

Security and Security Protocols

Martín Abadi

University of California, Santa Cruz

Contents

A brief overview of security problems, models, and mechanisms, in particular:

- access control,
- authentication,
- security protocols.

A formal calculus for protocol analysis:
the applied pi calculus.

Not:

- a course on cryptography,
- a balanced introduction to all of security.

(See also: F. Pottier, C. Fournet.)

Reading

Two background books (on the web):

- [The handbook of applied cryptography](#)
- www.cacr.math.uwaterloo.ca/hac/index.html
- [Trust in cyberspace](#)
- www.nap.edu/readingroom/books/trust/.

Some papers (and the references there!).

Unintended behavior

Often systems do not behave as we intend.

The unintended behaviors can be traced to:

- environmental disruption,
- operator errors,
- poor design or implementation (bugs),
- deliberate attacks.

These problems mean that systems don't meet their functional requirements.

Unintended behavior (cont.)

Some approaches to addressing these problems are:

- environmental disruption,
 - ⇒ stronger interfaces,
 - ⇒ replication,
- operator errors,
 - ⇒ operator-tolerance,
 - ⇒ operator education,
- poor design or implementation (bugs),
 - ⇒ languages and tools,
 - ⇒ testing,
 - ⇒ verification,
- deliberate attacks,
 - ⇒ lower expectations,
 - ⇒ ???

The unchanging nature of security

Security for computer systems is much like security in the rest of the real world.

Security is not black and white.

It is not about perfect defenses against well-funded, capable, and determined attackers.

You don't have to run faster than the tiger,
you only have to run faster than your friends.

The unchanging nature of security (cont.)

Security is about:

- value:
 - sometimes a simple figure
 - not always easy to calculate
- locks:
 - not always cheap, seldom convenient
 - imperfect
- detecting attacks:
 - not always possible
 - not always possible in real time
- catching and punishing attackers
 - hard at a distance!

A trend toward vulnerable systems

Some increasingly common system characteristics enable attacks (and aggravate other problems).

- Interaction with an uncertain physical environment.
E.g., for a laptop in the enemy's hands.
- Interaction with an uncertain network environment.
Everything is connected.
- Interaction with an uncertain software environment.
E.g., with mobile code in Web pages.
E.g., with an untrusted media player.

A trend toward vulnerable systems (cont.)

- Use of an open infrastructures.
E.g., of the phone network.
- Distributed administration.
No central design and control.
- Diverse operators.
Everyone is an operator.
- Automation.
Including automated infection!

A trend toward vulnerable systems (cont.)

- Building from COTS components
(COTS = commercial, off-the-shelf).
E.g., from x86.
- Importance of time to market.
And the market seldom pays for security.
- Monocultures.
E.g., homogeneous systems based on Windows and x86.

A trend toward vulnerable systems (cont.)

These characteristics are unlikely to just go away:

- They are the result of fundamental economic or technical trends.
- Many are generally desirable.

Policies and Mechanisms

Security properties

The main security properties are:

- **Integrity**
(no improper modification of information)
- **Secrecy**
(no improper disclosure of information)
- **Availability**
(no improper denial of service)

Variations on integrity

Authenticity is often the same as integrity, with a difference only in emphasis.

Other concepts are closely related to integrity:

- **non-repudiation**,
- **accountability**.

Variations on secrecy

Similarly, **confidentiality** is basically secrecy.

So is **privacy**, though it is applied differently, particularly in the context of personal information.

Anonymity is basically an instance of secrecy.

Pseudonymity is anonymity plus linkability.

Plausible deniability is the contrary of non-repudiation and might be viewed as a weak form of secrecy.

Security policies

Security properties are combined into security policies.

For example, a bank may want:

- authenticity of clients at ATMs and on the Web,
- non-repudiation of transactions,
- integrity of the books,
- integrity of the messaging systems,
- secrecy for client data and for internal data,
- availability of the alarm system.

The conjunction of these properties may be the bank's security policy.

Security policies (cont.)

There is no unique definition of security.

- Different security policies are appropriate for different users, organizations, and systems.
- In a composite system, we may find conflicting desires, e.g., non-repudiation and plausible deniability.

Common themes

- Interaction with an uncertain environment.
(Contrast with mutual exclusion.)
- Some security even against lucky, powerful, or persistent attackers.
 - Even if the attacker controls the network.
 - Even if a secret is compromised.
 - Even if an insider is dishonest.
- Doing without full functional correctness.
E.g., message origin, not message correctness.

Principals

A lot of security properties concern principals or subjects:

- users,
- computers,
- services.

The notion of principal varies (dangerously) across systems and abstraction layers:

- IP addresses,
- the computers at those addresses,
- the people who control the computers,
- ⋮

Access control

Access control is prominent at many levels:

- memory management hardware,
 - operating systems, file systems, and related services,
 - middleware,
 - applications,
 - in particular, sandboxing for mobile code,
- and (with separate bodies of techniques):
- firewalls,
 - physical protection.

It is a mechanism and a model.

The access control model

Elements:

- **objects**, resources
- **requests** for **operations** on objects
- sources for requests, called **principals**
- a **reference monitor** to decide on requests

Authentication

Access control:

Is A trusted on s ?

If A requests s , should s be granted?

Access control is based on **authentication**:

Who says s ?

The principle of complete mediation

Every access to every object is checked.

This principle can be enforced in several ways:

- The OS intercepts some of the subject's requests.
The hardware catches others.
(E.g., as in Unix.)
- A software wrapper / interpreter intercepts some of the subject's requests.
(E.g., as in the JVM.)

Implementing access control: ACLs

An **access control list** says which subjects can access a particular object.

It is a column of an access control matrix, typically maintained “near” the object that it protects.

- ACLs can be compact.
- ACLs can be easy to review.
- Revoking a subject can be painful.

Implementing access control: capabilities

An alternative is to associate **capabilities** with subjects.

These capabilities form a row of an access control matrix for the subject.

- Capabilities are easy to pass around, so they enable delegation.
- They can be hard to revoke.

Implementing capabilities

A capability should convey an object and an operation. It means that the holder can perform the operation on the object.

Subjects should not be allowed to forge capabilities.

This leads to implementations of capabilities:

- stored in a protected address space,
- with special tags with hardware support,
- as references in a typed programming language,
- as passwords,
- with cryptographic certificates.

Comparisons

ACLs and capabilities are dual.

Both yield practical implementations of access matrices.

In actual systems, they are often combined.

Much more on access control

Groups and roles.

Delegation.

Running programs.

Theory.

Particular embodiments in systems.

Common dangers and precautions.

Security Protocols

Security protocols

Security protocols are concerned with properties such as integrity and secrecy.

- Primary examples are protocols that establish communication channels with authentication and confidentiality.
- Other examples include protocols for commerce and electronic voting.

In distributed systems, security protocols invariably rely on cryptography.

They have to be right, but they often aren't.

Cryptography

One-way hashing (e.g., MD5, SHA).

Shared-key cryptography (e.g., DES, AES, RC4):

- Two principals know a key.
- Both principals use the key both for encrypting and for decrypting.

Public-key cryptography (e.g., RSA):

- Each principal has a secret key, that it uses for signing and for decrypting.
- The inverse of the secret key is a public key, for checking signatures and for encrypting.

Bits of wisdom (?)

Cryptography is not broken, it is circumvented.

(Shamir according to Rivest)

Bits of wisdom (?) (cont.)

If you think that cryptography is the answer to your problem then you don't understand cryptography and you don't understand your problem.

(Needham or Lampson)

Cryptography and security

Cryptography is not the same as security.

But the security applications of cryptography are so significant that they have shaped both fields.

Authentication protocols (or channel-establishment protocols)

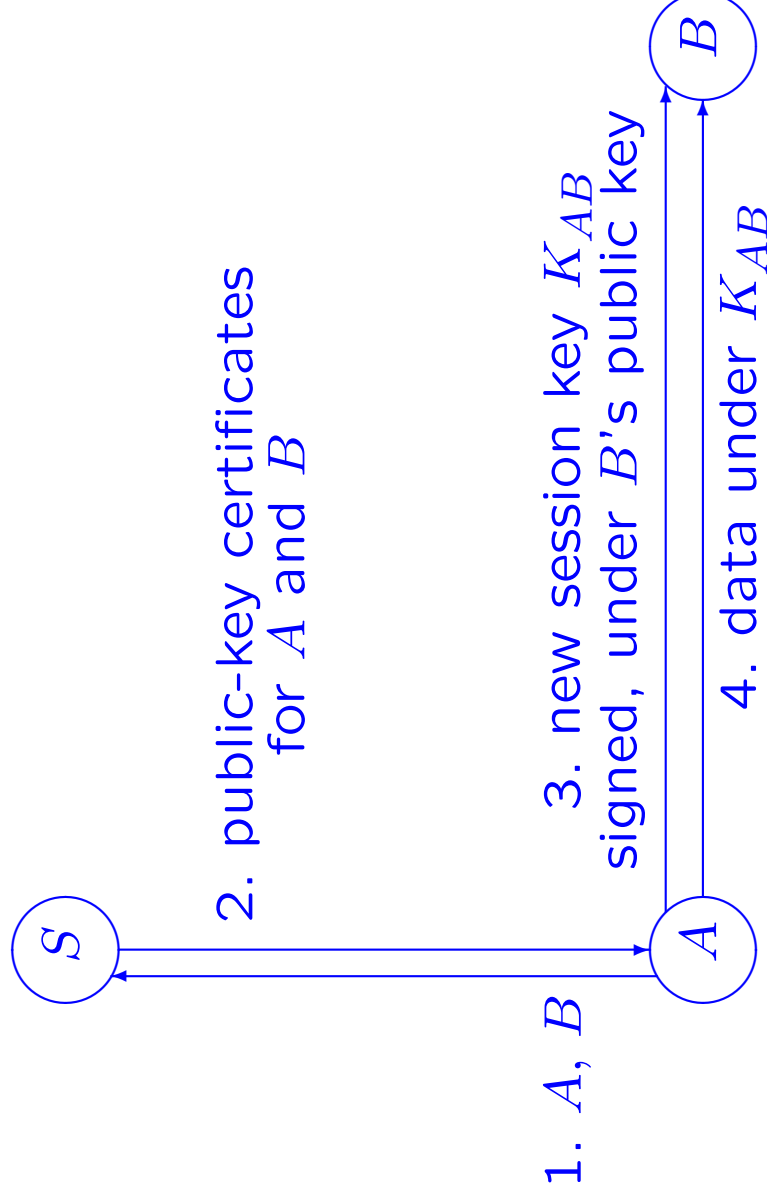
There are many authentication protocols.

They typically involve:

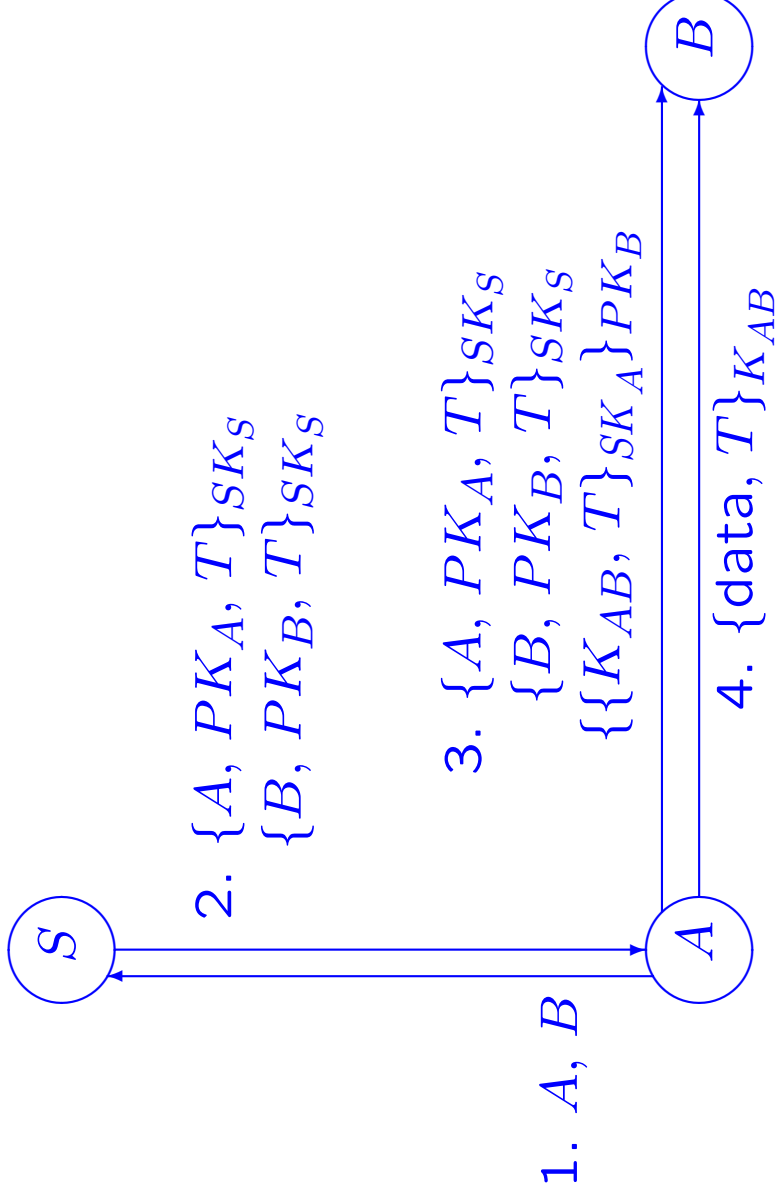
- two principals
 - hosts, users, services
- secrets (possibly shared)
 - usually encryption keys
- encryption
 - shared-key or public-key
- trusted servers
- proofs of timeliness
 - nonces, timestamps

A typical protocol

- Each principal has a public-key pair.
- S is a **certification authority** for public keys.
- A and B agree on a **session key** K_{AB} and send data under K_{AB} :



The Denning-Sacco protocol



$\{X\}_{PK_B}$: X encrypted with B 's public key, for secrecy

$\{X\}_{SK_A}$: X signed with A 's secret key, for integrity

T : a timestamp

Questions

Does the protocol work?

What assumptions are we making?

Are there redundant messages or encryptions?

No single protocol will do.

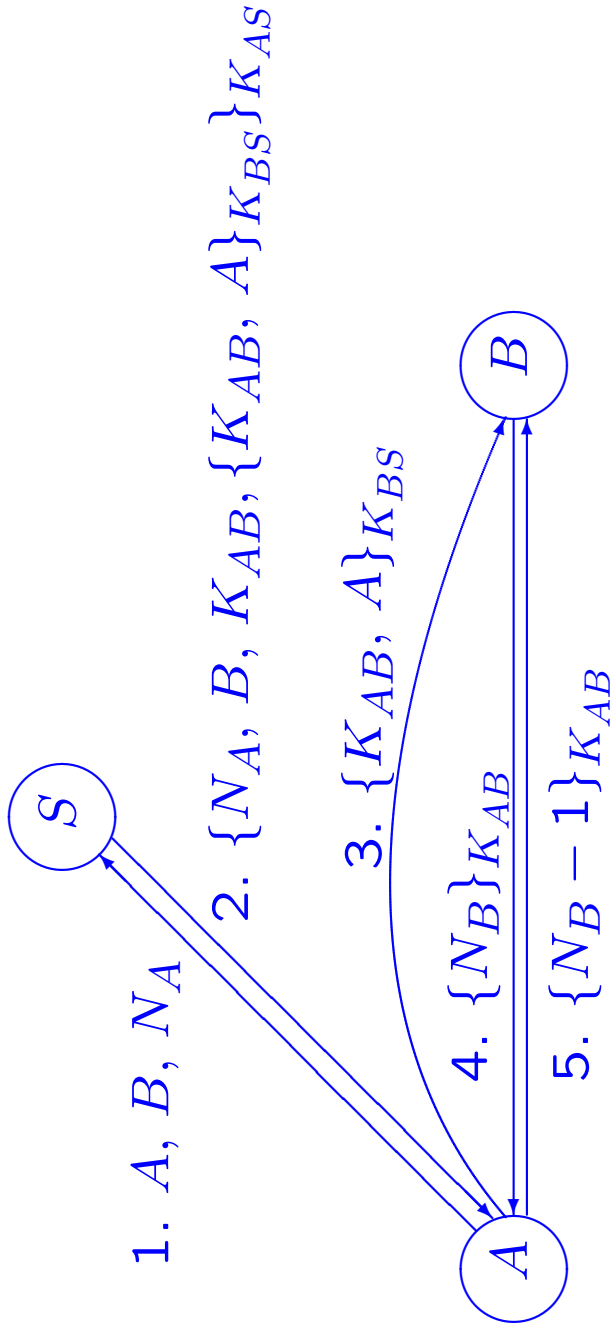
We may want:

- few messages,
- little encryption,
- little trust of other machines.

and also:

- asynchronous checking (for e-mail),
- different cryptosystems,
- little server state,
- one-way or two-way authentication,
- client anonymity.

The Needham-Schroeder shared-key protocol



$\{X\}_K$: X encrypted with the shared key K

N_A, N_B : nonces

K_{AB} : a session key for A and B , for encrypting subsequent communications

A criticism

Long after a run, an attacker may

- discover K_{AB} ,
- replay $\{K_{AB}, A\}_{K_{BS}}$ to B ,
- conduct a handshake with B ,
- send arbitrary data to B under K_{AB} ,
impersonating A . (Denning & Sacco)

Solutions:

- make K_{AB} strong, change K_{BS} often
- let B and S interact (Needham & Schroeder),
- use timestamps (Denning & Sacco, Kerberos).

Kerberos

Initially, the Kerberos protocol was based on the Needham-Schroeder shared-key protocol plus timestamps. (And an early version did not even check the timestamps properly!)

Over time, it incorporated improvements and extensions.

For example:

- bootstrapping from passwords,
- cross-domain authentication (that is, several S 's).

Some observations

Most security protocols have subtleties and flaws.

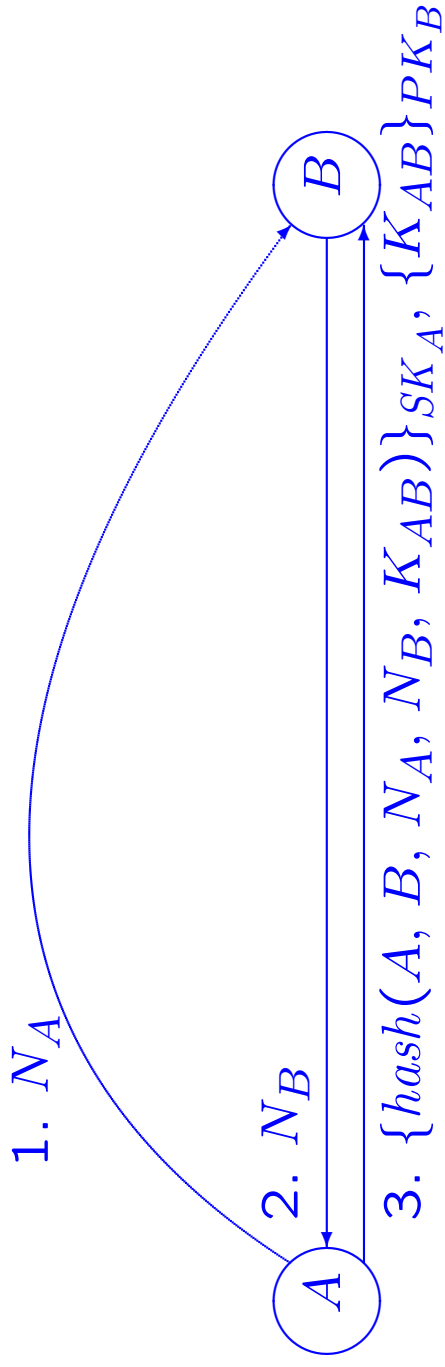
Many of these have to do with cryptography.

Many of these don't have to do with the details of cryptography.

For design, implementation, and analysis, a fairly abstract view of cryptography is often practical.

Netscape's SSL v3 protocol

(fragment, simplified)



3. $\{hash(A, B, N_A, N_B, K_{AB})\}_{SK_A}, \{K_{AB}\}_{PK_B}$

$\{X\}_{PK_B}$: X encrypted with B 's public key, for secrecy

$\{X\}_{SK_A}$: X signed with A 's secret key, for integrity

K_{AB}, N_A, N_B are used in computing a session key

The full SSL (or TLS)

A more complete and complex protocol with a bulkier specification:

- Specifics of cryptographic operations.
- Several options:
 - various cryptographic algorithms,
 - protocol versions,
 - one-way or two-way authentication,
 - a fast mode.
- Option negotiation.
- Message transmission (“the record layer”).
- Alert messages.

Design Principles for Protocols

(Mostly joint work with R. Needham.)

Principle

Every message should say what it means: the interpretation of the message should depend only on its content.

It should be possible to write down an English sentence describing the content—though if there is a suitable formalism available that is good too.

An attack

The core of the Denning-Sacco protocol is:

$$A \rightarrow B : \{\{K_{AB}, T\}_{SK_A}\}_{PK_B}$$

A tells B that K_{AB} is a good key for A and B at time T .

An attack goes:

$$A \rightarrow C : \{\{K_{AC}, T\}_{SK_A}\}_{PK_C}$$

$$C \rightarrow B : \{\{K_{AC}, T\}_{SK_A}\}_{PK_B}$$

so any C may have the key that B believes shared with A .

Then:

$$C \rightarrow B : \{\text{data}, T\}_{K_{AC}}$$

and B would believe that A sent data.

Diagnosis

Optimistic use of encryption.

Names are missing.

⇒ It is not possible to parse the message into the statement that represents its meaning.

Solution

$A \rightarrow B : \{\{A \text{ says } K_{AB} \dots B \dots T\}_{SK_A}\}_{PK_B}$

or

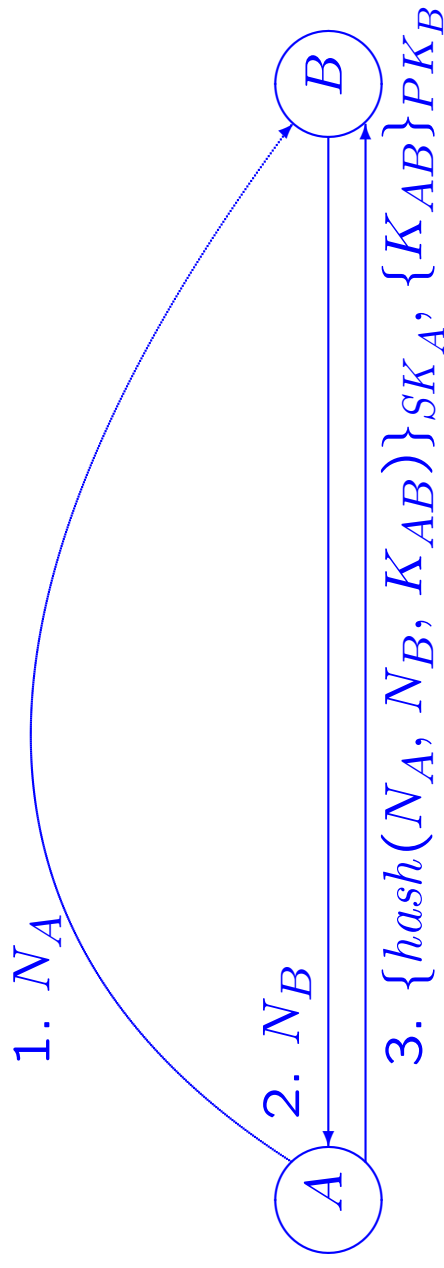
$A \rightarrow B : \{\{A, B, K_{AB}, T\}_{SK_A}\}_{PK_B}$

or

any other unambiguous encoding of the meaning of the message.

Netscape's SSL v3 protocol

Specification of Nov. 10, 1995 (simplified):



An analogous attack works.

Corollary

If the identity of a principal is important for the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

Other principles concern

- encryption
- timeliness
- trust
- secrecy

The principles serve to

- simplify protocols
- simplify formal analysis
(cf. Paulson's work with Isabelle)
- catch and avoid many mistakes

Staying out of trouble

Keep your protocols simple.

Be suspicious of clever optimizations.

Be explicit. Interpreting a message should be a simple matter of parsing.

Cryptography helps, but it is not the whole story.

We can and should go beyond patching security holes one by one.

The Applied Pi Calculus

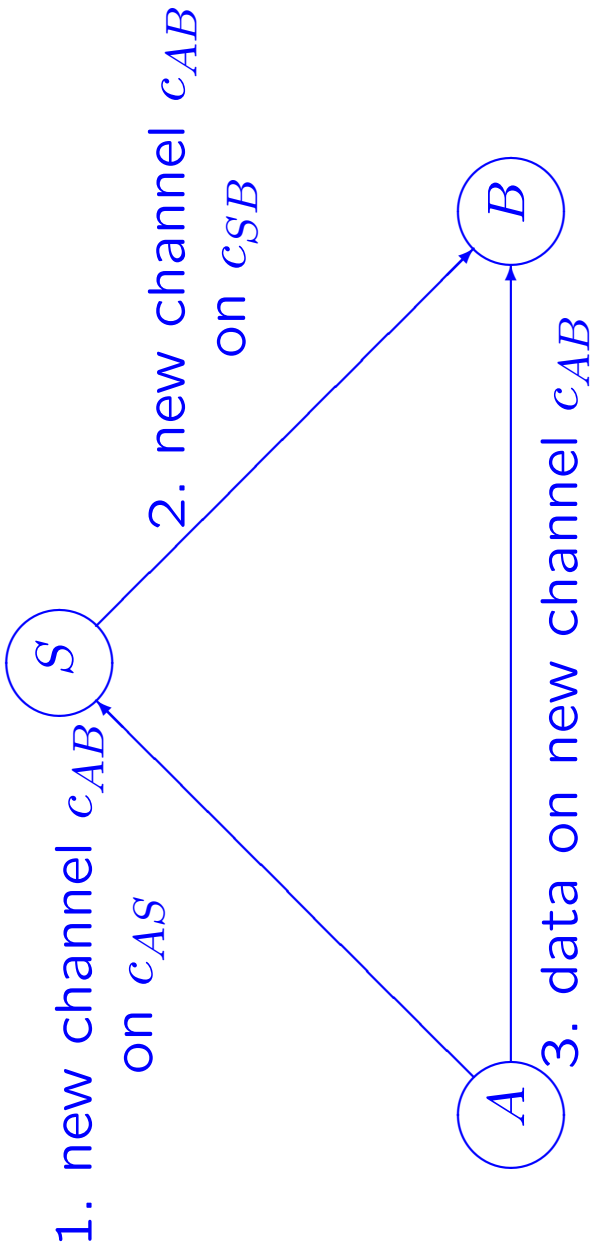
Design and analysis methods

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

- rigorous but informal frameworks,
- theorem-proving tools,
- special-purpose formalisms,
- temporal logics,
- **process calculi**.

A typical protocol

Establishment and use of a secure channel:

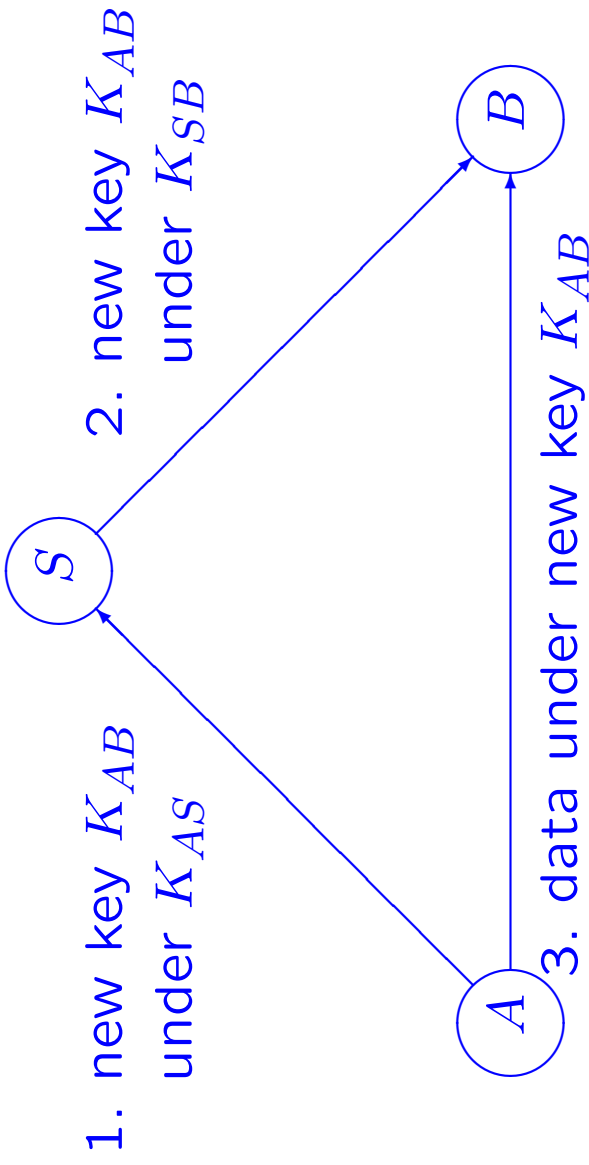


A and B : two clients S : an authentication server
 c_{AS} and c_{SB} : channels with the server
 c_{AB} : a new channel for the clients

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

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.

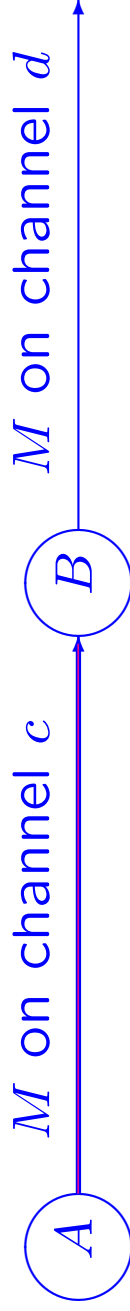
We can express some security properties, using tools from process calculi.

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.

Rephrasing authenticity

Authenticity can be rephrased by comparing P with a more magical protocol P' :

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

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

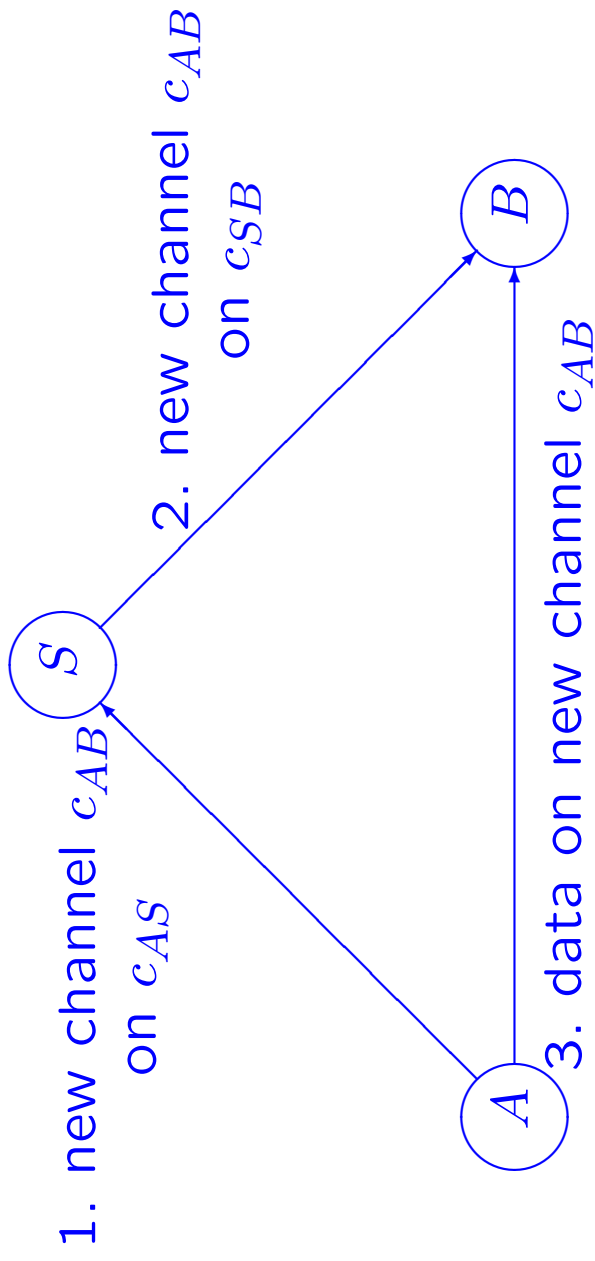
$$P'(M) \triangleq (\nu c)(A(M) \mid B'(M))$$

Here B' knows what A sent when it receives an input on c .

Authenticity can be “reduced” to the equivalence of $P(M)$ and $P'(M)$.

Again, almost any definition of equivalence will do.

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
$if\ M = N\ then\ P\ else\ Q$	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 enc and dec, with the equation:

$$\text{dec}(\text{enc}(x, y), y) = x$$

Dealing with errors

Types cannot statically avoid all “errors”.

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

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

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

We may then code derived constructs, such as:

$$\text{case } N \text{ of } \{x\}_M \text{ in } P \triangleq \begin{cases} N & \text{if } \text{dec}(N, M) = \text{wrong} \\ P[\text{dec}(N, M)/x] & \text{else} \end{cases}$$

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

Examples: public-key encryption

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

$$dec(enc(x, pk(y)), sk(y)) = x$$

Optionally, we may for example add:

$$dec(enc(x, y), z) = enc(dec(x, z), y)$$

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 closed by substitution of equal terms.

Operational semantics (cont.)

Reduction \rightarrow is the smallest relation on extended processes 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

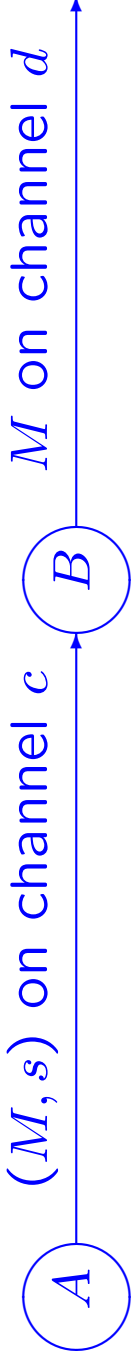
such that $\Sigma \not\vdash M = N$

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.

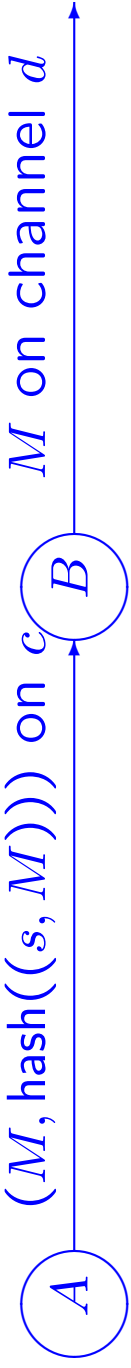
Interesting extrusions

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



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

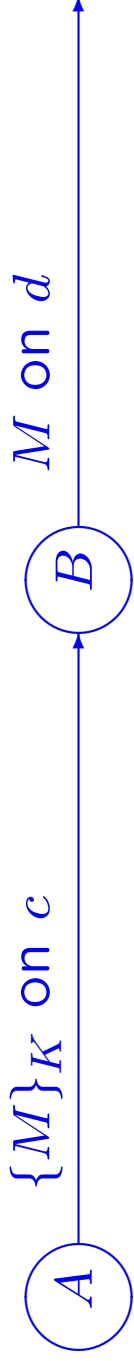
$$B \triangleq c(x).\text{if } \text{hash}((s, \text{fst}(x))) = \text{snd}(x) \text{ then } \bar{d}\langle\text{fst}(x)\rangle$$

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

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



$$A(M) \triangleq \bar{c}(\text{enc}(M, K))$$

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

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

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:

$$\begin{aligned} \text{Send}(i, j, M) &\triangleq \overline{c_S}\langle i \rangle \mid \\ &\quad c_i(x_{\text{nonce}}). \\ &\quad (\nu K)(\overline{c_S}\langle \text{enc}((i, i, j, K, x_{\text{nonce}}), K_{iS}) \rangle \mid \\ &\quad \overline{c_j}\langle (i, \text{enc}(M, K)) \rangle) \end{aligned}$$

$$\text{Recv}(j) \triangleq \dots \overline{d}\langle y \rangle$$

$$S \triangleq \dots$$

$$\begin{aligned} \text{Sys}(I_1, \dots, I_m) &\triangleq (\nu K_{1S}) \dots (\nu K_{nS}) \\ &\quad (\nu K_{S1}) \dots (\nu K_{Sn}) \\ &\quad (\text{Send}(I_1) \mid \dots \mid \text{Send}(I_m) \mid \\ &\quad !S \mid \\ &\quad !\text{Recv}(1) \mid \dots \mid !\text{Recv}(n)) \end{aligned}$$

Again:

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

The notion of equivalence becomes delicate:

$(\nu K)\bar{c}\langle \text{enc}(M, K) \rangle$ and $(\nu K)\bar{c}\langle \text{enc}(N, K) \rangle$ 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.

The environment

We define, implicitly, a simple but compelling model of the environment.

The environment is an arbitrary program:

- it can try to create confusion through concurrent sessions,
- it can initiate sessions,
- it can replay messages,
...
- it can make up random numbers,
- but it cannot get too lucky with random numbers, because of scoping.

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, proof techniques.

Type systems.

Decision procedures for some problems.

Automated analysis via logic programs.

Partial complexity-theoretic foundations.

Examples.

Logics? Models?