

- Arrange that non-deterministic choice of a key always yields a fresh key.
- Distinguish keys from bitstrings.
- Treat cryptographic operations symbolically.

Common assumptions

Some informal assumptions commonly underlying formal methods, e.g., for symmetric encryption:

- Given K , anyone can compute $\{M\}_K$ from M and vice versa.
- $\{M\}_K$ cannot be produced by anyone who does not know M and K .
- $\{M\}_K$ cannot be understood by anyone who does not know K .
- Decryption with an incorrect key is evident.

The formal view: strengths

An often appropriate level of abstraction.

- Some simple, effective intuitions.
- Easy human reasoning (sometimes).
- “Enough” interesting detail.

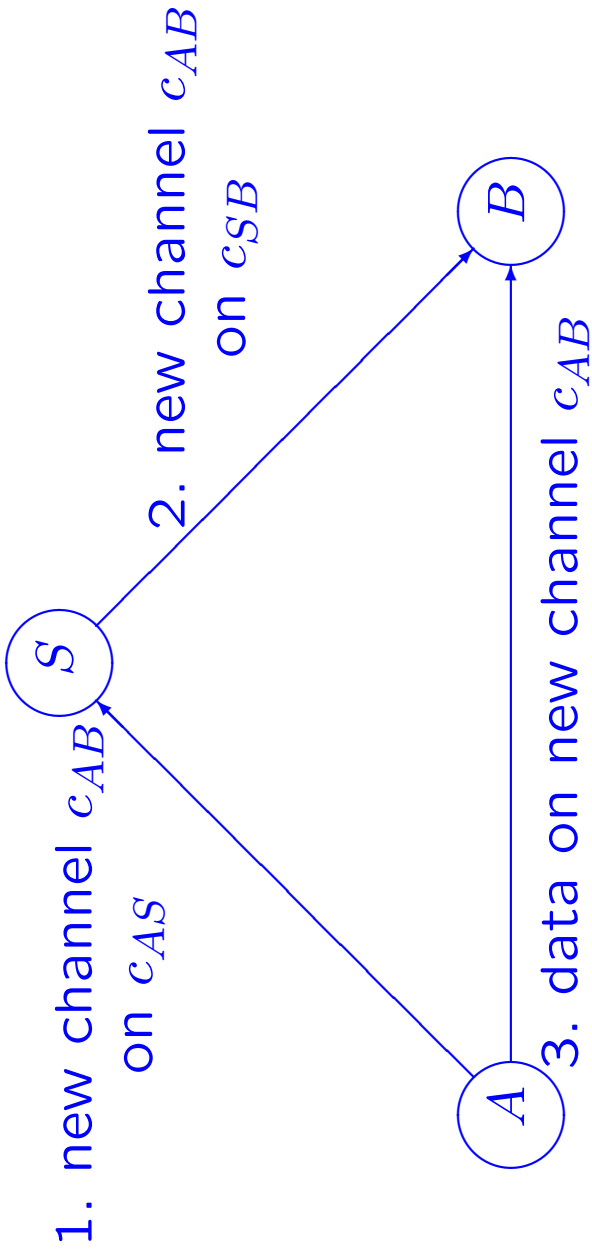
Techniques and tools for automated reasoning.

Logical methods and foundations
(in modal logics, process algebras, ...).

The Applied Pi Calculus

A typical protocol

Establishment and use of a secure channel:

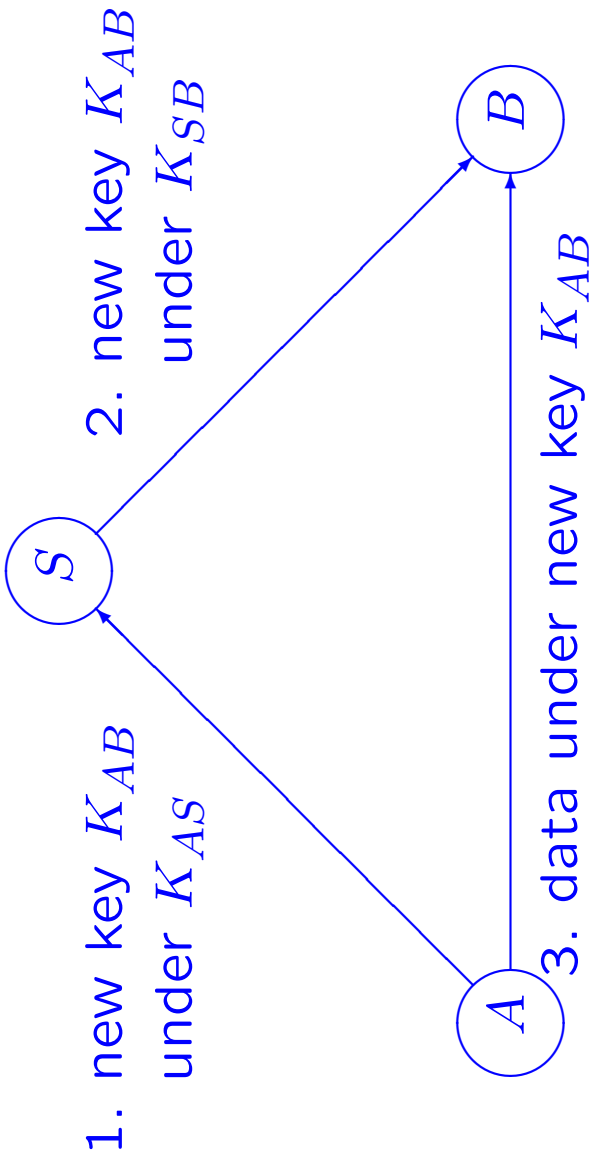


A and B : two clients S : an authentication server
 c_{AS} and c_{SB} : channels with the server
 c_{AB} : a new channel for the clients

The pi calculus should help with such protocols.

A typical protocol, in more detail

Establishment and use of a secure channel:



A and B: two clients S: an authentication server

K_{AS} and K_{SB} : shared keys with the server

K_{AB} : a new session key for the clients

The applied pi calculus

(joint work with C. Fournet)

applied pi calculus = pi calculus + function symbols

The pi calculus part is enough

- for general programming,
- for manipulating secure channels abstractly.

The functions enable us to show more detail, e.g.,

- constructing a secure channel by encryption,
- deriving a key from another key.

Syntax of a pi calculus

We start out with a sort of variables (x) and a sort of names (n).

We define a sort of terms (data):

$$M, N ::= \quad \text{terms} \\ \quad | \quad x \quad \text{variable} \\ \quad | \quad n \quad \text{name}$$

We let u range over variables and names.

Syntax of a pi calculus (cont.)

We also define a sort of processes:

$P, Q ::=$	processes
nil	nil process
$\bar{u}\langle N \rangle.P$	sending
$u(x).P$	receiving
$!P$	replication
$P \mid Q$	parallelism
$(\nu n)P$	restriction

nil may be omitted.

Secrecy from scoping



$$A(M) \triangleq \bar{c}\langle M \rangle$$

$$B \triangleq c(x).nil$$

$$P(M) \triangleq (\nu c)(A(M) \mid B)$$

P represents a protocol where

- A sends M to B over a channel c ,
- B swallows its input,
- communication is secure: only A and B can access c .

Secrecy as equivalence

We obtain a secrecy property:

$P(M)$ and $P(N)$ are equivalent,
for all M and N .

- Almost any definition of equivalence (e.g., bisimilarity) will do in this example.
- The equivalence should mean that no attacker can distinguish $P(M)$ and $P(N)$.
- The restriction (νc) is crucial: without it, an attacker $c(x)$... could get the message.

Secrecy: an alternative formulation

For the special case where M is a name n :

No attacker can learn n from $P(n)$:

for all Q in which n does not occur,
 Q will not get n from $P(n)$ in $P(n) \mid Q$.

Equivalently (but more precisely):

for all Q in which n does not occur free,
 $P(n) \mid Q$ will never output n .

Initial comments

We can write down some protocols.

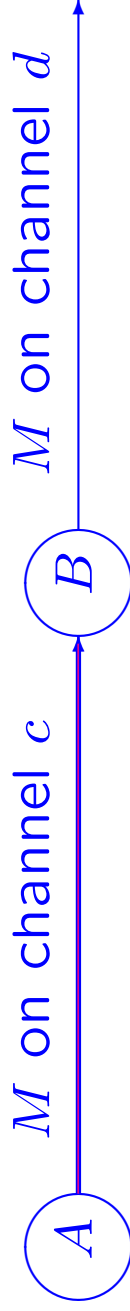
We can express some security properties, using tools from process calculi.

We can model an (arbitrary) attacker as a process Q .

This process runs in parallel with our protocol, and may interact with it.

Authenticity from scoping

Here B forwards its input over a public channel d .



$$A(M) \triangleq \bar{c}\langle M \rangle$$

$$B \triangleq c(x).\bar{d}\langle x \rangle$$

$$P(M) \triangleq (\nu c)(A(M) \mid B)$$

An authenticity property

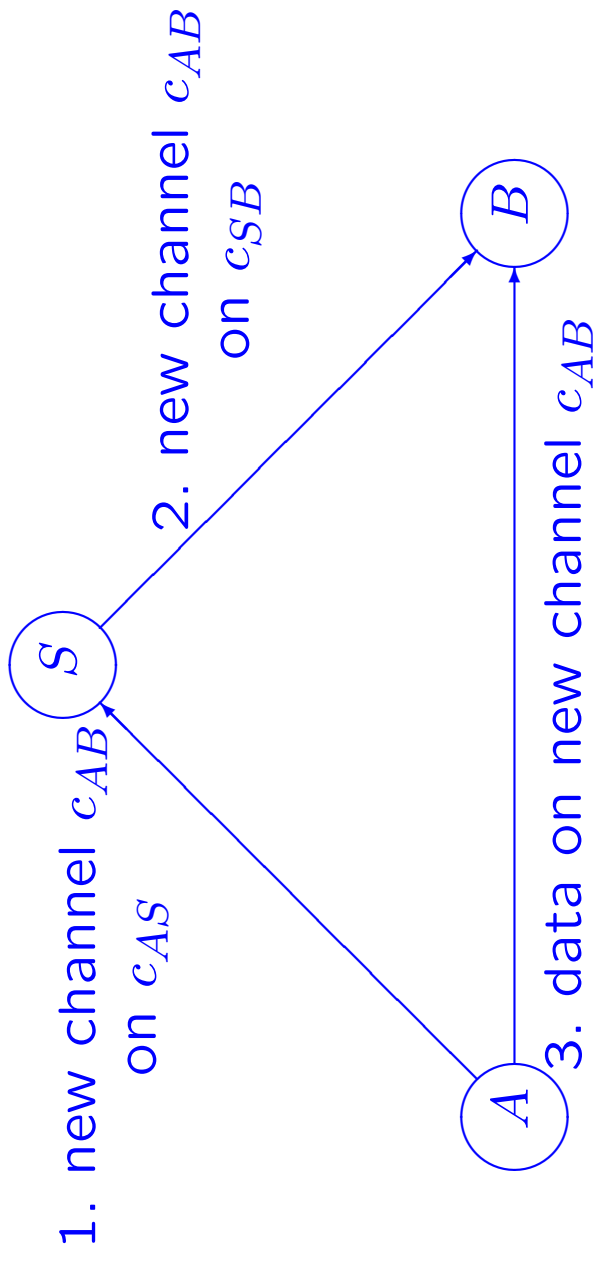
We obtain an authenticity property:

for all N ,
if B outputs N on d
then A sent N to B .

This is an ordinary safety property.

(Authenticity can also be paraphrased as an equivalence.)

Passing channels: a typical protocol, revisited



An abstract version of the protocol:

$$A(M) \triangleq (\nu c_{AB}) \overline{c_{AS}} \langle c_{AB} \rangle . \overline{c_{AB}} \langle M \rangle$$

$$S \triangleq c_{AS}(x) . \overline{c_{SB}} \langle x \rangle$$

$$B \triangleq c_{SB}(x) . x(y) . \overline{d} \langle y \rangle$$

$$P(M) \triangleq (\nu c_{AS}) (\nu c_{SB}) (S \mid A(M) \mid B)$$

- A makes up a channel c_{AB} and sends it to server S on preexisting channel c_{AS} ,
- then S forwards c_{AB} to B on preexisting channel c_{SB} ,
- then A sends M to B on c_{AB} ,
- and finally B outputs its input on d .

We can add concurrent sessions.

For example:

$$A(M) \triangleq (\nu c_{AB}) \overline{c_{AS}} \langle c_{AB} \rangle . \overline{c_{AB}} \langle M \rangle$$

$$S \triangleq !c_{AS}(x) . \overline{c_{SB}} \langle x \rangle$$

$$B \triangleq !c_{SB}(x) . x(y) . \overline{d} \langle y \rangle$$

$$P(M_1, \dots, M_n) \triangleq (\nu c_{AS}) (\nu c_{SB}) (S \mid A(M_1) \mid \dots \mid A(M_n) \mid B)$$

The pi calculus and security

The channels of the pi calculus have nice built-in properties, such as:

- integrity,
- confidentiality,
- exactly-once semantics,
- mobility,
- forward secrecy.

These properties are useful in protocol descriptions.

The applied pi calculus retains these properties.

Toward the applied pi calculus

We wish to express cryptographic operations.

- Interestingly, we can encode some forms of encryption in the pi calculus.
- In the end, we added primitives for certain operations.

This resulted in the **spi calculus** (work with A. Gordon).

The **applied pi calculus** is a more general extension, with a generic treatment of function symbols.

Applied lambda calculus

We start out with

- a sort of variables (x),
- a set of function symbols, such as f and pair , each with an arity.

We may give types and equations for the functions.

We define the sort of terms:

$M, N ::=$		terms
	x	variable
	$\lambda x.M$	abstraction
	$M(N)$	application
	$f(M_1, \dots, M_k)$	function application

Syntax of the applied pi calculus

We start out with

- a sort of variables (x),
- a sort of names (n),
- a set of function symbols, such as f , encrypt , and pair , each with an arity.

We define the sort of terms (data):

$M, N ::=$		terms
	x	variable
	n	name
	$f(M_1, \dots, M_k)$	function application

Syntax (cont.): processes

We define the sort of processes:

$P, Q ::=$	processes
nil	nil process
$\bar{u}\langle N \rangle.P$	sending
$u(x).P$	receiving
$!P$	replication
$P \mid Q$	parallelism
$(\nu n)P$	restriction
$\text{if } M = N \text{ then } P \text{ else } Q$	conditional

Syntax (cont.): types for terms

A simple type system for terms:

- a set of base types, such as Integer, Key, or simply a universal base type Data,
- a type $\text{Channel}(\tau)$ for those channels that convey messages of type τ .

For simplicity, function symbols take arguments and produce results of the base types only.

Equations

Given a signature Σ , we equip it with an equational theory, that is, with an equivalence relation on terms with some closure conditions.

We write $\Sigma \vdash M = N$ when the equation $M = N$ is in the theory associated with Σ .

We write $\Sigma \not\vdash M = N$ for the negation of $\Sigma \vdash M = N$.

Examples: pairs

Σ includes pair, fst, and snd, with the equations:

$$\text{fst}(\text{pair}(x, y)) = x$$

$$\text{snd}(\text{pair}(x, y)) = y$$

We may write (M, N) for $\text{pair}(M, N)$.

Examples: shared-key encryption

Σ includes the binary symbols senc and sdec , with the equation:

$$\text{sdec}(\text{senc}(x, y), y) = x$$

Dealing with errors

Types cannot statically avoid all “errors”.

We may add a constant wrong, with equations such as:

$$\text{sdec}(M, N) = \text{wrong}$$

for ground terms M and N such that $\Sigma \not\vdash M = \text{senc}(M', N)$ for any M' .

We may then code derived constructs, such as:

$$\text{case } N \text{ of } \{x\}_M \text{ in } P \triangleq \begin{array}{l} \text{if } \text{sdec}(N, M) = \text{wrong} \text{ then } \text{nil} \text{ else } P[\text{sdec}(N, M)/x] \end{array}$$

(This is the construct for decryption in the spi calculus.)

Dealing with errors (cont.)

An alternative is to add checking constructs, as in:

`scheck(senc(M' , N), N) = ok`

Then we may leave `sdec(M , N)`, for M not of the form `senc(M' , N)`, without any equations.

Checking constructs also help elsewhere, for example in checking signatures.

Examples: public-key encryption

Σ includes unary function symbols pk and sk for generating public and secret keys from a seed, and the equation:

$$pdec(penc(x, pk(y)), sk(y)) = x$$

For simplicity, we may omit sk .

Optionally, we may for example add:

$$pdec(penc(x, y), z) = penc(pdec(x, z), y)$$

Examples: one-way hashing

Σ includes a unary symbol hash, with no equations.

$\text{hash}(M)$ represents the result of applying a one-way hash function to M .

Other easy examples

Nondeterministic (“probabilistic”) encryption.

Digital signatures.

Message authentication codes.

Exponentiation as used in Diffie-Hellman.

XOR.

⋮

Operational semantics

Structural equivalence \equiv is defined much as usual.

It is also closed by substitution of equal terms.

So:

$$P \mid nil \equiv P$$

$$P \mid Q \equiv Q \mid P$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$!P \equiv P \mid !P$$

$$(\nu m)(\nu n)P \equiv (\nu n)(\nu m)P$$

$$(\nu n)nil \equiv nil$$

$$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin \text{fn}(P)$$

$$P[M/x] \equiv P[N/x] \quad \text{if } \Sigma \vdash M = N$$

Operational semantics (cont.)

Reduction \rightarrow is the smallest relation on closed processes that is closed by structural equivalence and application of evaluation contexts such that:

$$\bar{a}\langle M \rangle.P \mid a(x).Q \rightarrow P \mid Q[M/x]$$

$$\text{if } M = M \text{ then } P \text{ else } Q \rightarrow P$$

$$\text{if } M = N \text{ then } P \text{ else } Q \rightarrow Q$$

for any ground terms M and N

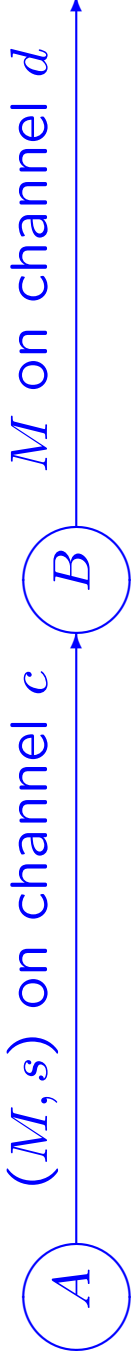
such that $\Sigma \not\vdash M = N$

Names as data

In the pi calculus, channels can be dynamically created and communicated.

In the applied pi calculus, keys, nonces, and other names can be dynamically created and communicated as well.

Names as data: example



$$A(M) \triangleq \bar{c}\langle(M, s)\rangle$$

$$B \triangleq c(x).if\ snd(x) = s\ then\ \bar{d}\langlefst(x)\rangle$$

$$P(M) \triangleq (\nu s)(A(M) \mid B)$$

P represents a protocol where

- A sends M to B tagged with s over a public channel c ,
- then, if the tag matches, B outputs its input on another channel d .

So s serves as a capability, but can be intercepted.

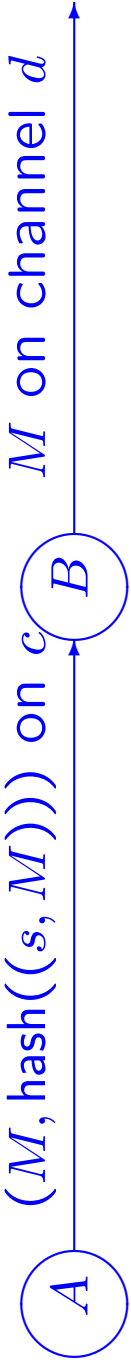
Interesting extrusions

In the applied pi calculus, a process may reveal a term that contains a fresh name s without revealing s itself.

This possibility does not arise in the pure pi calculus.

It is a source of expressiveness and complications.

Interesting extrusions: example



$$A(M) \triangleq \bar{c}\langle(M, \text{hash}((s, M)))\rangle$$

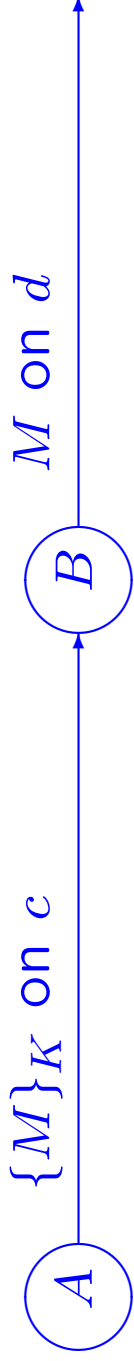
$$B \triangleq c(x).\text{if } \text{hash}((s, \text{fst}(x))) = \text{snd}(x) \text{ then } \bar{d}\langle\text{fst}(x)\rangle$$

$$P(M) \triangleq (\nu s)(A(M) \mid B)$$

Although $(M, \text{hash}((s, M)))$ travels on the public channel c , no other process can extract s from this, or produce $(N, \text{hash}((s, N)))$ for some other N .

So only the intended term M will be forwarded on d .

Another variant



$$A(M) \triangleq \bar{c}\langle \text{senc}(M, K) \rangle$$

$$B \triangleq c(x). \text{case } x \text{ of } \{y\}_K \text{ in } \bar{d}\langle y \rangle$$

$$P(M) \triangleq (\nu K)(A(M) \mid B)$$

Bigger examples (sketch)

Suppose we have n clients plus a server, and m instances I_1, \dots, I_m (sender, receiver, message). We may write:

$$\begin{aligned} \text{Send}(i, j, M) &\triangleq \overline{c_S} \langle i \rangle \mid \\ &\quad c_i(x_{\text{nonce}}). \\ &\quad (\nu K_{AB})(\overline{c_S} \langle \text{senc}((i, i, j, K_{AB}, x_{\text{nonce}}), K_{iS}) \rangle \mid \\ &\quad \overline{c_j} \langle (i, \text{senc}(M, K_{AB})) \rangle) \\ \\ \text{Recv}(j) &\triangleq \dots \overline{d} \langle y \rangle \\ \\ S &\triangleq \dots \\ \\ \text{Sys}(I_1, \dots, I_m) &\triangleq (\nu K_{1S}) \dots (\nu K_{nS}) \\ &\quad (\nu K_{S1}) \dots (\nu K_{Sn}) \\ &\quad (\text{Send}(I_1) \mid \dots \mid \text{Send}(I_m) \mid \\ &\quad !S \mid \\ &\quad !\text{Recv}(1) \mid \dots \mid !\text{Recv}(n)) \end{aligned}$$

Again:

- Scoping is the basis of security.
- We can discuss security properties formally.

The notion of equivalence becomes delicate:

$(\nu K)\bar{c}\langle \text{send}(M, K) \rangle$ and $(\nu K)\bar{c}\langle \text{send}(N, K) \rangle$ send
different messages,
but they should be equivalent.

This is important for getting a sensible criterion for security properties.

Defining equivalence

Two processes are equivalent if no environment can distinguish them.

Technically, we use a testing equivalence or observational congruence.

Defining equivalence (cont.)

A definition is:

A **test** is a process R plus a channel name w .

Intuitively, R is the environment and w is used for signaling the test outcome.

A process P **passes** a test (R, w) if $P \mid R$ may communicate on w .

Two processes are **equivalent** if they pass the same tests.

Observational congruence pays attention to branching structure, in addition.

Features of these equivalences

Equivalence captures observational indistinguishability, with respect to an implicit, arbitrary attacker process.

Equivalence is “coarse enough” .

When an equivalence fails, we can often find an attack.

Equivalence is a congruence.

Equivalence can be hard to prove! It is useful to develop proof techniques.

Using the applied pi calculus

Like the spi calculus, the applied pi calculus is rather abstract.

- We are able to ignore details.
- In particular, we ignore details of encryption.

Using the applied pi calculus (cont.)

The applied pi calculus is also expressive.

- We can represent process behavior and cryptography.
- We need not commit to a fixed cryptosystem.
- We can describe:
 - every message,
 - how it is acted on, and
 - under what circumstances it is sent.

It allows expressing and proving security properties.

It may serve as a basis for higher-level reasoning.

Further developments

Study of equivalences.

Type systems.

Decision procedures for some problems.

Automated analysis via logic programs.

Other proof techniques and tools.

Partial complexity-theoretic foundations.

Examples.

Logics? Models?