

Introduction to Computational Soundness



Martín Abadi

Slides from M. Abadi presented by

Anupam Datta

CMU

Fall 2011

Security and simplification

Security is hard to define precisely.

“Security is fractal.” (Butler Lampson)

Whatever our model, some attacker will probably ensure that it is incomplete or flawed.

Simplistic models are perhaps inevitable and dangerous, but they can be useful as

- metaphors,
- heuristics,
- abstractions.

Nevertheless, it is both satisfying and useful to justify simplifications when possible.

This lecture

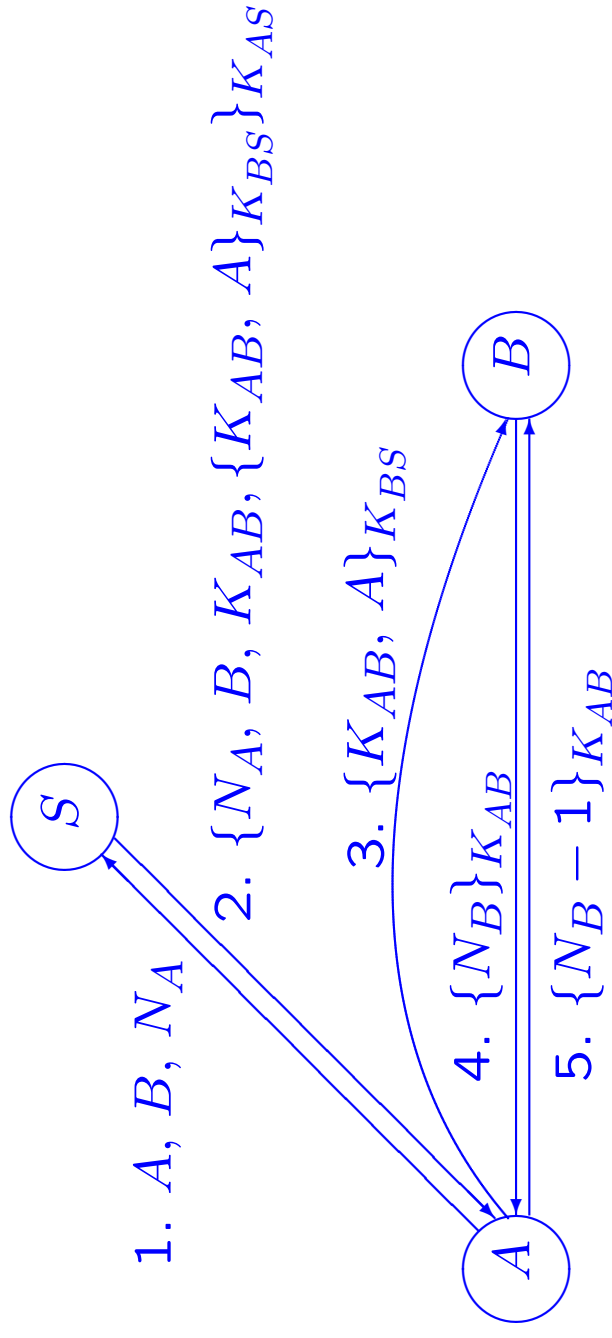
Some background and motivation

A first computational justification of formal cryptography

Some variants and extensions (briefly)

Background and motivation

The Needham-Schroeder shared-key protocol



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

using a symmetric (shared-key) cryptosystem

N_A, N_B : nonces

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

for encrypting subsequent communications

A typical problem

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)

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.

The origin of formal cryptology

An attacker

- knows some data in advance,
- may intercept messages on the public network,
- may inject messages into the public network

What messages can the attacker produce?

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

A simple solution:

- Key generation always yields a fresh key.
- Keys are not bitstrings.
- Cryptographic operations are treated symbolically.

The formal view: strengths

Some simple, effective intuitions.

Easy human reasoning.

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

Techniques and tools for automated reasoning:

- Decision procedures (since Dolev & Yao).
- Model-checkers (FDR, Mur ϕ , ...).
- Theorem provers (Coq, Isabelle, ProVerif, ...).

The formal view: blind spots

Relevance to instantiations of protocols, where the cryptographic functions are not black boxes.

Details of assumptions, particularly on cryptography.

Probabilities and complexities.

Attacks that are assumed impossible, rather than actually proved impossible (e.g., guessing keys).

Quantitative design and analysis.

Some early concerns

What does formal analysis prove, if anything?

Even more dangerous to the protocol designer is the possibility that cryptographic operations may satisfy simple algebraic identities of which he is unaware [...]

Such properties might allow an attacker to make more powerful inferences than the model reflects, defeating some protocols. However, once such identities are discovered, new algebras representing the cryptosystem can be formulated to represent these more powerful inferences.

Michael Merritt, *Cryptographic Protocols*, 1983

However, formal analysis was often more effective and useful than the alternatives.

The computational view

Keys and messages are bitstrings,
not formal expressions.

A good protocol is one that resists computationally
reasonable attacks with high probability.

This computational view leads to another rigorous
approach for reasoning about protocols.

Comparison

The computational approach is

- more complete,
- foundationally more satisfying,
- closer to Turing machines,
- further from general methods for reasoning about reactive systems,
- harder to apply,
- sometimes overkill,
- harder to link to naive intuitions.

Towards a convergence (mid 1990s)

Computational cryptography became more relevant to the protocols that were the subject of formal analysis.

- See in particular the papers by Bellare and Rogaway on authentication and key distribution (1993–1998).

There were new efforts and successes in using formal methods for verification (not simply for finding bugs).

- With increasingly sophisticated tool support.
- Even for infinite-state systems.

The field as a whole grew.

Towards a convergence (cont.)

Concerted efforts to relate formal and computational models probably started around 1996 or 1997.

It was immediately clear that there is room for diverse approaches, and that the problem is much bigger than what a single paper could treat.

This workshop goes well beyond what we all envisioned at the start.

Computational soundness:
the approach and a result

Computational soundness

(starting in joint work with Phil Rogaway)

Soundness property (desired):

If a security property can be proved formally,
then it holds in the computational model.

The formal proof will

- not mention probabilities and complexities,
- consider attacks only in the formal model,
- establish an all-or-nothing statement.

Soundness means that the statement is true for all computationally reasonable attacks with high probability.

A simple case

Suppose that

(new K_1, \dots, K_n) send $M \approx$ (new K_1, \dots, K_n) send N

in a process calculus (e.g., the spi calculus), where \approx is observational equivalence.

Can we justify this computationally?

For example, are M and N indistinguishable in polynomial time, provided that K_1, \dots, K_n are generated at random?

A term language

The set of expressions **Exp**:

$M, N ::=$	expressions
i	bits (for $i \in \mathbf{Bool}$)
K	keys (for $K \in \mathbf{Keys}$)
(M, N)	pairs
$\{M\}_K$	symmetric encryptions

Patterns

We map each expression M to a *pattern* that the attacker can see with no a priori knowledge:

$$\begin{aligned} \text{pattern}(i) &= i && \text{(for } i \in \mathbf{Bool}\text{)} \\ \text{pattern}(K) &= K && \text{(for } K \in \mathbf{Keys}\text{)} \\ \text{pattern}((N_1, N_2)) &= (\text{pattern}(N_1), \text{pattern}(N_2)) \\ \text{pattern}(\{N\}_K) &= \begin{cases} \{\text{pattern}(N)\}_K & \text{if } M \vdash K \\ \square & \text{otherwise} \end{cases} \end{aligned}$$

where

- $M \vdash M$.
- If $M \vdash (N_1, N_2)$ then $M \vdash N_1$ and $M \vdash N_2$.
- If $M \vdash \{N\}_K$ and $M \vdash K$ then $M \vdash N$.

Example: $\text{pattern}(\{\{0\}_{K_1}\}_{K_2, K_2}) = (\{\square\}_{K_2}, K_2)$

Equivalence (\cong)

Informally, two expressions are equivalent if they look the same to an attacker.

Formally, two expressions are *equivalent* if they yield the same pattern (up to renaming).

Examples:

$$0 \cong 0$$

$$0 \not\cong 1$$

$$\{0\}_K \cong \{1\}_K$$

$$(K, \{0\}_K) \not\cong (K, \{1\}_K)$$

$$(K, \{\{0\}_{K'}\}_K) \cong (K, \{\{1\}_{K'}\}_K)$$

$$(\{0\}_K, \{0\}_K) \cong (\{0\}_K, \{1\}_K)$$

Finer points

Key equalities are concealed:

$$(\{0\}_K, \{1\}_K) \cong (\{0\}_K, \{1\}_{K'})$$

This is not standard in computational cryptography, but can be accommodated.

Finer points (cont.)

Ciphertexts do not reveal the size of plaintexts:

$$\{0\}_K \cong \{((1, 1), (1, 1)), ((1, 1), (1, 1))\}_K$$

Encryption cycles do not leak data:

$$\{0\}_K \cong \{K\}_K$$

Both of these are problematic computationally.

Finer points (cont.)

Such discrepancies are not all bad.

In some cases they correspond to legitimate variations in definitions.

Research on computational soundness encouraged more research in cryptography, on key-dependent encryption.

The computational view

An encryption scheme consists of algorithms:

$$\mathcal{K} : \text{Parameter} \times \text{Coins} \rightarrow \text{Key}$$

$$\mathcal{E} : \text{Key} \times \text{String} \times \text{Coins} \rightarrow \text{Ciphertext}$$

$$\mathcal{D} : \text{Key} \times \text{String} \rightarrow \text{Plaintext}$$

where $\text{Parameter} = 1^*$ (numbers in unary),
 Key , Plaintext , $\text{Ciphertext} \subseteq \text{String}$.

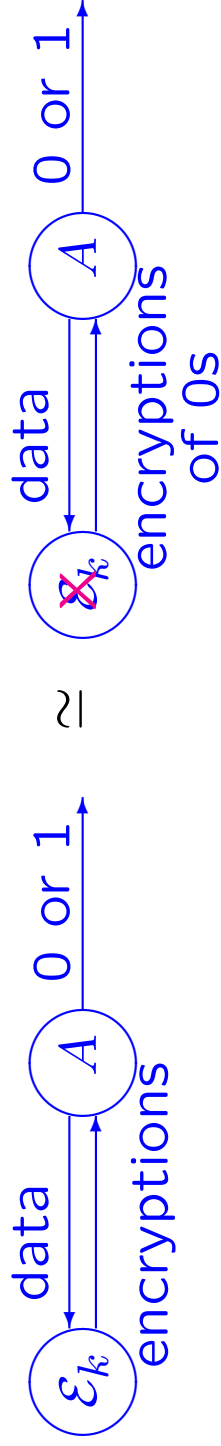
For all $\eta \in \text{Parameter}$, $k \in \mathcal{K}(\eta)$, and $r \in \text{Coins}$,

- if $m \in \text{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m, r)) = m$,
- if $m \notin \text{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m, r)) = \mathbf{0}$

where $\mathbf{0} \in \text{Plaintext}$ (a fixed string).

Secure encryption

In a standard definition, an encryption scheme is secure if it is hard to distinguish a real encryption oracle from a fake one.



Encryption scheme Π is **secure** if, for every probabilistic polynomial-time adversary A ,

$$\text{Adv}_{\Pi[\eta]}(A) \triangleq \Pr \left[k \stackrel{R}{\leftarrow} \mathcal{K}(\eta) : A^{\mathcal{E}_k(\cdot)}(\eta) = 1 \right] - \Pr \left[k \stackrel{R}{\leftarrow} \mathcal{K}(\eta) : A^{\mathcal{E}_k(0^{\cdot})}(\eta) = 1 \right]$$

is negligible (that is, “very small”).

Semantics

We map:

$M \in \mathbf{Exp}$
 $\eta \in \mathbf{Parameter}$
 Π an encryption scheme

\mapsto

a distribution on
bitstrings $[[M]]_{\Pi[\eta]}$

and thereby

$M \in \mathbf{Exp} \mapsto$ an ensemble $[[M]]_{\Pi}$

(An *ensemble* is a collection of distributions on strings, one for each value of the security parameter.)

Semantics (cont.)

First, we map each key symbol K that occurs in M to a bitstring $\tau(K)$, using $\mathcal{K}(\eta)$.

Then we set (roughly):

$$\llbracket 0 \rrbracket_{\Pi[\eta]} = 0$$

$$\llbracket 1 \rrbracket_{\Pi[\eta]} = 1$$

$$\llbracket K \rrbracket_{\Pi[\eta]} = \tau(K)$$

$$\llbracket (M, N) \rrbracket_{\Pi[\eta]} = (\llbracket M \rrbracket_{\Pi[\eta]}, \llbracket N \rrbracket_{\Pi[\eta]})$$

$$\llbracket \{M\}_K \rrbracket_{\Pi[\eta]} = \mathcal{E}_{\tau(K)}(\llbracket M \rrbracket_{\Pi[\eta]})$$

We assume that lengths depend only on structure.

Indistinguishable probability ensembles

D and D' are *indistinguishable* ($D \approx D'$) if,

for every polynomial-time adversary A ,

$$\epsilon(\eta) \triangleq \Pr[x \stackrel{R}{\leftarrow} D_\eta : A(\eta, x) = 1] - \Pr[x \stackrel{R}{\leftarrow} D'_\eta : A(\eta, x) = 1]$$

is negligible.



A computational soundness theorem

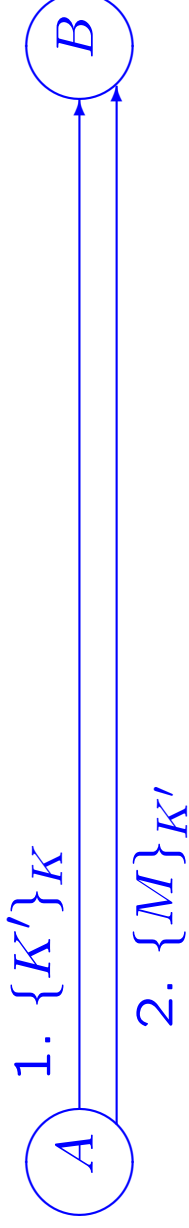
Let M and N be expressions without encryption cycles, and Π be a secure encryption scheme.

If $M \cong N$ then $\llbracket M \rrbracket_{\Pi} \approx \llbracket N \rrbracket_{\Pi}$.

There are many variants of this result.

Converses also hold (see Micciancio and Warinschi).

Applications: A first, small example



- A and B share a long-term symmetric key K .
- A creates a new symmetric key K' .
- A sends K' to B encrypted under K .
- A and B can communicate under K' .

Formally, the transcript $(\{K'\}_K, \{M\}_{K'})$ is equivalent to any variant with a different payload $(\{K'\}_K, \{N\}_{K'})$.

A computational guarantee follows.

Interpretation / Benefits

Formal reasoning is sound.

If there is an attack in the computational model, then there is also an attack in the formal model.

We obtain simple intuitions and a proof method for the computational model.

Further developments (cont.)

Active attackers

Soundness results for particular proof methods:

- tools such as Casrul and ProVerif
- some type systems

Hybrid models

Direct computational proofs

Outlook

It is now clear that formal proofs (even automated ones) can be interpreted computationally.

The work is not finished.

Computational proofs are progressing too.

