Lecture 3: Authentication Protocols

Jeannette M. Wing

15-827 Security and Cryptography

Plan for Today

- Identification and Entity Authentication (Chapter 10)
  - Passwords, Biometrics
  - Zero-Knowledge Proofs
- Key Establishment Through Key Transport (Chapter 12)
  - Needham-Schroeder
  - Needham-Schroeder Fixed
    - Denning-Sacco Fix
    - Needham-Schroeder Fix
  - Otway-Rees
  - Wide-Mouthed Frog
  - Kerberos
- Protocol Weaknesses and Design Principles
  - Three-pass protocol
  - TMN
  - CCITT X.509

Goals for Today

- Main Goals
  - Expose you to well-known authentication protocols.
  - Expose you to well-known or typical flaws found.

- Meta-Goal
  - Cause you some uneasiness.
    - A security protocol based on strong cryptographic techniques doesn't mean it's secure.
    - Lots of places to look for protocol weaknesses.
    - It's not easy to design a security protocol!

Entity Authentication

- Problem: Alice wants to prove her identity to someone (Bob) or something (computer). Usually we want two-way (mutual) authentication.

Solution: “Weak” Authentication

- Passwords, PINS
  - Problems: easy to guess, easy to forget, easy to capture (in transit), easy to do exhaustive search, …
  - Partial solution to dictionary attack: Use *salt,* a random bit string; apply one-way hash function to “password || salt” before storing <hash(password || salt), salt> in password file. Makes exhaustive search more difficult.
- Biometrics
  - Fingerprints, retinal scans, voice, facial recognition,

    hand geometry, signature dynamics, typing characteristics, ...
  - Problems: they are inconvenient or uncomfortable; body parts become valuable.

Solution: Challenge-Response Authentication

- General idea
  - One party sends a challenge message.
  - Second party sends a response message in a pre-agreed manner that demonstrates identity.
  - Usually a time constraint too; if response arrives too slowly the authentication is not successful
  - Use nonces, timestamps, sequence numbers for "freshness." (Section 10.3.1)
    - Nonce: a number used only once.

Example Using Public-Key

1. B -> A: r
2. A -> B: $S_A(r)$

Bob accepts message after checking Alice's signature.

Problems:

- Bob has a way to get Alice to sign anything. ("r" = "Alice hereby gives Bob all her money.")
- Alice gives some info (her signature) to Bob. Bob now has a way to conduct a chosen-plaintext attack.
  - Solve this by having Alice send a random number back to Bob and Bob sends back a third message containing both random numbers.

Challenge-Response with Zero-Knowledge Protocols

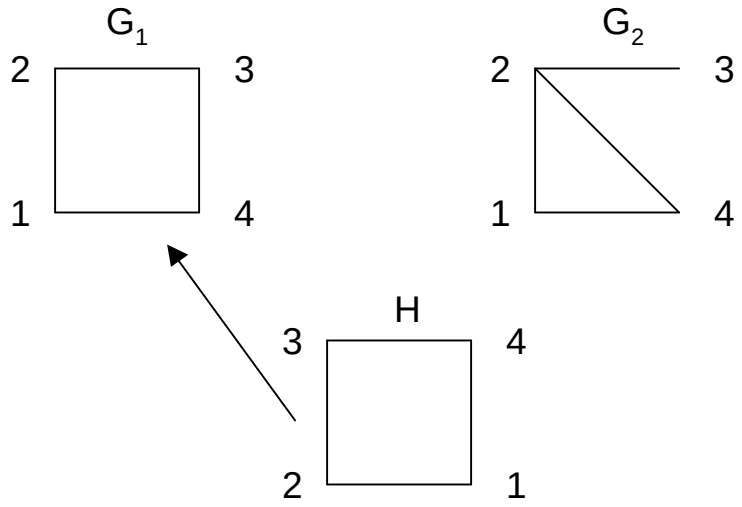
- Solution: Use ZK protocol to allow *prover* (Alice) to demonstrate knowledge of a secret without revealing the secret (or any other useful information) to *verifier* (Bob).
- General structure of protocol

```
A -> B: witness
B -> A: challenge
A -> B: response
```

- Assumptions
  - Bob's computations are done in polynomial time. Alice can be all powerful.
- Properties
  - Bogus Alice will be caught with high probability.
    - In any protocol there's a chance bogus Alice will guess the correct response but by repeating the process we can raise the probability of catching Alice arbitrarily high.
  - Alice leaks no information not already computable by Bob.
    - Hence the term "zero-knowledge"

Impractical example based on graph isomorphism

- GRAPH ISOMORPHISM: Given two graphs, G_1 and G_2, are they isomorphic?
  - No polynomial-time algorithm is known to solve this problem but it is not known to be NP-complete.



Perfect Zero-Knowledge Interactive Proof for Graph Isomorphism

Input: Two graphs, G_1 and G_2, each having vertex set {1, …, *n*}.

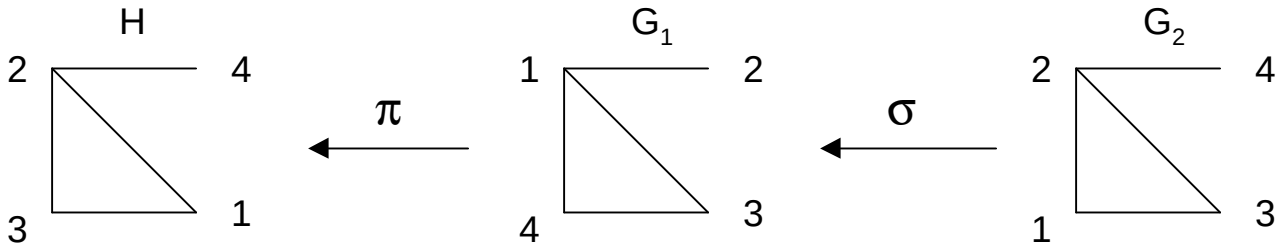
1. Repeat the following *n* times:
2. Alice chooses a random permutation π of {1, …, *n*}. She computes H to be the image of G_1 under π and sends H to Bob.
3. Bob chooses randomly $i = 1$ or 2 and send value to Alice.
4. Alice computes a permutation ρ of {1, …, *n*} s.t. H is the image of G_i under ρ. She sends ρ to Bob. (So if $i = 1$, then $\rho = \pi$; if $i = 2$, $\rho = \pi \circ \sigma$.)
5. Bob checks to see if H is the image of G_i under ρ.
6. Bob accepts Alice's proof if H is the image of G_i in each of the *n* rounds.

Example

Suppose Alice chooses permutation π = (2 4 1 3).

If Bob's challenge *i* = 1, then Alice gives Bob the permutation π.
Bob checks that the image of G_1 under π is H.

If Bob's challenge *i* = 2, then Alice gives Bob the composition
$\rho = \pi \circ \sigma$ = (3 2 1 4) and Bob checks that the image of G_2 under ρ is H.



Three Properties of ZK Protocols

1. *Completeness:* If problem instance is "yes" then Bob must accept Alice's proof.

   E.g., if G_1 and G_2 are isomorphic the probability that Bob accepts is 1.

2. *Soundness:* If problem instance is "no" then the probability that Bob accepts is "very small."

   E.g., if G_1 and G_2 are not isomorphic the only way Alice can deceive Bob is for her to guess correctly the value of *i* that Bob chooses in each round and compute a (random) isomorphic copy of G_i. Her probability of correctly guessing Bob's n random challenges is 2^{-n}.

3. *Zero-knowledge:*

   The transcript (exchange of messages Alice and Bob have) could just as easily have been produced (forged) by Bob (or anyone else) with the same probability distribution of the true conversation. Hence, zero-knowledge.

Computational Zero-Knowledge

- It would be more useful to have ZK proof systems for problems that are known to be NP-complete.
  - However, there is theoretical evidence that perfect ZK proofs do not exist for NP-complete problems.
- So, in practice we try for something weaker--*computational ZK*.

Fiat-Shamir Identification Protocol (Basic)

One-time setup:

- Trusted T publishes RSA-like modulus n = pq but keeps primes p and q secret.
- A and B each select a secret s relatively prime to n $1 \leq s \leq n-1$, computes $v = s^2 \bmod n$ and registers v with T as its public key.

Protocol: Iterate the following *t* times. B accepts proof if all *t* rounds succeed:

1. A -> B: $x = r^2 \bmod n$	A chooses random (commitment) r and sends witness (r^2).
2. B -> A: e in {0, 1}	B randomly chooses and sends challenge e.
3. A -> B: $y = rs^e \bmod n$	A computes and sends response y. Either y = r (if e = 0) or y = rs mod n (if e = 1).

B rejects proof if y = 0, and otherwise accepts upon verifying $y^2 = xv^e \bmod n$.

- Depending on e, $y^2 = x$ or $y^2 = xv \bmod n$, since $v = s^2 \bmod n$.
- Checking y = 0 precludes case r = 0.

- Based on difficulty of extracting square roots modulo large composite integers *n* of unknown factorization, which is equivalent to factoring *n*.

Summary of Zero-Knowledge Interactive Protocols

- General structure of protocol

  A -> B: witness

  B -> A: challenge

  A -> B: response

- Combines ideas of cut-and-choose and challenge/response:
  - Cut-and-choose: two children share a piece of cake. One cuts; the other chooses.
  - A responds to at most one challenge (question) for a given witness, and should not reuse any witness.

Level of Abstraction Change

Perfect Encryption Assumption:

A principal must possess a key K in order to use K to encrypt or decrypt a message, or to obtain any information whatsoever about a message encrypted with K.

(In a public-key system, a principal needs the private key to decrypt a message.)

Key Transport Protocols: Starters

- Protocol
  - A sequence of messages among parties.
    - **A**lice, **B**ob, Trusted **S**erver, Intruder **Z**
- Entity authentication goal
  - Alice and Bob prove their identity to each other.
- Key-exchange goal
  - Alice and Bob use a shared secret **session key** to communicate.

Notation (in Literature, not Textbook)

A, B principals

S trusted server

Z intruder

K_{ab} key shared by A and B

K_a, K_a^{-1} public, private key of A

$\{M\}_K$ encryption of message M with key K

N_a nonce generated by A

A -> B: X, Y

A sends to B a message containing two parts, X and Y.

A -> Z(B): M

Z intercepts message M sent from A to B.

Z(A) -> B: M

Z masquerades as A sending M to B.

Needham-Schroeder [1978]

1. A -> S: A, B, N_a
2. S -> A: $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. A -> B: $\{K_{ab}, A\}_{K_{bs}}$
4. B -> A: $\{N_b\}_{K_{ab}}$
5. A -> B: $\{N_b - 1\}_{K_{ab}}$

Flaw #1: Denning-Sacco Replay Attack

3. Z(A) -> B: $\{CK, A\}_{K_{bs}}$
4. B -> Z(A): $\{N_b\}_{CK}$
5. Z(A) -> B: $\{N_b - 1\}_{CK}$

Denning-Sacco Fix [1981]: Use Timestamps

1. A -> S: A, B
2. S -> A: $\{B, K_{ab}, \mathbf{T}, \{K_{ab}, A, \mathbf{T}\}_{K_{bs}}\}_{K_{as}}$
3. A -> B: $\{K_{ab}, A, \mathbf{T}\}_{K_{bs}}$

Needham-Schroeder Fix [1987]: Use Nonces

-1. A -> B: A

0. B -> A: $\{A, J\}_{K_{bs}}$

1. A -> S: $A, B, N_a, \{A, J\}_{K_{bs}}$
2. S -> A: $\{N_a, B, K_{ab}, J, \{K_{ab}, A, J\}_{K_{bs}}\}_{K_{as}}$
3. A -> B: $\{K_{ab}, A, J\}_{K_{bs}}$
4. B -> A: $\{N_b\}_{K_{ab}}$
5. A -> B: $\{N_b - 1\}_{K_{ab}}$

Other Flaws in Needham-Schroeder Protocols

- Crypto-level
  - Nonces [Boyd90]
- Public-key protocol (Protocol 12.38)
  - Interleaved runs [Lowe96]

Flaw in Needham-Schroeder Public-Key [Lowe96]

Reduced protocol:

1. A -> B: A, B, $\{N_a, A\}_{K_b}$
2. B -> A: B, A, $\{N_a, N_b\}_{K_a}$
3. A -> B: A, B, $\{N_b\}_{K_b}$

The attack (interleaving of two runs):

α .1. A -> Z: A, Z, $\{N_a, A\}_{K_z}$

β .1. Z(A) -> B: A, B, $\{N_a, A\}_{K_b}$

β .2. B -> Z(A): B, A, $\{N_a, N_b\}_{K_a}$

α .2. Z -> A: Z, A, $\{N_a, N_b\}_{K_a}$

α .3. A -> Z: A, Z, $\{N_b\}_{K_z}$

β .3. Z(A) -> B: A, B, $\{N_b\}_{K_b}$

The Fix

- Add identity of sender in M2.

  1. A -> B: A, B, $\{N_a, A\}_{K_b}$
  2. B -> A: B, A, $\{N_a, N_b, \mathbf{B}\}_{K_a}$
  3. A -> B: A, B, $\{N_b\}_{K_b}$

- Prevents attack because $\beta.2$ would become:

  $\beta.2$. B -> Z(A): B, A, $\{N_a, N_b, \mathbf{B}\}_{K_a}$

  and intruder cannot successfully replay $\alpha.2$, because A is expecting a message containing Z's identity.

Otway-Rees [1987]

1. A -> B: $M, A, B, \{N_a, M, A, B\}_{K_{as}}$
2. B -> S: $M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}$
3. S -> B: $M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
4. B -> A: $M, \{N_a, K_{ab}\}_{K_{as}}$

Flaw # 1: Type Flaw

1. A -> Z(B): M, A, B, $\{N_a, M, A, B\}_{K_{as}}$

.

.

.

4. Z(B) -> A: M, $\{N_a, M, A, B\}_{K_{as}}$

Flaw #2: M needs to be an unpredictable nonce.

Run 1

1. A -> B: 007, A, B, $\{N_a, 007, A, B\}_{K_{as}}$
1'. Z(A) -> B: 008, A, B, garbage
2. B -> S: 008, A, B, garbage , $\{N_b, 008, A, B\}_{K_{bs}}$
3. S rejects B's message.

Run 2

1. A -> B: 008, A, B, $\{N_a, 008, A, B\}_{K_{as}}$
2. Z(B) -> S: 008, A, B, $\{N_a, 008, A, B\}_{K_{as}}$, $\{N_b, 008, A, B\}_{K_{bs}}$
3. S -> Z(B): 008, $\{N_a, K_{ab}\}_{K_{as}}$, $\{N_b, K_{ab}\}_{K_{bs}}$
4. Z(B) -> A: 008, $\{N_a, K_{ab}\}_{K_{as}}$

Wide-Mouthed Frog [Burrows]

1. A -> S: $A, \{T_a, B, K_{ab}\}_{K_{as}}$
2. S -> B: $\{T_s, A, K_{ab}\}_{K_{bs}}$

Flaws/Weaknesses

- Flaw/Weakness #0
  - Assumption that A can generate good session keys.
- Flaw/Weakness #1
  - Assumption of synchronized clocks.
    - Replay 1st message within appropriate time window.
    - Causes reauthentication since S will produce a new second message with an updated timestamp.

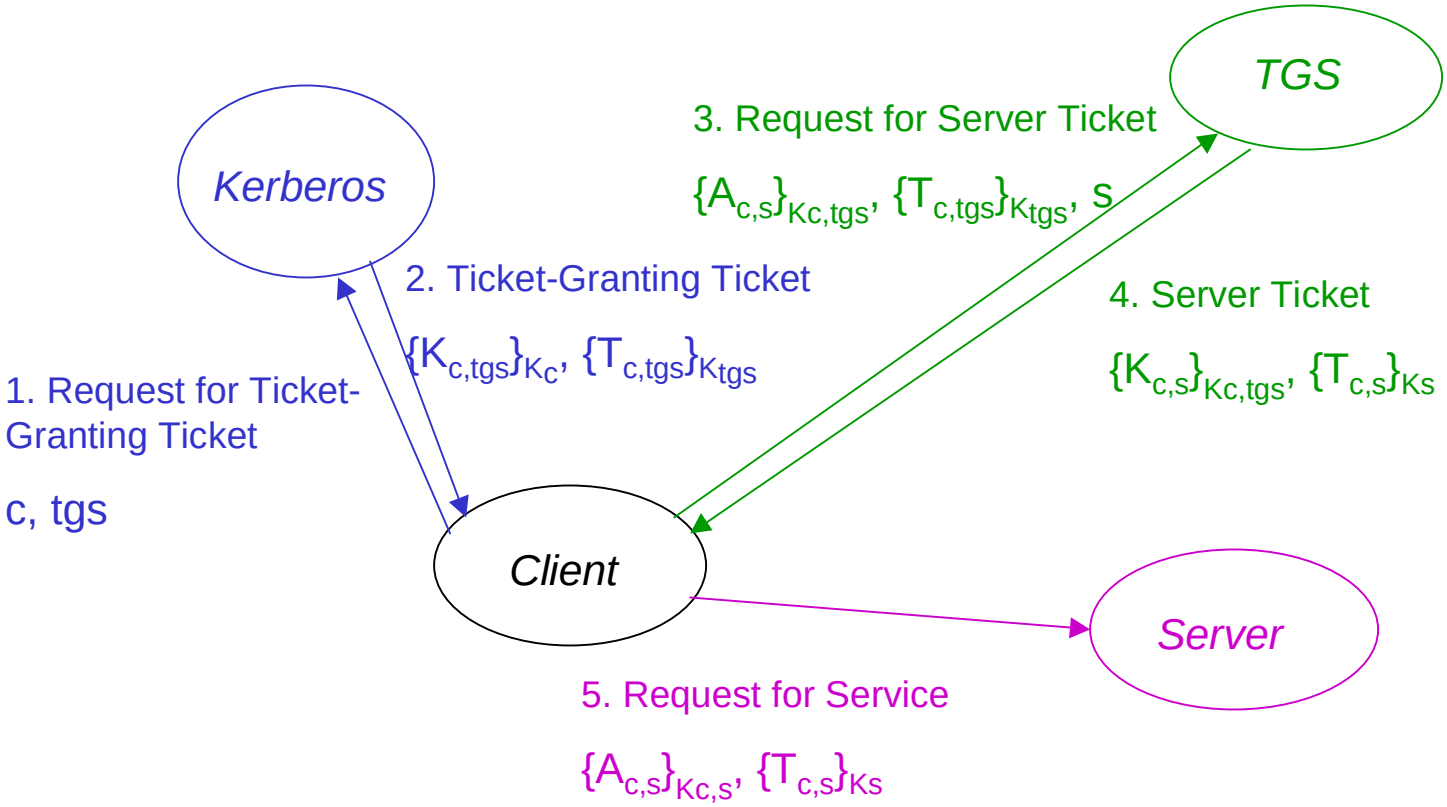
Flaw #2

1. A -> S: $A, \{T_a, B, K_{ab}\}_{K_{as}}$
2. S -> B: $\{T_s, A, K_{ab}\}_{K_{bs}}$
1'. Z(B) -> S: $B, \{T_s, A, K_{ab}\}_{K_{bs}}$
2'. S -> Z(A) : $\{T_s', B, K_{ab}\}_{K_{as}}$
1''. Z(A) -> S: $A, \{T_s', B, K_{ab}\}_{K_{as}}$
2''. S -> Z(B): $\{T_s'', A, K_{ab}\}_{K_{bs}}$

Now Z can replay appropriate messages to A and B:

2'''. Z(S) -> A: $\{T_s', B, K_{ab}\}_{K_{as}}$
2'''. Z(S) -> B: $\{T_s'', A, K_{ab}\}_{K_{bs}}$

Kerberos (Version 5)



Kerberos Tickets

Ticket

$T_{c,s} = s, \{c, addr, lifetime, K_{c,s}, \ldots\}_{K_s}$

Authenticator

$\{A_{c,s}\}_{K_{c,s}} = \{c, addr, timestamp, ...\}_{K\ c,s}$

Kerberos Weaknesses

- Secure and synchronized clocks
  - Implied by use of timestamps.
- Password-guessing attacks
  - In real implementations initial shared keys are password-derived.
- Replay attacks
  - Reuse of authenticators within lifetime period (e.g., 8 hours).
- See Bellovin and Merritt paper (on v4, but criticisms hold for v5 too) for more.
- These are weaknesses in the overall protocol, not with the underlying cryptography!

Weaknesses Summarized

- Password-guessing flaws
  - Users do not choose long enough or truly random passwords.
    - E.g., Kerberos, any login protocol.
- Freshness flaws
  - Adversary can masquerade as another principal.
    - E.g., Needham-Schroeder symmetric key, Andrew Secure RPC.
  - Certificates/signatures that have long lifetimes.
    - Revocation of public keys is difficult.
- Type flaws
  - Bit strings with multiple interpretations, multiple parses.
    - E.g., Otway-Rees ("K_{ab}" and "M, A, B")
  - Replacement of message or its parts by random bit strings.

Weaknesses (cont'd)

- Oracle flaws
  - Using a principal/server to do a local computation for you to your advantage.
    - E.g., three-pass protocol (next slide).
- Timing flaws
  - Synchronized clocks.
    - E.g., Kerberos.
- Cryptosystem/crypto-protocol interface flaws
  - Choice of underlying cryptosystem may matter.
    - E.g., three-pass protocol: If the XOR function is used to encrypt messages, then Z can find the plaintext by adding two encrypted messages transferred between A and B.
    - E.g., TMN (two slides after next).

Three-Pass Protocol

- Applications: Mental poker, remote coin flipping. Due to Shamir, Rivest, Adleman.
- Assume crypto function is commutative: $\{\{M\}_{K_a}\}_{K_b} = \{\{M\}_{K_b}\}_{K_a}$
- The Protocol
  1. A -> B: $\{M\}_{K_a}$
  2. B -> A: $\{\{M\}_{K_a}\}_{K_b}$ Alice decrypts with K_a.
  3. A -> B: $\{M\}_{K_b}$
- The Problem
  1. A -> Z(B): $\{M\}_{K_a}$
  2. Z(B) -> A: $\{M\}_{K_a}$ Alice decrypts with K_a!
  3. A -> Z(B): M

Simmon's Attack on Tatebayashi-Matsuzaki-Newmann (TMN)

- The Protocol: A and B are users, e.g., smartcards. Trusted S can factor n. The result is that A and B can use r_B as a shared secret key.

  A -> S: $r_A^3 \mod n$

  B -> S: $r_B^3 \mod n$

  S -> A: r_A *xor* r_B S can factor n => S can extract cube roots.

- The Problem: If Charlie and David conspire, or even if David just generates a predictable random number r_D, then Charlie can get r_B:

  C -> S: $r_B^3 r_C^3 \mod n$

  D -> S: $r_D^3 \mod n$

  S -> C: $r_B r_C$ *xor* r_D

Lessons

- Don’t trust in the secrecy of other people’s secrets.
- Always be careful when signing or decrypting data that we don’t be an oracle for someone.

Weaknesses (cont'd)

- Multiple runs of protocol (interleaved runs).
  - E.g., Lowe's bug in Needham-Schroeder PK protocol.
- Unstated assumptions
  - E.g., Wide-mouthed frog flaw #0.

Design Principles

- Be explicit.
- Know why you're encrypting.
  - E.g., elimination of double encryption from Kerberos v4 to v5. (See next slide.)
- Know when to sign and when to encrypt and which to do first.
  - E.g., CCITT X.509 (See later slides.)
- Be careful about freshness and time.
- Don't be an oracle.
- Know what assumptions about the underlying cryptosystem you are making or exploiting.

Encryption in Kerberos V4

1. A -> S: A, B
2. S -> A: $\{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. A -> B: $\{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$
4. B -> A: $\{T_{a+1}\}_{K_{ab}}$

M1. No encryption needed.

M2. Double encryption not needed. (Inherited from N-S. Removed in Kerberos v5.)

M3. Prove knowledge of K_{ab} near time T_a. Encryption in second part of M3 may be redundant if T_a is not much different from T_s. (And so could eliminate this part of M3 B can use T_s is M4.)

M4. Prove knowledge of K_{ab} near time T_a

Encrypting Encrypted Messages

- Use three-pass protocol for remote coin flipping:
  1. A -> B: $\{M_1\}_{Ka}$, $\{M_2\}_{Ka}$
  2. B -> A: $\{\{M_i\}_{Ka}\}_{Kb}$
  3. A -> B: $\{M_i\}_{Kb}$
  4. B -> A: M_i
  5. A -> B: K_a
  6. B -> A: K_b
- But if devious Bob can find two keys, b_1 and b_2, such that

  $\{\{M_1\}_{Ka}\}_{Kb1} = \{\{M_2\}_{Ka}\}_{Kb2}$ and $\{\{M_1\}_{Ka}\}_{Kb2} = \{\{M_2\}_{Ka}\}_{Kb1}$

  then he can win the coin flip each time.

- The **real lesson** here: Do not assume a message has a particular form unless you can verify it.
  - Even if it does, however, it might not be the *only* form (type flaw).

Signing Encrypted Data

CCITT X.509 one-message protocol:

1. A -> B: A, $\{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$

- Y_a is transferred in a signed message but what does A know of Y_a?
- Suppose Z intercepts the message, removes the existing signature, and then adds his own (blindly copying the encrypted section of message)?
  - Also B can find key K' such that $\{foo\}_{K'} = \{Y_a\}_{K_b}$. B registers K' with certification authority. B can convince judge that A sent foo not Y_a.
- Avoid this by signing first, then encrypting. (Assumes you know what you're signing!)

But "Signing Before Encrypting is not a Bill of Health" [AN96]

Denning and Sacco 1981:

1. A -> S: A, B
2. S -> A: CA, CB
3. A -> B: CA, CB, $\{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b}$

- M3 is supposed to mean "At time T_a A says K_{ab} is a good key for communicating between A and B."
- Sign before encrypt rule followed.
- Protocol still flawed because of absence of principal names (violation of [AN96] Principle 3). Need this fix:

  3. A -> B: CA, CB, $\{\{\mathbf{A, B,} K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b}$

More Design Principles

- State your assumptions explicitly.
- State your protocol design goals explicitly.
- Keep the protocol as simple as possible.
- See Abadi and Needham's and Anderson and Needham's papers.

Next lecture

Topics: Formal reasoning about authentication

- Logics: BAN
- Models: algebraic, state-machine models

Readings

For this lecture

- Relevant sections from Chapters 10, 12.
- Papers by Bellovin and Merritt, Abadi and Needham, Anderson and Needham ("Satan" and Robustness)

For next lecture

- Papers by Burrows, Abadi, and Needham SRC TR, Dolev and Yao, Abadi and Tuttle.