The Design of Cryptosystems: The Interplay between Proofs and Attacks

French-German-Singaporean Workshop on Applied Cryptography

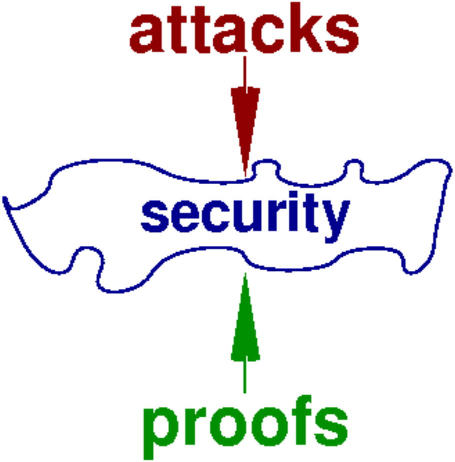
3rd of December, 2010

Stefan Lucks

Bauhaus-Universität Weimar, Germany

Bauhaus-Universität Weimar

Roadmap


attacks
security
proofs

Example: Entity Recognition

The Problem with Proofs

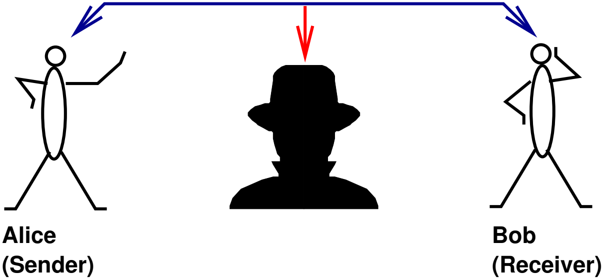
The Jane Doe Protocol

The Random Oracle Debate

The Jane Doe Protocol (revisited)

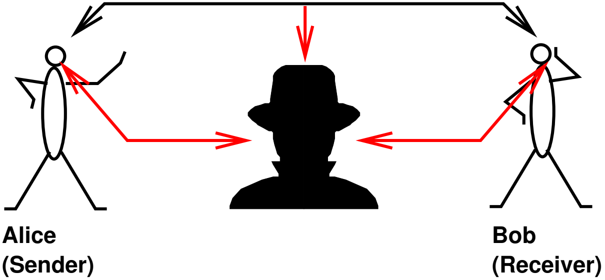
Summary

Example: Entity Recognition



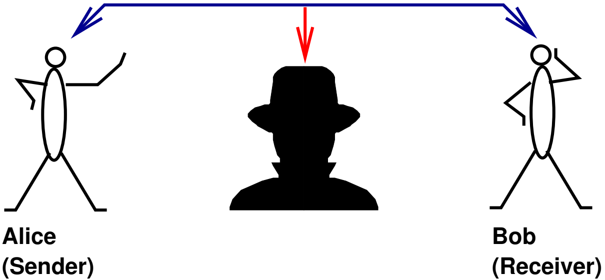
- Alice ("Jane Doe") and Bob meet at a party.
  They make a bet. Others listen.

Example: Entity Recognition



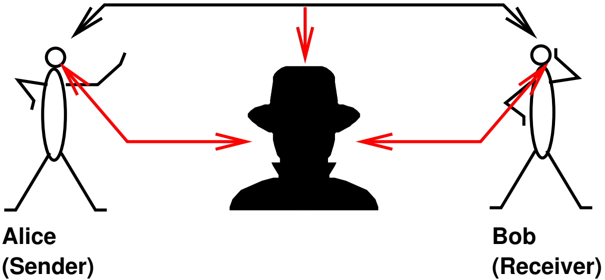
- Alice ("Jane Doe") and Bob meet at a party.
  They make a bet. Others listen.
- Much later, when it had turned out that Alice won, Bob receives a mail from "Jane Doe":
  "Bob, please transfer the money I have won to . . .
- How can Bob verify that this mail is from Alice?

Entity Recognition: More Formal



- **The initial communication** may be observed – but not tampered with

Entity Recognition: More Formal

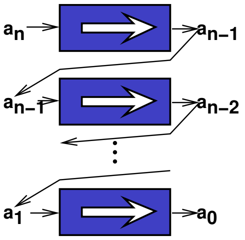


- **The initial communication** may be observed – but not tampered with
- **Later**, communication may be observed but also tampered with (read, modify, suppress, or re-send data).

"Cheap" Symmetric Operations

Hash Chain

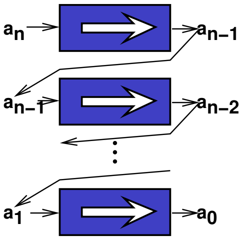
$a_0 := \mathbf{H}(\mathbf{H}(\ldots(\mathbf{H}(\mathbf{a_n})))):$



"Cheap" Symmetric Operations

Hash Chain

$a_0 := \mathbf{H}(\mathbf{H}(\ldots(\mathbf{H}(\mathbf{a_n})))):$



Assumption (one-wayness):

given $\mathbf{a_{i-1}}$:

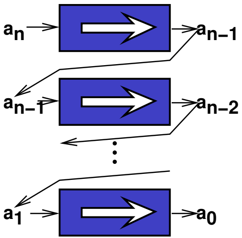
infeasible to find any $\mathbf{a'}$ with

$\mathbf{H}(\mathbf{a'}) = \mathbf{a_{i-1}}$

“Cheap” Symmetric Operations

Hash Chain

$a_0 := \mathbf{H}(\mathbf{H}(\ldots(\mathbf{H}(\mathbf{a_n}))))$:



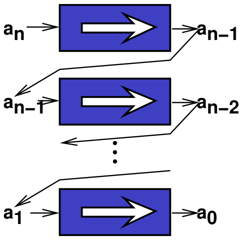
Assumption (one-wayness):
given $\mathbf{a_{i-1}}$:
infeasible to find any $\mathbf{a'}$ with
$\mathbf{H}(\mathbf{a'}) = \mathbf{a_{i-1}}$

Message Authentication Code (MAC)

"Cheap" Symmetric Operations

Hash Chain

$a_0 := \mathbf{H}(\mathbf{H}(\ldots(\mathbf{H}(\mathbf{a_n})))) :$



Assumption (one-wayness):
given $\mathbf{a_{i-1}}$:
infeasible to find any $\mathbf{a'}$ with
$\mathbf{H}(\mathbf{a'}) = \mathbf{a_{i-1}}$

Message Authentication Code (MAC)

Assumption (secure against existential forgery):
given many (tag,message) pairs:
infeasible to forge tag for another message

Protocols for Entity Recognition

- "Zero Common-Knowledge": Weimerskirch, Westhoff, 2003.
- "Jane Doe": (not yet using that name):
  Lucks, Zenner, Weimerskirch, Westhoff, 2005.
- "Jane Doe": Lucks, Zenner, Weimerskirch, Westhoff, 2008.

The Zero Common Knowledge Protocol

- one hash chain $(a_0, a_1, \ldots)$ for Alice, another one $(b_0, b_1, \ldots)$ for Bob.
- Bob knows $\mathbf{a_{i-1}}$, Alice knows $\mathbf{b_{j-1}}$.
- Alice authenticates m by tag := $\mathrm{MAC}_{\mathbf{a_{j+1}}}(m)$, and $\mathbf{a_j}$.
- Bob verifies $\mathbf{H}(\mathbf{a_i}) = \mathbf{a_{i-1}}$ and responds $\mathbf{b_i}$.
  Alice verifies $\mathbf{H}(\mathbf{b_i}) = \mathbf{b_{i-1}}$.
- Alice sends $\mathbf{a_{j+1}}$.
  Bob verifies $\mathbf{H}(\mathbf{a_{j+1}}) = \mathbf{a_j}$ and $\mathrm{MAC}_{\mathbf{a_{j+1}}}(m) = \text{tag}$.

The Attack

- one hash chain $(a_0, a_1, \ldots)$ for Alice, another one $(b_0, b_1, \ldots)$ for Bob.
- Bob knows $\mathbf{a_{i-1}}$, Alice knows $\mathbf{b_{j-1}}$.
- Alice authenticates m by tag $:= \mathrm{MAC}_{\mathbf{a_{j+1}}}(m)$, and $\mathbf{a_j}$.
- Bob verifies $\mathbf{H}(\mathbf{a_i}) = \mathbf{a_{i-1}}$ and responds $\mathbf{b_i}$.
  Alice verifies $\mathbf{H}(\mathbf{b_i}) = \mathbf{b_{i-1}}$.
- Alice sends $\mathbf{a_{j+1}}$. **Eve does not deliver this to Bob.**

The Attack

- one hash chain $(a_0, a_1, \ldots)$ for Alice, another one $(b_0, b_1, \ldots)$ for Bob.
- Bob knows $\mathbf{a_{i-1}}$, Alice knows $\mathbf{b_{j-1}}$.
- Alice authenticates m by tag $:= \mathrm{MAC}_{\mathbf{a_{j+1}}}(m)$, and $\mathbf{a_j}$.
- Bob verifies $\mathbf{H}(\mathbf{a_i}) = \mathbf{a_{i-1}}$ and responds $\mathbf{b_i}$.
  Alice verifies $\mathbf{H}(\mathbf{b_i}) = \mathbf{b_{i-1}}$.
- Alice sends $\mathbf{a_{j+1}}$. **Eve does not deliver this to Bob.**
- Next time, Alice will send tag $:= \mathrm{MAC}(\ldots)$, and $\mathbf{a_{j+2}}$.
- Eve then knows $\mathbf{a_{j+1}}$ and $\mathbf{a_{j+2}}$ from Alice's hash chain, which are unknown to Bob.
- This allows her to forge a message for Bob.

Comments

- The *Zero Common Knowledge Protocol* has been published at SAC 2003 and *proven secure!*
- The proof makes the (implicit) assumption that messages are always delivered.

Comments

- The *Zero Common Knowledge Protocol* has been published at SAC 2003 and *proven secure!*
- The proof makes the (implicit) assumption that messages are always delivered.
- We could "repair" this by *making the assumption explicit*.
- The proof would be theoretically sound, but (most probably) practically useless!

The Problem with Proofs

- Lars Knudsen: *"If it is provably secure, it is probably not"*
- Birgit Pfitzmann, Michael Waidner: *"How To Break and Repair A 'Provably Secure' Untraceable Payment System"*, CRYPTO 1991
- Birgit Pfitzmann, Matthias Schunter, Michael Waidner: *"How to Break Another Provably Secure Payment System"*, EUROCRYPT 1995:

  > *"Short statements of cryptographic properties (formal or informal) should always come with an explicit trust model, i.e., for whom a property is guaranteed, and which other participants have to be trusted to guarantee this."*

The Trinity of Cryptographic Security Proofs

A cryptographic “Security Proof” is actually a trinity of

1. some definitions (trust model, cryptographic assumptions, . . .),
2. a theorem, and
3. the proof itself.

The Trinity of Cryptographic Security Proofs

A cryptographic "Security Proof" is actually a trinity of

1. some definitions (trust model, cryptographic assumptions, ...),
2. a theorem, and
3. the proof itself.

- Leave it to the theoreticians to verify the proof.

The Trinity of Cryptographic Security Proofs

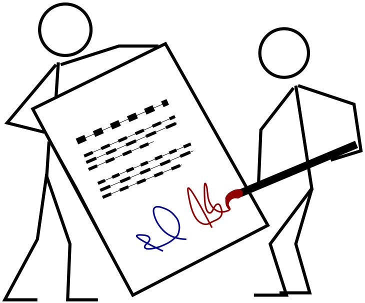
A cryptographic "Security Proof" is actually a trinity of

1. some definitions (trust model, cryptographic assumptions, ...),
2. a theorem, and
3. the proof itself.

- Leave it to the theoreticians to verify the proof.
- But understand that the theorem, with the associated definitions, is like a contract between
  - a **service provider** (the crypto-designer) and
  - a **client** (the application designer).

The Proof as a Contract

- obligations for the **client**, such as choosing primitives (MAC, Hash, ...), secure against well-defined classes of attack
- responsibility of the **service provider** (security against the specified class of attacks in the specified trust model)
- if the **client** follows her obligations, **mathematical laws** guarantee the promised security

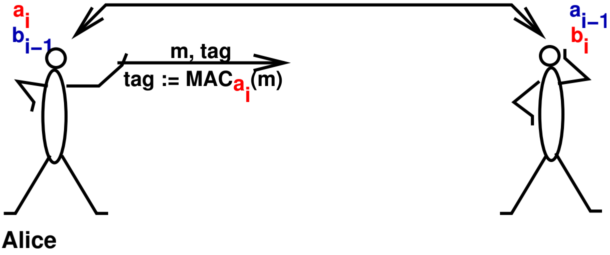


This is an extremely useful concept for Applied Cryptography!
But **the client must understand the contract!**

The Jane Doe Protocol

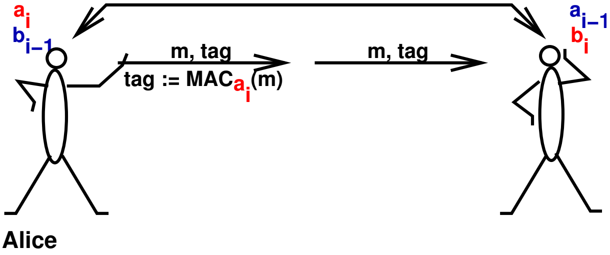
- Two hash chains $(a_0, a_1, \ldots)$ for Alice, and $(b_0, b_1, \ldots)$ for Bob. Initially, Alice knows $\mathbf{b_{i-1}}$ and Bob knows $\mathbf{a_{i-1}}$.
- Later, Alice will reveal $\mathbf{a_i}$, and Bob will reveal $\mathbf{b_i}$.

The Jane Doe Protocol



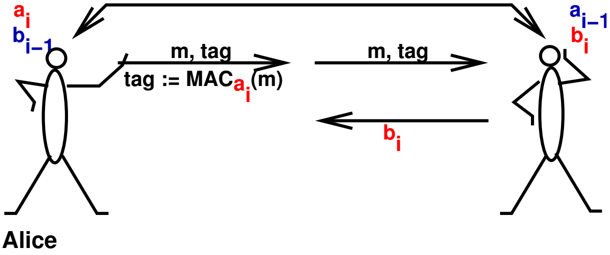
- Two hash chains $(a_0, a_1, \ldots)$ for Alice, and $(b_0, b_1, \ldots)$ for Bob. Initially, Alice knows $\mathbf{b_{i-1}}$ and Bob knows $\mathbf{a_{i-1}}$.
- Later, Alice will reveal $\mathbf{a_i}$, and Bob will reveal $\mathbf{b_i}$.

The Jane Doe Protocol



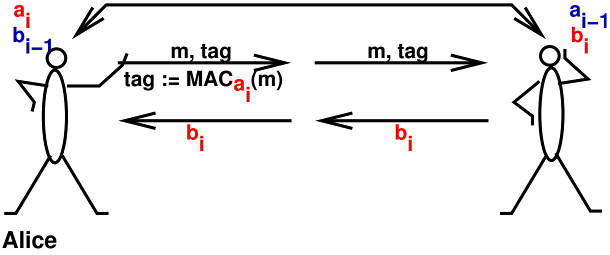
- Two hash chains $(a_0, a_1, \ldots)$ for Alice, and $(b_0, b_1, \ldots)$ for Bob. Initially, Alice knows $\mathbf{b_{i-1}}$ and Bob knows $\mathbf{a_{i-1}}$.
- Later, Alice will reveal $\mathbf{a_i}$, and Bob will reveal $\mathbf{b_i}$.
- Alice sends message m and tag $:= \mathrm{MAC}_{a_i}(m)$.

The Jane Doe Protocol



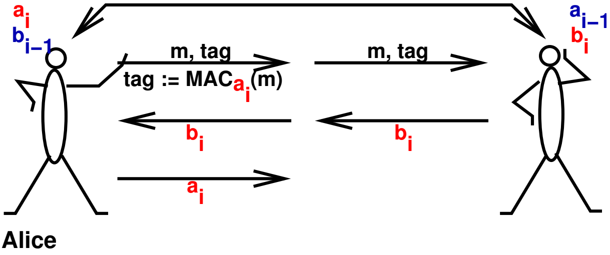
- Two hash chains $(a_0, a_1, \ldots)$ for Alice, and $(b_0, b_1, \ldots)$ for Bob. Initially, Alice knows $\mathbf{b_{i-1}}$ and Bob knows $\mathbf{a_{i-1}}$.
- Later, Alice will reveal $\mathbf{a_i}$, and Bob will reveal $\mathbf{b_i}$.
- Alice sends message m and tag $:= \mathrm{MAC}_{a_i}(m)$.

The Jane Doe Protocol



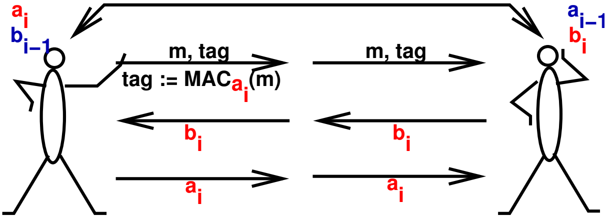
- Two hash chains $(a_0, a_1, \ldots)$ for Alice, and $(b_0, b_1, \ldots)$ for Bob. Initially, Alice knows $\mathbf{b_{i-1}}$ and Bob knows $\mathbf{a_{i-1}}$.
- Later, Alice will reveal $\mathbf{a_i}$, and Bob will reveal $\mathbf{b_i}$.
- Alice sends message m and tag $:= \text{MAC}_{a_i}(m)$.
- Bob responds with $\mathbf{b_i}$. $\boxed{\text{Alice verifies } \mathbf{H}(\mathbf{b_i}) = \mathbf{b_{i-1}}.}$

The Jane Doe Protocol



- Two hash chains $(a_0, a_1, \ldots)$ for Alice, and $(b_0, b_1, \ldots)$ for Bob. Initially, Alice knows $\mathbf{b_{i-1}}$ and Bob knows $\mathbf{a_{i-1}}$.
- Later, Alice will reveal $\mathbf{a_i}$, and Bob will reveal $\mathbf{b_i}$.
- Alice sends message m and tag $:= \mathrm{MAC}_{a_i}(m)$.
- Bob responds with $\mathbf{b_i}$. $\boxed{\text{Alice verifies } \mathbf{H}(\mathbf{b_i}) = \mathbf{b_{i-1}}.}$

The Jane Doe Protocol

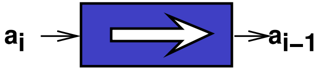


- Two hash chains $(a_0, a_1, \dots)$ for Alice, and $(b_0, b_1, \dots)$ for Bob. Initially, Alice knows $\mathbf{b_{i-1}}$ and Bob knows $\mathbf{a_{i-1}}$.
- Later, Alice will reveal $\mathbf{a_i}$, and Bob will reveal $\mathbf{b_i}$.
- Alice sends message m and tag $:= \text{MAC}_{a_i}(m)$.
- Bob responds with $\mathbf{b_i}$. $\boxed{\text{Alice verifies } \mathbf{H}(\mathbf{b_i}) = \mathbf{b_{i-1}}.}$
- Alice sends $\mathbf{a_i}$.

  $\boxed{\text{Bob verifies } \mathbf{H}(\mathbf{a_i}) = \mathbf{a_{i-1}} \text{ and } \text{MAC}_{\mathbf{a_i}}(m) = \text{tag}.}$

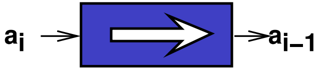
Assumptions

We need both a one-way (hash) function and secure MAC.



Assumptions

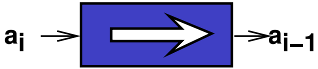
We need both a one-way (hash) function and secure MAC.



Problem: For the same Key, the protocol uses both Hash(Key) and MAC(Key,$*$). Thus, MAC(Key,$*$) must provide security, even if Hash(Key) is known.

Assumptions

We need both a one-way (hash) function and secure MAC.



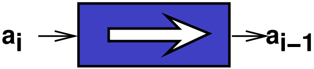
Problem: For the same Key, the protocol uses both Hash(Key) and MAC(Key,$*$). Thus, MAC(Key,$*$) must provide security, even if Hash(Key) is known.

Here comes the problem:

- One can (maliciously) define a secure Hash and a secure MAC

Assumptions

We need both a one-way (hash) function and secure MAC.



Problem: For the same Key, the protocol uses both Hash(Key) and MAC(Key,$*$). Thus, MAC(Key,$*$) must provide security, even if Hash(Key) is known.

Here comes the problem:

- One can (maliciously) define a secure Hash and a secure MAC
- such that the Jane Doe protocol is actually *insecure* when using both of them together.

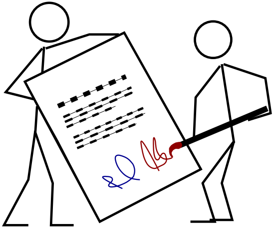
Two Alternative Solutions

1. introduce a complex nonstandard security definition for the *combined security* of MAC and Hash
   (this is, what we actually did 2005)
2. model Hash as a random oracle

Two Alternative Solutions

1. introduce a complex nonstandard security definition for the *combined security* of MAC and Hash
(this is, what we actually did 2005)
2. model Hash as a random oracle

If the security definitions are complex and nonstandard, the **client** will find it hard to understand the contract. That is bad!



Definitions in the random oracle model are quite easy to understand.
So why not just assume the Hash is a random oracle?

The Random Oracle Debate

Two reasons to criticize the usage of random oracles:

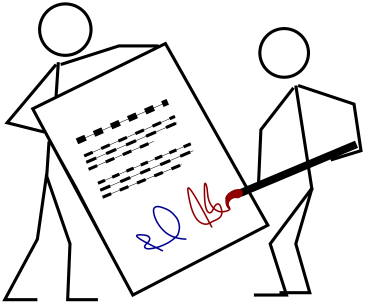
1. "Separation Results":
   Some (maliciously designed) cryptosystems are provable secure in the ROM, but insecure under any real-world instantiation.
   So far, this is a theoretical issue, only.
2. "Spoiling the Contract":
   Even if "good" primitives exist, the definition and the theorem don't tell the **client** how to choose "good" primitives.

Spoiling the Contract

- no “real world object”
- **client** must choose a real-world object, and thus (formally) violate her obligations

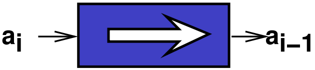
Spoiling the Contract

- no "real world object"
- **client** must choose a real-world object, and thus (formally) violate her obligations
- **no guarantee by mathematical laws** for **client**
  "Trust me! If the primitive is good, you are secure."
  "Sorry, the primitive you used is not good! That is not my fault!"



The Jane Doe Protocol (revisited)

We need both a one-way (hash) function and secure MAC.



Problem: For the same Key, the protocol uses both Hash(Key) and MAC(Key,$*$). Thus, MAC(Key,$*$) must provide security, even if Hash(Key) is known.

Same Protocol – but Slightly Different Requirements

Start with a single primitive m^* (a MAC):

- assume m^* to be one-way, <u>and</u>
- assume m^* to be secure against existential forgeries.

Derive a one-way function h and a new MAC m from M^*:

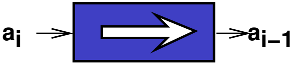
Same Protocol – but Slightly Different Requirements

Start with a single primitive m^* (a MAC):

- assume m^* to be one-way, and
- assume m^* to be secure against existential forgeries.

Derive a one-way function h and a new MAC m from M^*:

One-way function: $h(\text{Key}) := m^*(\text{Key}, 0)$



New MAC: $m(\text{Key}, \text{Message}) := m^*(\text{Key}, 1||\text{Message})$

Summary

- Security proofs are an important tool for Applied Cryptography.

- If you want to benefit from security proofs, you must understand what the proof is about (the definitions and the theorem, not necessarily the proof itself).
- If you want people benefit from your proofs, try to make reading and understanding your definitions and your theorem as simple as possible.

- Theoretical abstractions (random oracle model, ideal cipher model, . . .) help to avoid complex definitions, but hinder the selection of real primitives (famous example: TDES-RMAC).