Computer Security

Martín Abadi

University of California
at Santa Cruz

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.

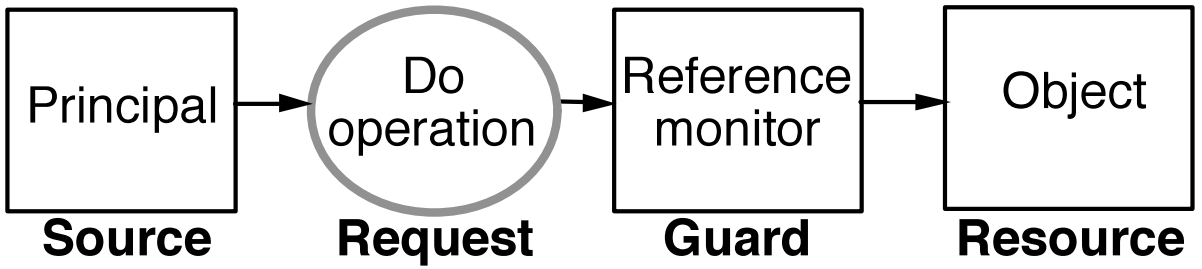
- Several trends conspire to make systems more vulnerable to deliberate attacks.

Computer security

- Much like other sorts of security:
  - not black-and-white,
  - not all about defenses,
  - often inconvenient.
- With some distinctive ingredients (e.g., attacks at a distance).
- Concerned with
  - secrecy,
  - integrity,
  - availability.

The access-control model

- Elements:
  - Objects or resources.
  - Requests.
  - Sources for requests, called principals.
  - A reference monitor to decide on requests.



Authentication vs. access control

- Access control (authorization):
  - Is principal A trusted on statement S?
  - If A requests S, is S granted?

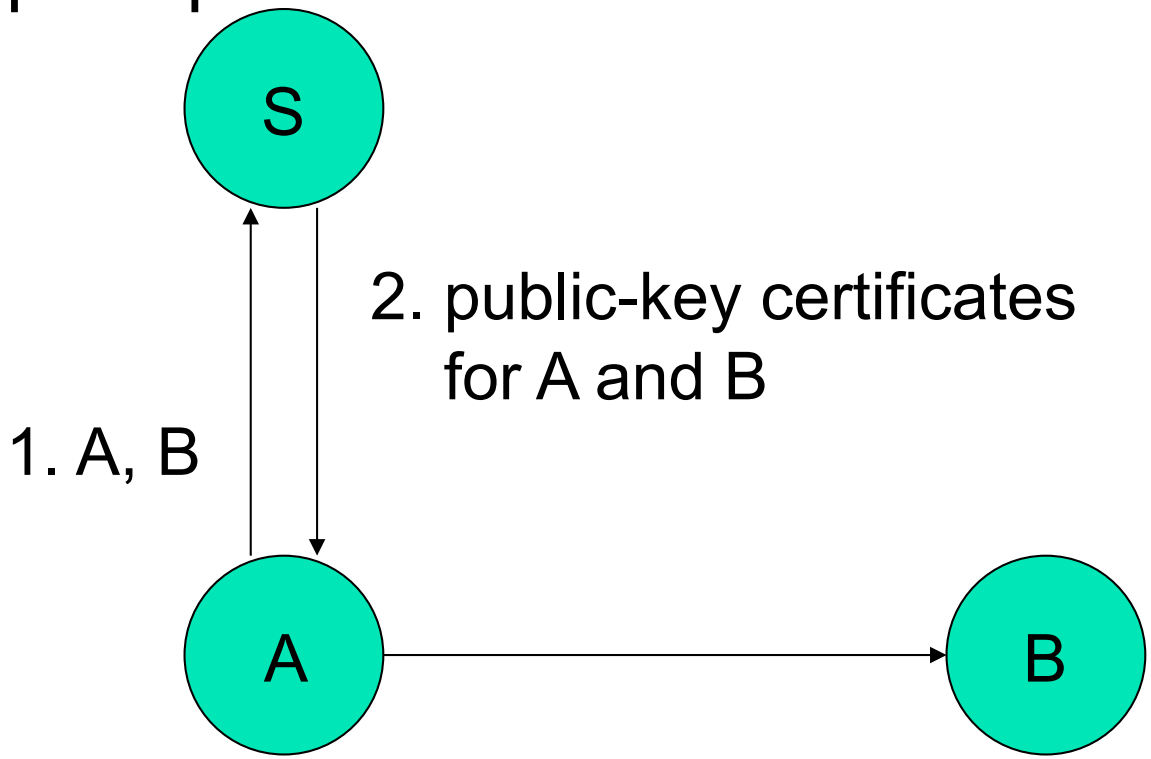
- Authentication:
  - Who says S?

Plan

- Protocols for authentication (and, more generally, for secure communication)
  - How they go wrong, and why
  - Formal analysis
  - Another example: fighting spam
- Access control
  - Logics and languages
  - A bit on the relation to data integration
- Speculation

Authentication protocols (and other security protocols)

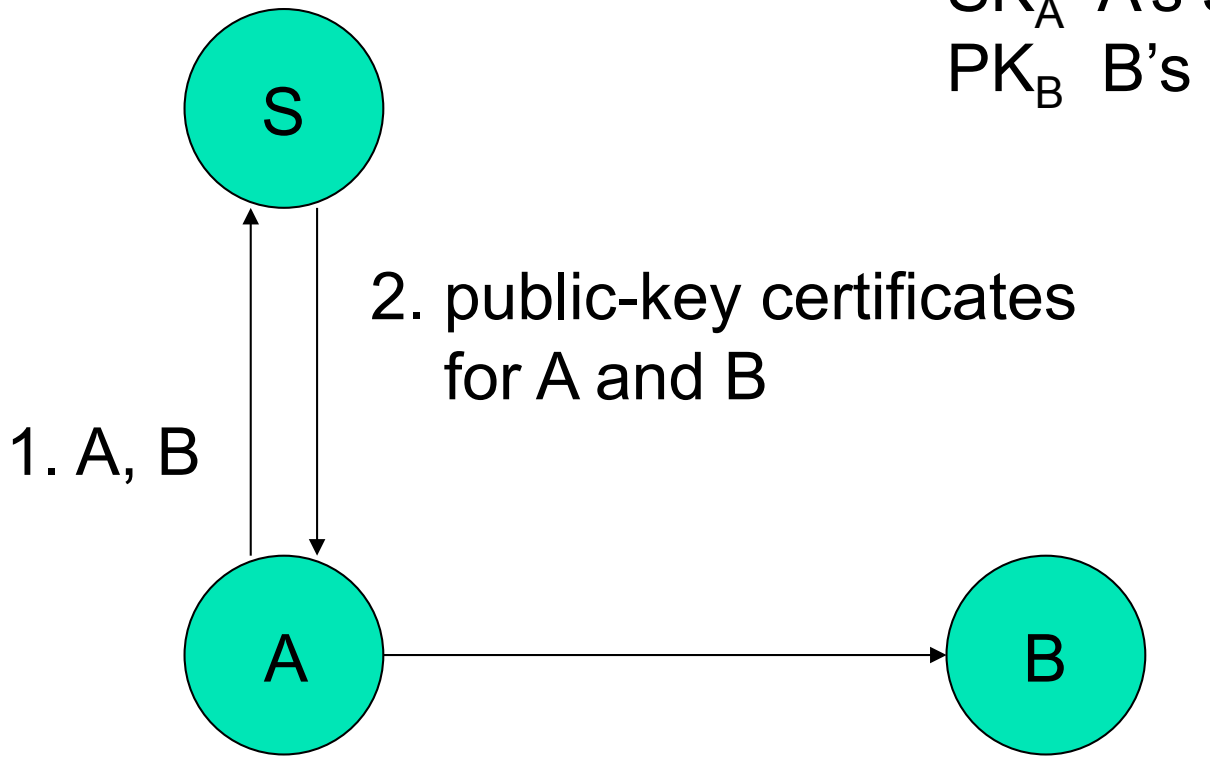
- A typical protocol:



3. new session key, signed, encrypted under B's public key (plus certificates)
4. data, encrypted under the session key

A closer look: the Denning-Sacco public-key protocol

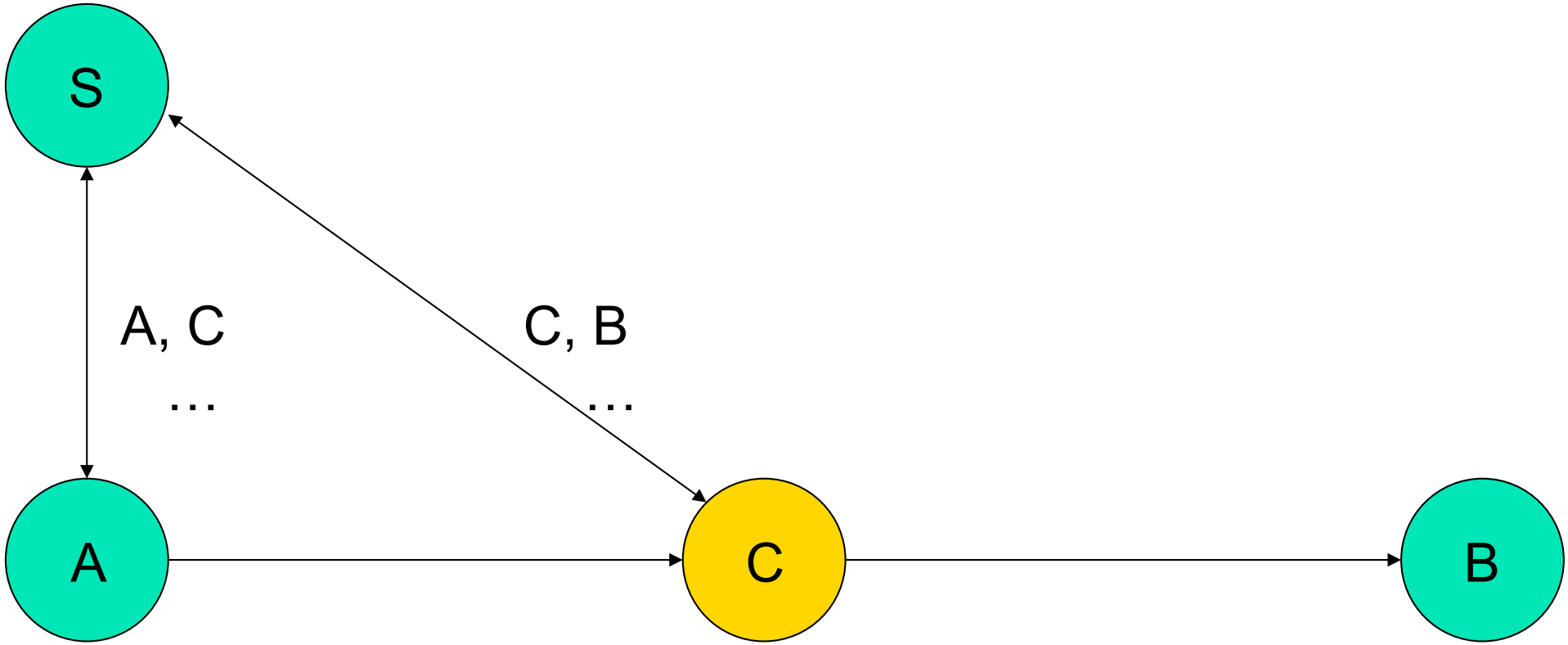
SK_A A's secret key
PK_B B's public key



Questions

- Does the protocol work?
- What assumptions are we making?
- Can we do better?

An attack
(joint work with R. Needham)



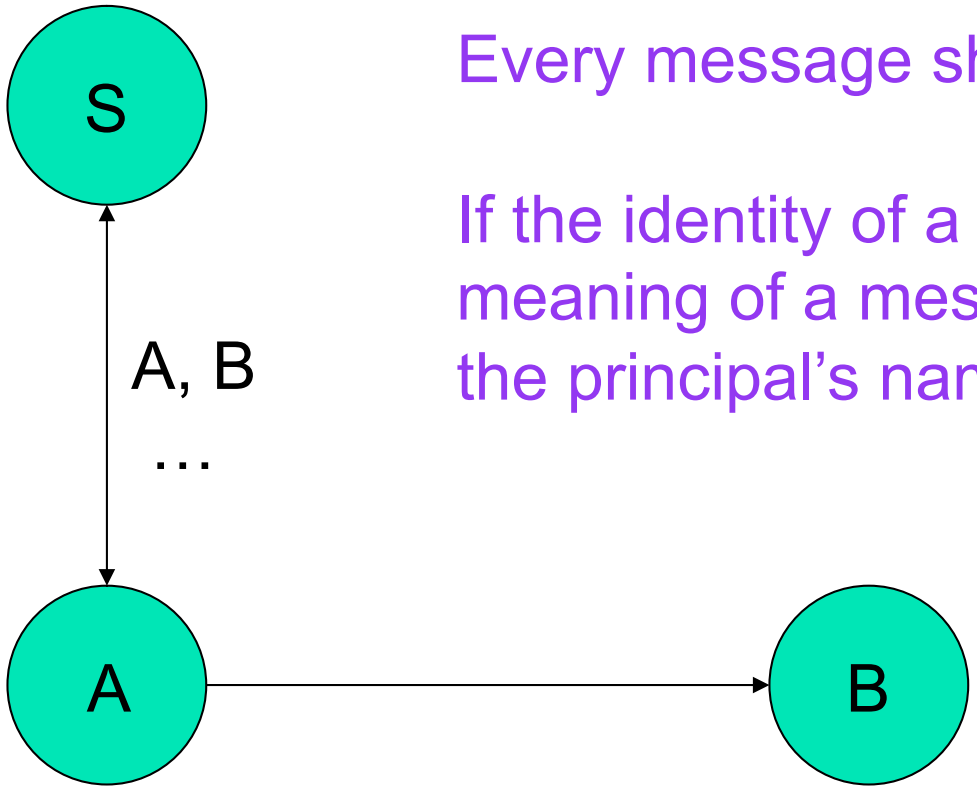
3. enc(sign((K, T), SK_A), PK_C)

4. enc(data, K)

3. enc(sign((K, T), SK_A), PK_B)

4. enc(data', K)

Correction



Every message should say what it means.

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.

3. enc(sign((K, A, B, T), SK_A), PK_B)

4. enc(data, K)

Analyzing security protocols

- The design of security protocols is notoriously error-prone.
- Many subtleties and flaws have to do with cryptography, but not with its details.
- A variety of methods have been developed to guide their design and to help in their analysis:
  - informal approaches based on complexity theory,
  - human-guided theorem proving,
  - finite-state model checking,
  - static analyses.

A language for protocols

- The language should include concurrency and communication through message passing.
- It may also include a black-box (symbolic) model of cryptography.

The pi calculus
(Milner, Parrow, and Walker)

- The pi calculus is a general, simple language for concurrent processes that communicate by sending messages on named channels.
- It includes a "new" construct for generating fresh names (fresh channels, fresh keys, …).

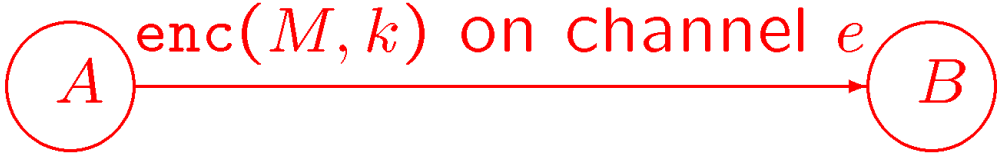
$$(\texttt{new}\ n)((\texttt{snd}\ M\ \texttt{on}\ n) \mid (\texttt{rcv}\ x\ \texttt{on}\ n\ \texttt{in}\ \ldots))$$

  - Here two processes run in parallel.
  - One sends the term *M* to the other on a fresh, private channel *n*.
  - *x* is a bound variable.

Extending the pi calculus

(joint work with A. Gordon, C. Fournet, B. Blanchet)

- What about



- All we need to do is add function symbols.

$$(\mathtt{new}\ k)((\mathtt{snd}\ \mathtt{enc}(M, k)\ \mathtt{on}\ e) \mid (\mathtt{rcv}\ x\ \mathtt{on}\ e\ \mathtt{in} \ldots \mathtt{dec}(x, k) \ldots))$$

  - Here “new” generates a key.
  - Encryption and decryption are function symbols, with equations.

$$\mathtt{dec}(\mathtt{enc}(M, k), k) = M$$

A syntax

$M, N ::=$	terms
x	variable
n	name
$f(M_1, \ldots, M_n)$	constructor application
$P, Q ::=$	processes
$\overline{M}\langle N \rangle.P$	output
$M(x).P$	input
0	nil
$P \mid Q$	parallel composition
$!P$	replication
$(\nu n)P$	"new"
$let\ x = g(M_1, \ldots, M_n)\ in$ $P\ else\ Q$	destructor application

Secrecy by typing

- Develop principles for secrecy.
  - Do not send secrets on public channels.
  - Secrets, suitably encrypted, can be made public.
- Formalize them as typing rules.
- Check adherence by typechecking.
- "Well-typed processes preserve their secrets."

Theorem:

If $E, c : \mathsf{Public}, s : \mathsf{Secret} \vdash P$,
and no secret names occur free in Q,
then $P \mid Q$ never outputs s on c.

Spam

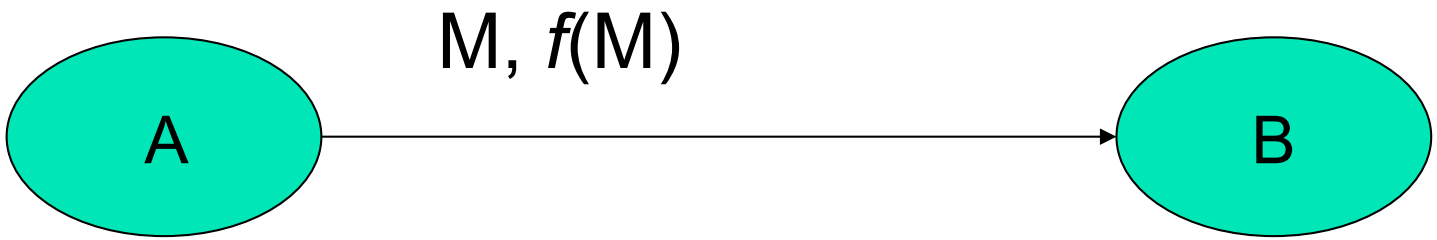
- Spam is unsolicited, junk e-mail.
  - It is hard to define exactly and universally.
  - Most of the mail at Hotmail and Yahoo! is spam.
- Deterring spam means:
  - Discourage or eliminate unwanted e-mail.
  - Allow public e-mail addresses.
  - Allow flow of legitimate e-mail.
  - Minimize inconvenience to legitimate senders.
- Non-technical measures may not suffice.
- There are related abuses, to which techniques against spam may be applicable.

Proposed techniques

- E-mail filtering
- Source-address confirmation
- Ephemeral e-mail addresses
- Requiring payment, somehow

Pricing via processing
(Dwork and Naor, and later Back)

- Pick a moderately hard function *f*.
  - 10 seconds or 10 minutes per application?
  - No amortization across applications.
- When A sends message M to B at time t, A tags M with *f*(M).



- The tag should be quick for B to verify.

Pricing via processing: a hurdle

(joint work with M. Burrows, M. Manasse, T. Wobber)

- To date, this work has relied on CPU-intensive computations.
  - E.g., breaking low-security Fiat-Shamir signatures.
  - E.g., finding hash collisions.
- Machines vary widely in CPU power.
  - 33MHz PDA vs. 2.5GHz PC.

- 25 billion CPU cycles may not discourage a spammer with high-end PCs or servers
  (who will compute 10 seconds, at leisure).
- 25 billion CPU cycles are a serious obstacle for users with low-end machines
  (who will wait minutes? hire help? give up?).

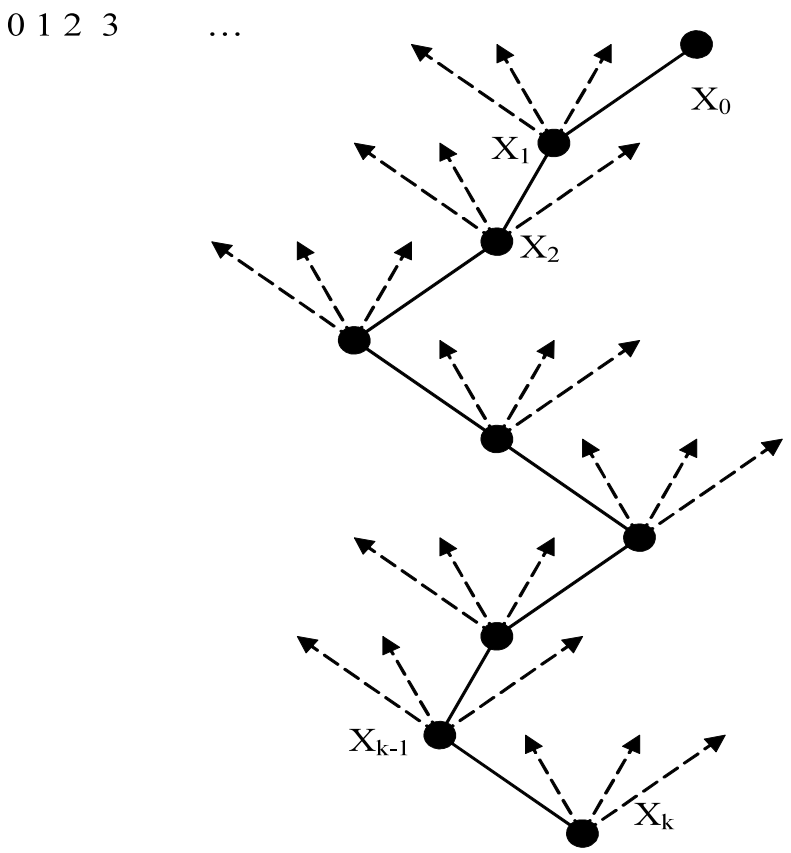
An idea: memory-bound functions

- Memory-system latency is fairly uniform across a wide range of machines (within a factor of 2-4 for machines built in the last 5 years).
- So we would like a family of problems that:
  - take many cache misses to solve (say, 2^{23}),
  - are fast to set/check (1,000 or 10,000 times faster),
  - can be expressed and answered concisely (in at most a few kilobytes),
  - can be made harder by changing a parameter, and
  - where solving one problem does not help in solving another one.

Approach

- For each e-mail, recipient B requires sender A to invert a well-chosen function *g* many times.
- A should build a large table and access it a lot.
  - Inverting *g* otherwise may be slow (100s of cycles or much more, each time).
  - The table should fit in a small memory (32MB) but not in a large cache (8MB).
  - Caches should not help much.
- B can easily check A's work by applying *g*.

An inversion tree



x_k a challenge (from B or dependent on M)

x_0 the expected response from A

$x_{i+1} = g(x_i)$ xor i

For a random function *g*:

Tree size quadratic in k

Number of leaves = k + 1

Machines for experiments

	model	processor
server	Dell PowerEdge 2650	Intel Pentium 4
desktop	Compaq DeskPro EN	Intel Pentium 3
laptop	Sony PictureBook C1VN	Transmeta Crusoe
settop	GCT AllWell STB3036N	NS Geode GX1
PDA	Sharp SL-5500	Intel SA-1110

Machine characteristics

	CPU clock	Memory read
server	2.4GHz	0.19μs
desktop	1GHz	0.14μs
laptop	600MHz	0.25μs
settop	233MHz	0.23μs
PDA	206MHz	0.59μs

Timings

	HashCash		Trees	
	times	ratios	times	ratios
server	110s	1.0	24s	1.1
desktop	140s	1.3	22s	1.0
laptop	330s	3.0	42s	1.9
settop	1430s	13.0	91s	4.1
PDA	1920s	17.5	100s	4.5

Access control

- Access control is pervasive
  - applications
  - virtual machines
  - operating systems
  - firewalls
  - doors
  - …
- Access control seems difficult to get right.
- Distributed systems make it harder.

General theories and systems

- Over the years, there have been many theories and systems for access control.
  - Logics
  - Languages
  - Infrastructures (e.g., PKIs)
  - Architectures
- They often aim to explain, organize, and unify access control.
- They may be intellectually pleasing.
- They may actually help.

An approach

- A notation for representing principals and their statements, and perhaps more:
  - objects and operations,
  - trust,
  - channels,
  - …
- Derivation rules

Early ideas

(Excerpts of a message from Roger Needham, Aug. 1987)

- Notations

  P$|= S Principals in the set P$ are the guarantors for S

  C ||- S Channel C actually asserts S

  C -> P Channel C authenticates Principal P

- Postulates

  C ||- S, C -> P

  P |= S

A calculus for access control

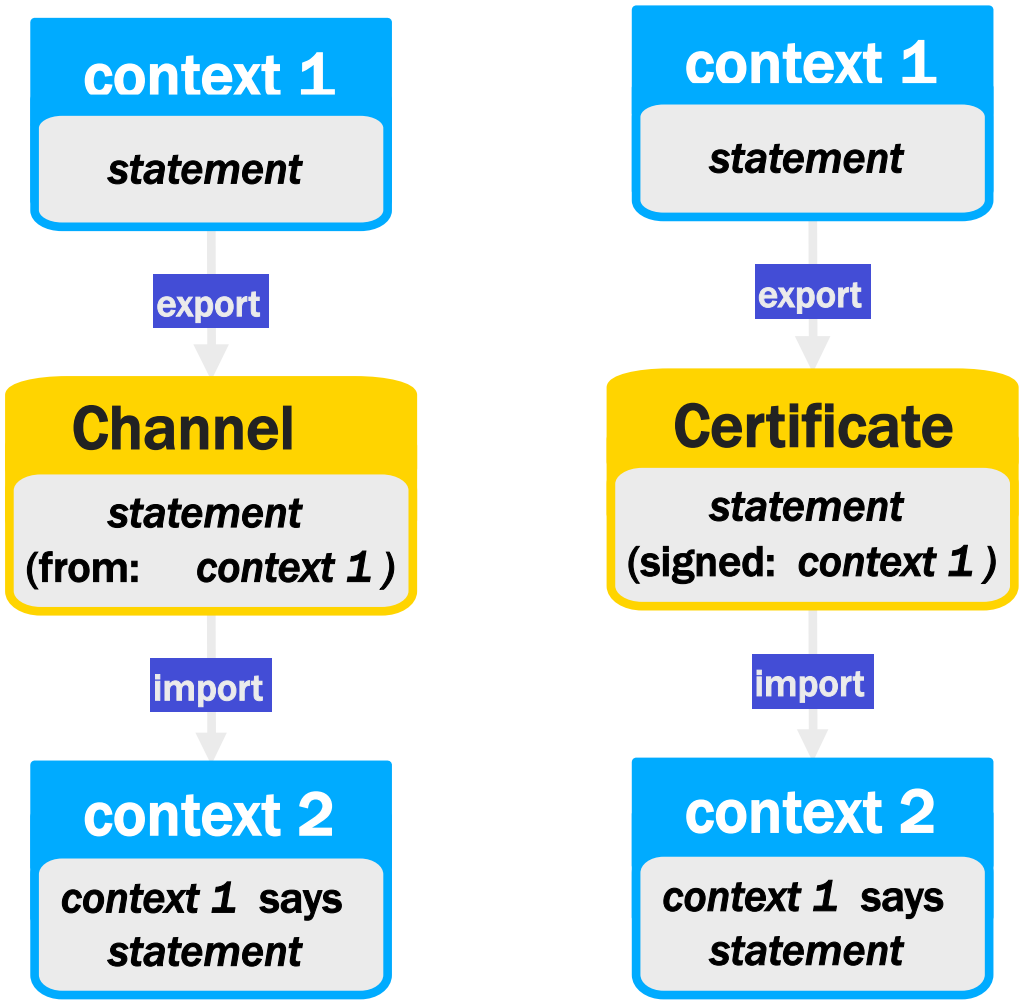
(joint work with M. Burrows, B. Lampson, and G. Plotkin)

- A simple notation for assertions
  - A says S
  - A speaks for B
- With compound principals
  - $A \wedge B$
  - A for B
- With logical rules
  - A says $(S \Rightarrow T) \Rightarrow$ (A says S) $\Rightarrow$ (A says T)
  - A speaks for B $\Rightarrow$ (A says S) $\Rightarrow$ (B says S)

Says

Says represents communication across contexts.

Says abstracts from the details of authentication.



Other languages and systems

- PolicyMaker and KeyNote [Blaze et al.]
- SDSI [Lampson and Rivest]
- SPKI [Ellison et al.]
- D1LP and RT [Li et al.]
- SD3 [Jim]
- Binder [DeTreville]
- XrML 2.0
- …

- Several of the most recent are based on ideas and techniques from logic programming.

Binder

- Binder is a relative of Prolog.
- Like Datalog, it lacks function symbols.
- It also includes the special construct says.
- It does not include much else.
- Binder programs can define and use new, application-specific predicates.
- Queries in Binder are decidable (in PTime).

An example in Binder

- Facts
  - owns(Alice, “foo.txt”).
  - Alice says good(Bob).
- Rules
  - may_access(p,o) :- owns(q,o), blesses(q,p).
  - blesses(Alice,p) :- Alice says good(p).
- Conclusions
  - may_access(Bob,“foo.txt”).

Binder's proof rules

- Suppose F has the rules
  - may_access(p,o) :- owns(q,o), blesses(q,p).
  - blesses(Alice,p) :- Alice says good(p).
- Some context D may import them as:
  - F says may_access(p,o) :-
    F says owns(q,o), F says blesses(q,p).
  - F says blesses(Alice,p) :- Alice says good(p).
- D and F should agree on Alice's identity.
- The meaning of predicates may vary.
  - good(p) :- Alice says good(p).
  - good(p) :- Bob says excellent(p).

Data integration

- A classic database problem is how to integrate multiple sources of data.
  - The sources may be heterogeneous. Their contents and structure may be partly unknown.
  - The data may be semi-structured (e.g., XML on the Web).

TSIMMIS and MSL

(Garcia-Molina et al.)

- Wrappers translate between a common language and the native languages of sources.
- Mediators then give integrated views of data from multiple sources.
- The mediators may be written in the Mediator Specification Language (MSL).

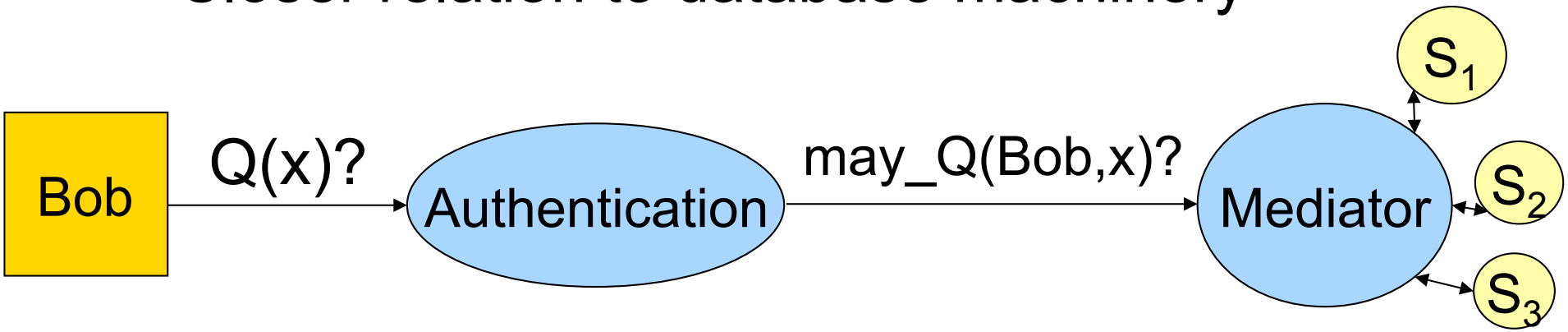
```
<cs_person {<name N> <relation R> Rest1 Rest2}>@med :-
    <person {<name N> <dept `CS'> <relation R> | Rest1}>@whois
    AND decompose_name(N, LN, FN)
    AND <R {<first_name FN> <last_name LN> | Rest2}>@cs
```

Similarities and caveats

- MSL is remarkably similar to Binder.
  - They start from Datalog.
  - They add sites (or contexts).
  - X@s corresponds to s says X.
  - In X@s, the site s may be a variable.
- MSL and Binder are used in different environments and for different purposes.
  - Work in databases seems to focus on a messy but benign and tolerant world, full of opportunities.
  - Work in security deals with a hostile world and tries to do less.

Potential outcomes (speculation)

- Language-design ideas
  - Constructs beyond Datalog
  - Semi-structured data
- More theory, algorithms, tools
- Closer relation to database machinery



may_Q(p,x) :- s_1 says Q(x),
s_2 says Ok(p,x)

Conclusions and open issues

- Big stakes—apparently getting bigger
- A growing body of sophisticated techniques
- Some art, some science

- Elaborate machinery, but
  not always easy to explain or to apply reliably
- Substantial, credible efforts, but
  without full proofs or good metrics of security