CS342 Computer Security **Handout # 4**
Profs. Daniel Bilar and Lyn Turbak **Friday, Sep. 15, 2006**
Wellesley College

Simple Protocols

Sources for this Lecture:

Kaufman, Perlman, and Speciner, *Network Security: PRIVATE Communication in a PUBLIC World*, Chapter 11 (*Security Handshake Protocols*)

Schneier, *Applied Cryptography*, Chapters 2–4 (*Protocol Building Blocks*, *Basic Protocols*, *Intermediate Protocols*)

What is a Protocol?

Schneier's definition: "A protocol is a series of steps, involving two or more parties, designed to accomplish a task."

Computational examples: low-level network protocols (UDP, TCP, IP), higher-level communication protocols (FTP, HTTP, SMTP, SSL), authentication (Kerberos), public key infrastructure, electronic voting, electronic money, ...

Simple Noncomputational Protocol Examples

- Exchanging names when meeting for the first time:

  Alice *(holding out hand to Bob)*: "Hi, I'm Alice".
  Bob *(shakes Alice's hand)*: "I'm Bob. Pleased to meet you."
  Conversation ensues.

- Establishing a phone conversation:

  Alice dials Bob's phone number.
  Bob *(answering ringing phone)*: "Hello."
  Alice: "May I please speak to Bob?"
  Bob: "This is Bob. Who's calling?"
  Alice: "This is Alice."
  Conversation ensues.

- Alice plans a pot-luck dinner party with a large group of friends. When is it? Who brings what? (Don't want only desserts!)

  – Many protocols for this problem involve a very large number of messages.

  – Here's a simple protocol in which Alice broadcasts a single message to all friends and needs no responses:

    I'm holding a pot-luck dinner party at my house on Fri. Sep. 15 at 7pm. Please come if you can make it. In order determine what kind of dish to bring, please flip two coins:

    ∗ if both are heads, bring an appetizer or salad;
    ∗ if both are tails, bring a dessert;
    ∗ otherwise (one head, one tail), bring a main dish.

Bicycle Transfer Protocols

Alice and Bob both work at Wellesley, but are never on campus at the same time. Alice wants to transfer her bicycle to Bob at Wellesley. How can she accomplish this in the following scenarios?

- Both Alice and Bob have keyed bicycle locks.

- Alice has a keyed bicycle lock.

- Neither Alice nor Bob has a lock.

Cryptographic Building Blocks

- Symmetric-key encryption/decryption:
  - Alice and Bob share a key K
  - encrypt a message M with key K: $E_K(M)$
  - decrypt a message M' with key K: $D_K(M')$
  - $D_K(E_K(M)) = M$
- One-way hash function H:
  - it's easy to compute $H(M) = h$.
  - given h, it's hard to find an M' such that $H(M') = h$.
  - can be combined with shared keys/encryption to yield a message authentication code (MAC), such as $H(\langle K, M\rangle)$, $E_K(H(M))$, $H(E_K(M))$.
- Public-key cryptography
  - Every participant has a public key K (published to the world) and a private key K' (known only by participant).
  - Encryption: $D_{K'}(E_K(M)) = M$. E.g., Bob uses Alice's public key to send her an encrypted message and only Alice can decrypt it. (Suffers from known plaintext attacks, since everyone has public key).
  - Digital signatures: $E_K(D_{K'}(M)) = M$. E.g., Alice signs a message with her private key and anyone can verify her signature.
  - Some public key systems (e.g. RSA) even have commutativity, a la the two-lock bicycle transfer protocol: $D_{K1'}(E_{K2}(E_{K1}(M))) = E_{K2}(M)$.
- Timestamps
  - Many protocols include the current time in a message to foil replay attacks.
  - Requires that participants have synchronized clocks, which can be challenging (maintained by other protocols).
  - Timestamps subject to clock-resetting attacks.
- Nonces = values used once (e.g., for unique IDs, challenges)
  - Typically a large random number, since hard for attacker to guess.
  - Timestamps and sequence numbers are often inappropriate, as we'll see, since easy for attackers to guess.

Electronic Coin Flip

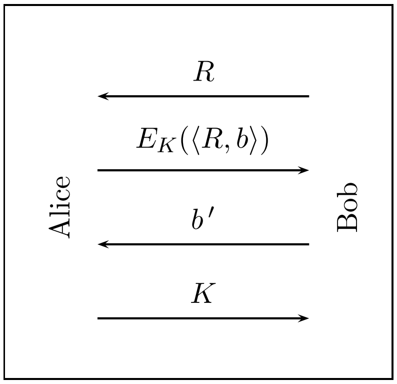
Alice and Bob are in separate locations but want to flip a coin fairly. I.e., both of Alice and Bob win/lose a flip with 50% probability. How can they do this?
Note: In this protocol, we aren't worried about Eve or Mallory.

Coin Flipping Protocol using Simple Encryption

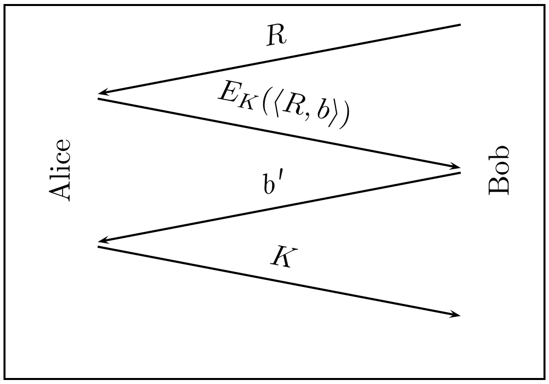
1. Bob sends nonce R to Alice.
2. Alice generates random bit b and random key K and sends $E_K(\langle R, b\rangle)$ to Bob.
3. Bob guesses that Alice's bit is b' and sends this to Alice. (He wins if $b = b'$ and loses otherwise.)
4. Alice now knows whether she won or lost; she sends K to Bob.
5. Bob calculates $D_K(E_K\langle R, b\rangle) = \langle R, b\rangle$ and now knows whether he won or lost.

Notes:

- Such protocols are often displayed graphically:

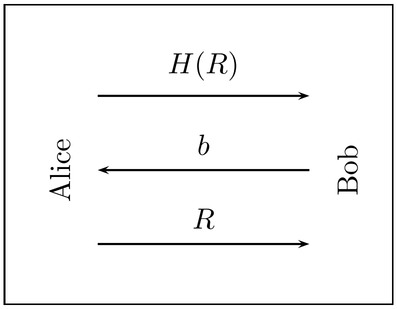


or



- Alice commits to b before Bob guesses. She cannot change her mind after Bob guesses.
- Bob does not know b choice before he guesses, but can verify b after he guesses.
- We're assuming both Alice and Bob "play by the rules". If not,
  - If Alice loses, she can (1) claim Bob sent her the wrong bit b' or (2) refuse to send him K or send him the wrong K, so he can't verify he won.
  - If Bob loses, he can (1) claim he sent the winning bit instead or (2) claim that the R he finds in the last step is not the one he sent.

Coin Flipping Protocol using Hashing

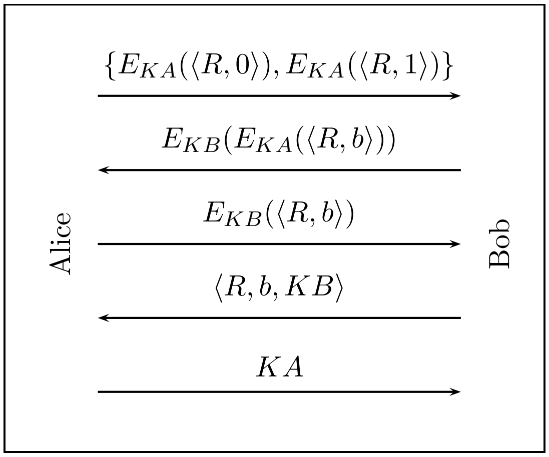


1. Alice chooses nonce R and sends $H(R) = h$ to Bob.
2. Bob guesses whether R is even or odd, and sends guess (b) to Alice.
3. Alice now knows whether she won or lost; she sends R to Bob.
4. Bob verifies $H(R) = h$ and now knows whether he won or lost.

Notes:

- This algorithm depends on R and $H(R)$ having uncorrelated even/oddness.
- Again, Alice commits to a choice (R) before Bob guesses; she cannot change her mind after Bob guesses.
- Again, Bob does not know Alice's choice before he guesses, but can verify her choice after the guess.

Coin Flipping Protocol using Commutative Key Crypto


$\{E_{KA}(\langle R,0\rangle), E_{KA}(\langle R,1\rangle)\}$
$E_{KB}(E_{KA}(\langle R,b\rangle))$
Alice
$E_{KB}(\langle R,b\rangle)$
Bob
$\langle R,b,KB\rangle$
KA

1. Alice and Bob generate new public/private key pairs KA/KA' and KB/KB', respectively.

2. Alice chooses nonce R and sends $E_{KA}(\langle R,0\rangle)$ and $E_{KA}(\langle R,1\rangle)$ to Bob in some random order. (Say $0 =$ Bob loses, $1 =$ Bob wins).

3. Bob chooses one of these two messages (call it $E_{KA}(\langle R,b\rangle)$) and sends it back to Alice as $E_{KB}(E_{KA}(\langle R,b\rangle))$.

4. Alice sends to Bob $D_{KA'}(E_{KB}(E_{KA}(\langle R,b\rangle))) = E_{KB}(\langle R,b\rangle)$ (must use a commutative encryption scheme for this to work).

5. Bob calculates $D_{KB'}(E_{KB}(\langle R,b\rangle)) = \langle R,b\rangle$ and now knows whether he won or lost. He sends $\langle R,b,KB\rangle$ to Alice.

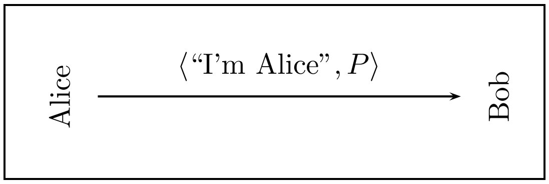
6. Alice verifies that the message she received in step 3 is $E_{KB}(E_{KA}(\langle R,b\rangle))$, and now knows whether she won or lost. She sends KA to Bob.

7. Bob verifies that he received $E_{KA}(\langle R,0\rangle)$ and $E_{KA}(\langle R,1\rangle)$ from Alice in step 2.

One-Way Authentication

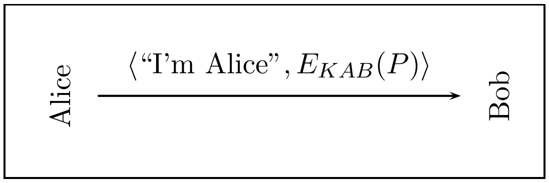
In *one-way authentication* Bob needs to be convinced that a conversation request from Alice is reall from Alice. In this protocol, we *do* need to worry about Eve and Mallory, who might try to impersonate Alice or Bob.

One-Way Authentication: Plaintext Passwords



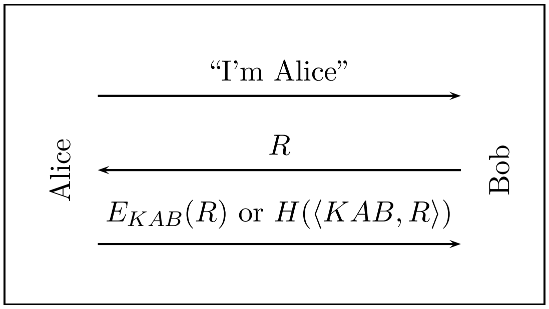
- Alice sends password P “in the clear” to Bob. (`ftp` and `telnet` actually do this!).
- *Password Sniffing*: Eve now knows Alice’s password and can later pretend to be Alice.

One-Way Authentication: Encrypted Password



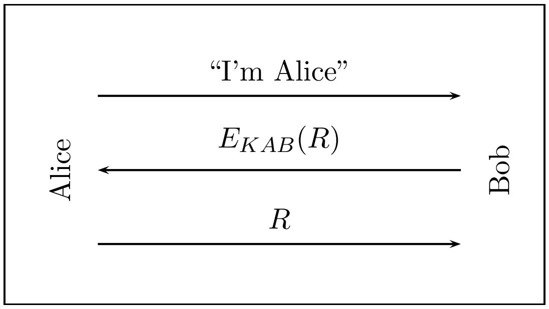
- Alice sends password P encrypted with key KAB she shares with Bob.
- *Replay Attack*: Eve can record encrypted password and later replay it.

One-Way Authentication: Challenge-Response



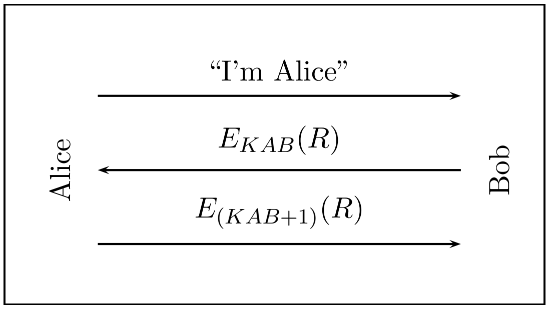
- Alice responds to nonce challenge R by encrypting or hashing R with shared key KAB.
- An improvement over simpler schemes, but subject to many attacks. (Each attack corresponds to a violated assumption.)
    - *Impersonating Bob*: Bob is not authenticated, so Mallory can impersonate him.
    - *Known-plaintext*: can help Eve find key.
    - *Hijacking*: Mallory can hijack conversation after initial handshake.
    - *Shared-Secret Vulnerability*: Mallory may be able to find KAB (and thereby impersonate Alice) by attacking Bob's database.
    - *Predictable Nonce*: If nonce is a sequence number or coarse-grained timestamp, Mallory can replace R by a "later" R', obtain $E_{KAB}(R')$ from Alice, and use this to impersonate Alice for later nonce R'. (This is impractical for fine-grained timestamp.)
- This and other examples show that while cryptography is a necessary tool, it's not enough by itself. Must consider the system in which it's used! (Radia Perlman talk: *How to Build an Insecure System out of Perfectly Good Cryptography.*)

One-Way Authentication: Encrypted-Challenge-Response



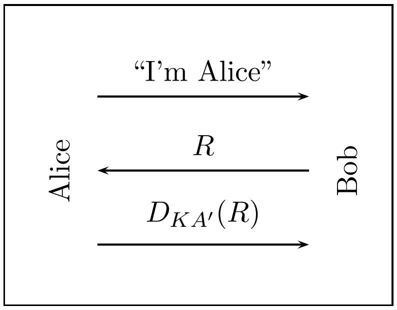
- Alice responds to nonce challenge R encrypted by Bob with shared key KAB by decrypting it.
- As with regular challenge-response, suffers from known-plaintext attack, hijacking, shared-secret vulnerability, and predictable nonce attack. Predictable nonce attack is even worse here, since Alice needn't be involved for Mallory to impersonate her by guessing "later"R.
- *Must* use invertible cryptography rather than hash.
- Advantages:
  - Mallory can't impersonate Bob if challenge R has unreplayable structure (e.g., has the form $\langle$*fine-grained timestamp*, *random number*$\rangle$) because Mallory doesn't know KAB.
  - If encrypted challenge includes fine-grained timestamp, then Bob is authenticated too!
  - Can reduce protocol to one message by having Alice send $\langle \text{``I}'\text{mAlice}'', E_{KAB}(timestamp)\rangle$. This is more efficient, but
    - Bob is no longer authenticated;
    - if timestamp is course-grained, Eve can replay message to impersonate Alice (unless Bob keeps records of timestamps already used);
    - if Mallory convinces Bob to set his clock back, can replay old messages.

One-Way Authentication: Foiling Predictable Nonce Attacks

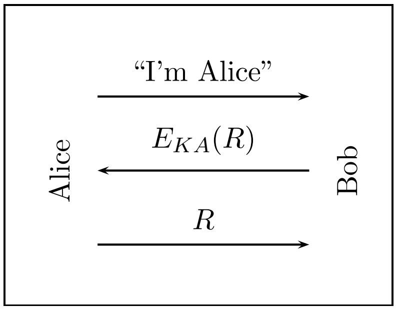


- Nonce is always sent encrypted, so it’s safe to use a predictable one.
- If nonce is unpredictable, foils known-plaintext attack.

One-Way Authentication: Public Keys Address Shared Secret Vulnerability



or

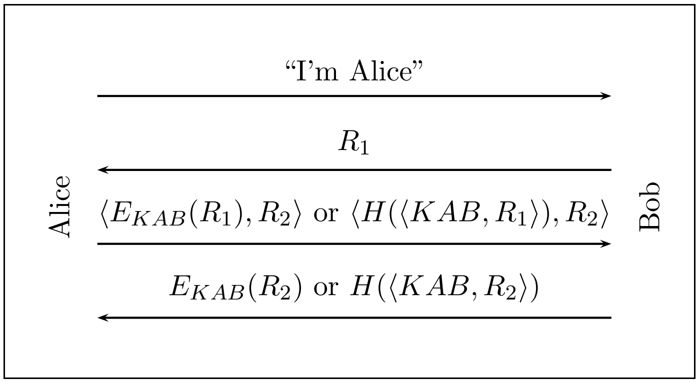


- Suppose Alice has public/private key pair KA/KA'. Then with the above protocols, no shared secrets are stored in Bob’s database.
- *Impersonation Attacks*: By impersonating Bob, Mallory can get Alice to sign or decrypt arbitrary messages! This can be avoided if protocol requires that R have structure. E.g, R might begin with “This is a challenge from Bob”.

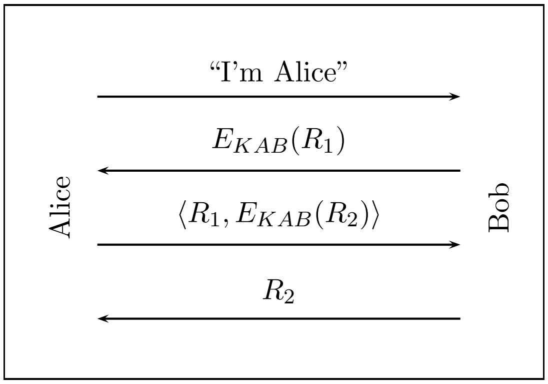
Mutual Authentication

In *mutual authentication* both Alice and Bob need to be convinced about the authenticity of the other party.

Mutual Authentication: Two One-way Authentications

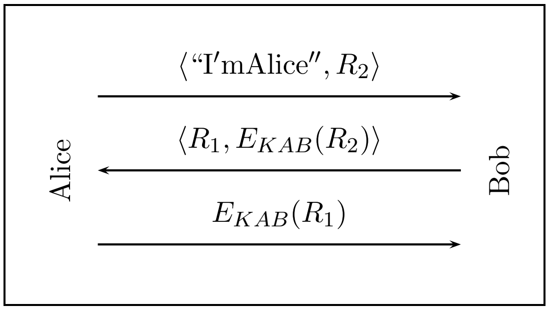

Alice
Bob
"I'm Alice"
R_1
$\langle E_{KAB}(R_1), R_2\rangle$ or $\langle H(\langle KAB, R_1\rangle), R_2\rangle$
$E_{KAB}(R_2)$ or $H(\langle KAB, R_2\rangle)$

or



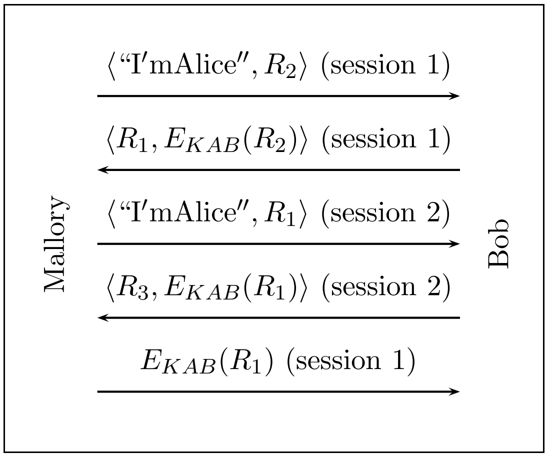
Mutual Authentication: Reflection Attack

It might seem that the mutual authentication protocol on the left could be optimized to use three messages rather than four:


Alice
$\langle$"I$'$mAlice$''$, $R_2\rangle$
$\langle R_1, E_{KAB}(R_2)\rangle$
$E_{KAB}(R_1)$
Bob

But this makes the protocol vulnerable to new attacks:

- *Chosen Plaintext Attack*: If Mallory claims to be Alice, he can get Bob to encrypt any message R_2.
- *Reflection Attack*:


Mallory
$\langle$"I$'$mAlice$''$, $R_2\rangle$ (session 1)
$\langle R_1, E_{KAB}(R_2)\rangle$ (session 1)
$\langle$"I$'$mAlice$''$, $R_1\rangle$ (session 2)
$\langle R_3, E_{KAB}(R_1)\rangle$ (session 2)
$E_{KAB}(R_1)$ (session 1)
Bob

Reflection attack can be foiled by making the protocol asymmetric:

- Alice and Bob use different authentication keys — e.g., KAB and $(KAB + 1)$.
- Alice's abd Bob's challenges look different. E.g., Alice's challenge begins "Alice's challenge" and Bob's begins "Bob's challenge".

Moral: beware protocol optimizations!