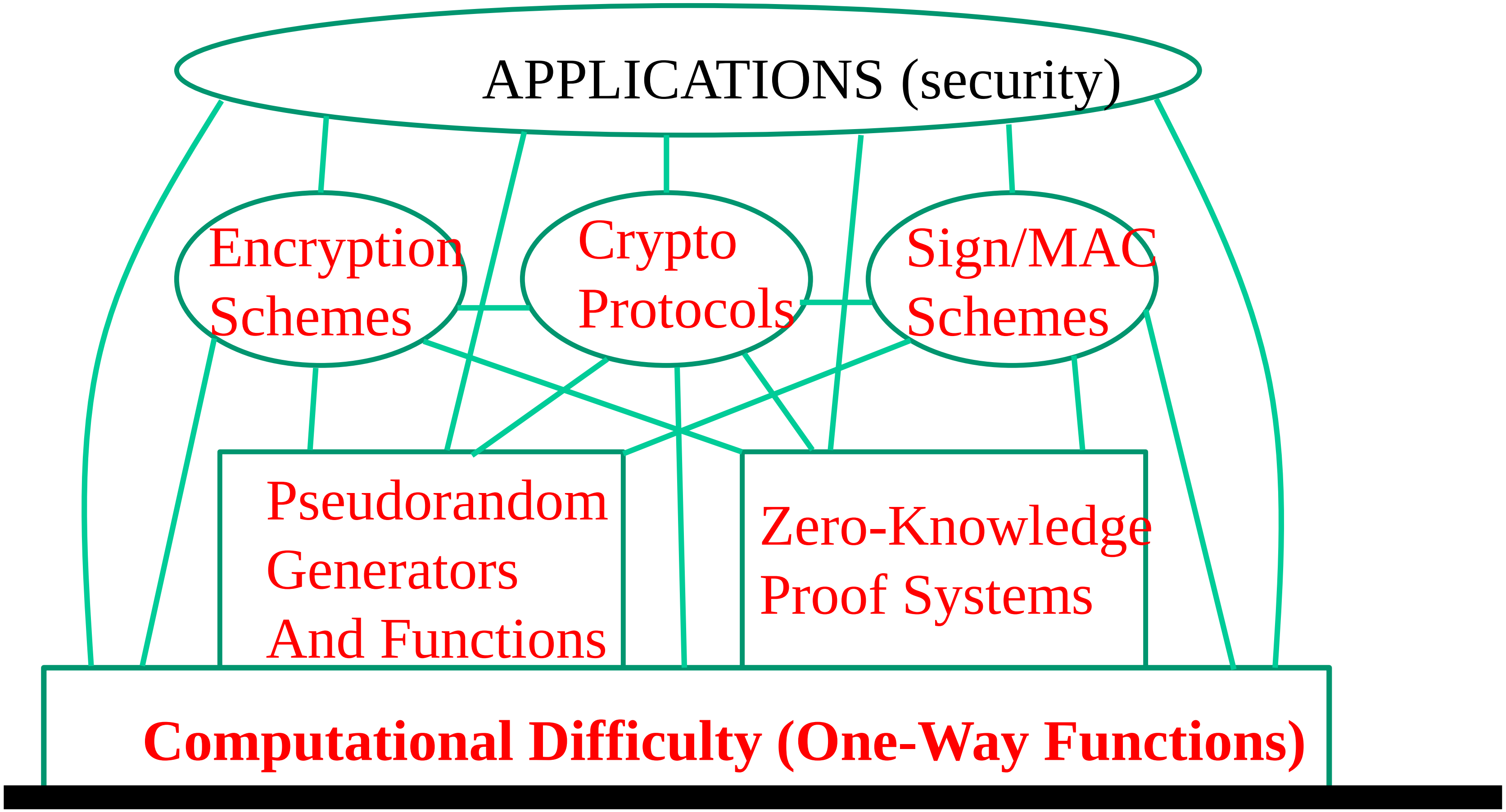
Cryptographic Protocols

CSE 5351

Spring 2017

This course:


APPLICATIONS (security)
Encryption Schemes
Crypto Protocols
Sign/MAC Schemes
Pseudorandom Generators And Functions
Zero-Knowledge Proof Systems
Computational Difficulty (One-Way Functions)

Cryptographic Protocols

- Entity Authentication
- Key Agreement
- Commitment Schemes

Entity Authentication

- Problem: Alice wants to prove to Bob that she is Alice and/or vice versa.
- Basic idea: Alice shows that she knows some secrecy which is presumably known only to Alice (and Bob).
- That secrecy could be, for example:
  - Alice's password or PIN
  - a MAC or encryption key shared by Alice and Bob, or
  - Alice's RSA private key.

Is it secure against an eavesdropper?

Protocol:

0. Alice $\rightarrow$ Bob: "I'm Alice"
1. Alice $\leftarrow$ Bob: "What's your password?"
2. Alice $\rightarrow$ Bob: Alice's password
3. Bob verifies the password

Challenge-and-response using a secrete key

Alice and Bob share a secret key k.

Protocol

(0. Alice $\to$ Bob: "I'm Alice")

1. Alice $\leftarrow$ Bob: a random challenge r.
2. Alice $\to$ Bob: $y = \mathrm{MAC}_k(r)$.
3. Bob computes $y' = \mathrm{MAC}_k(r)$ and checks if $y = y'$.

Or

Use encryption instead of MAC.

Parallel sessions attack

Alice Eve Bob

$$\xleftarrow{\qquad r \qquad}$$

$$\xrightarrow{\quad y = \mathrm{MAC}_k(r) \quad}$$

Countermeasure

Alice Eve Bob

$\xleftarrow{\quad r \quad}$

$\xrightarrow{\quad y = \mathrm{MAC}_k(r \| \mathrm{Alice}) \quad}$

Challenge-and-response using a secret key

Alice and Bob share a secret key k.

Protocol (secure):

1. Alice $\leftarrow$ Bob: a random challenge r.
2. Alice $\rightarrow$ Bob: $y = \mathrm{MAC}_k(\mathrm{ID(Alice)} \,\|\, r)$.
3. Bob computes $y' = \mathrm{MAC}_k(\mathrm{ID(Alice)} \,\|\, r)$

   and checks if $y = y'$.

Mutual authentication using a secret key

Alice and Bob share a secret key k.

Protocol

1. Alice $\leftarrow$ Bob: a random challenge r_1.
2. Alice $\rightarrow$ Bob: $y_1 = \mathrm{MAC}_k(\mathrm{ID(Alice)} \| r_1)$ and r_2.
3. Alice $\leftarrow$ Bob: $y_2 = \mathrm{MAC}_k(\mathrm{ID(Bob)} \| r_2)$.
4. Alice and Bob verify each other's response.

Man-in-the-middle attack

Alice Eve Bob

$\xleftarrow{\quad r_1 \quad}$

$\xrightarrow{\quad \mathrm{MAC}_k\,(\mathrm{A} \,\|\, r_1),\ r_2 \quad}$

$\xleftarrow{\quad \mathrm{MAC}_k\,(\mathrm{B} \,\|\, r_2) \quad}$

Countermeasure

Alice Eve Bob

$\xleftarrow{\quad r_1 \quad}$

$\xrightarrow{\quad \mathrm{MAC}_k\,(\mathrm{A} \,\|\, r_1 \|\, r_2\,),\ r_2 \quad}$

$\xleftarrow{\quad \mathrm{MAC}_k\,(\mathrm{B} \,\|\, r_2\,) \quad}$

Mutual authentication using a secret key

Alice and Bob share a secret key k.

Protocol (secure):

1. Alice $\leftarrow$ Bob: a random challenge r_1.
2. Alice $\rightarrow$ Bob: $y_1 = \mathrm{MAC}_k(\mathrm{ID}(\mathrm{Alice}) \,\|\, r_1 \,\|\, r_2)$ and r_2.
3. Alice $\leftarrow$ Bob: $y_2 = \mathrm{MAC}_k(\mathrm{ID}(\mathrm{Bob}) \,\|\, r_2)$.
4. Alice and Bob verify each other's response.

Public-key mutual authentication

Protocol (secure):

1. Alice $\leftarrow$ Bob: a random challenge r_1.
2. Alice $\rightarrow$ Bob: $y_1 = \mathrm{Sign}_{sk(\mathrm{Allice})}(\mathrm{ID(Bob)} \| r_1 \| r_2)$ and r_2.
3. Alice $\leftarrow$ Bob: $y_2 = \mathrm{Sign}_{sk(\mathrm{Bob})}(\mathrm{ID(Alice)} \| r_2)$.
4. Alice and Bob verify each other's response.

Key Agreement

Two levels of keys

- Master (long-lived) keys: (asymmetric) keys used for entity authentication and session key agreement.
- Session keys: (symmetric) keys used only for a session.

Reasons for using session keys:

1. Limiting the amount of ciphertext available to attackers.
2. Limiting the damage to only a session in case of session key compromise.
3. Symmetric encryption is faster.

Diffie-Hellman key agreement

- Alice and Bob want to set up a session key.
  1. Alice and Bob agree on a large prime p and a generator $\alpha \in \mathrm{Z}_p^*$.
  2. Alice $\rightarrow$ Bob: $\alpha^a \bmod p$, where $a \in_{\mathrm{R}} Z_{p-1}$.
  3. Alice $\leftarrow$ Bob: $\alpha^b \bmod p$ where $b \in_{\mathrm{R}} Z_{p-1}$.
  4. They agree on the key: $\alpha^{ab} \bmod p$.
- Security:
  - Provides protection against eavesdroppers.
  - Insecure against active adversaries.
  - Problem: lack of authentication.

Authentication is important in key establishment

- When establishing a session key, make sure you are doing it with the right entity.

- Two approaches:
  - Entity authentication + Diffie Hellman
  - Entity authentication + Encrypted session key

Recall: Public-key mutual authentication

Protocol:

1. Alice $\rightarrow$ Bob: a random challenge r_1.
2. Alice $\leftarrow$ Bob: $y_1 = \text{Sign}_{sk(\text{Bob})}(\text{ID(Bob)} \| r_1 \| r_2)$ and r_2.
3. Alice $\rightarrow$ Bob: $y_2 = \text{Sign}_{sk(\text{Alice})}(\text{ID(Alice)} \| r_2)$.
4. Alice and Bob verify each other's response.

Combine Diffie-Hellman with the above protocol:

- Alice uses α^a for r_1.
- Bob uses α^b for r_2.

The resulting protocol is called Station-to-Station Protocol.

Station-to-station protocol

Alice and Bob each have a signature key pair.

Protocol:

0. A and B agree on p and $\alpha \in \mathrm{Z}_p^*$ as in DH key agreement.
1. $\mathrm{A} \rightarrow \mathrm{B}$: $r_1 = \alpha^a$, where $a \in_{\mathrm{R}} Z_{p-1}$.
2. $\mathrm{A} \leftarrow \mathrm{B}$: $r_2 = \alpha^b$, $y_1 = \mathrm{Sign}_{sk(\mathrm{B})}(\mathrm{B} \| r_1 \| r_2)$, where $b \in_{\mathrm{R}} Z_{p-1}$.
3. $\mathrm{A} \rightarrow \mathrm{B}$: $y_2 = \mathrm{Sign}_{sk(\mathrm{A})}(\mathrm{A} \| r_2 \| r_1)$.
4. If all verifications pass, use $k = \alpha^{ab}$ as the session key.

Remark: all computations are done modulo p.

Public-key based authenticated key agreement

Alice and Bob each have an encryption and a signature key pair.

Protocol:

1. $A \to B$: a random challenge r_1.
2. $A \leftarrow B$: $y_1 = \text{Sign}_{\text{sk(B)}}(\text{A} \| r_1 \| r_2 \quad)$, r_2,

3. $A \to B$: $y_2 = \text{Sign}_{\text{sk(A)}}(\text{B} \| r_2)$.
4. Alice and Bob verify each other's response. If all verifications pass, Alice decrypts c to obtain k. They now can use k as the session key.

Security: this protocol provides no forward secrecy.

Forward secrecy

- Suppose Eve records all (encrypted) messages exchanged between Alice and Bob during a session. If later Eve gets Alice's decryption key d_A, she will be able to decrypt c to get the session key k.
- A session-key agreement scheme is said to provide forward secrecy if it resists this kind of attacks (i.e., session keys are secure even if master keys are later compromised.)
- Station-to-station provides forward secrecy.

Commitment Schemes

Commitment schemes

Two parties: sender S and receiver R.

Scheme:

1. Commit: S sends a message c_b, committed to a bit/value b.
2. Reveal: S sends an additional message m_b to reveal b.
3. Verify: $R(c_b, m_b) = accept$ iff the committed bit/value equals the revealed bit.

Security equirements:

1. Hiding: R cannot learn anything about b from c_b.
2. Binding: S cannot change the committed bit/value without being detected.

Hiding:

- Computationally hiding: R cannot in polynomial time
- Unconditionally hiding: R absolutely cannot

Binding:

- Computationally binding: S cannot in polynomial time
- Unconditionally binding: S absolutely cannot

An application: coin tossing by email or phone

Problem: Alice and Bob want to toss a coin by email to decide who wins.

Protocol:

1. Alice sends c_b to Bob, committed to a random bit b.
2. Bob generates a random bit b' and sends it to Alice.
3. Alice sends her committed bit b to Bob.
4. Bob verifies that $R(c_b, b) = accept,$ and both parties agree on the outcome $b \oplus b'$.

Note: if b or b' is random then $b \oplus b'$ is random.

Using symmetric encryption

Protocol:

1. Commit: To commit a value m, Alice sends $c := Enc_k(m)$ to Bob, where k is a symmetric encryption key chosen by Alice.
2. Reveal: Alice sends k to Bob.
3. Verify: Bob accepts the value $m := Dec_k(c)$.

Question: does it meet the hiding and binding requirement?

Using public-key encryption

Protocol:

1. Commit: To commit a value m, Alice generates a pair of keys (pk, sk), and sends $c := Enc_{pk}(m)$ along with pk (and system parameters) to Bob.
2. Reveal: Alice reveals m to Bob.
3. Verify: Bob accepts m if $Enc_{pk}(m) = c$.

Question: Does it meet the hiding and binding requirement?

Using a hash function H

Protocol:

1. Commit: To commit a value m, Alice sends the hash value $c := H(m \| r)$ to Bob, where r is random.
2. Reveal: Alice reveals m and r to Bob.
3. Verify: Bob accepts m if $H(m \| r) = c$.

Question: Does it meet the hiding and binding requirement?

DL-based commitment scheme

1. System setup (known to S and R):
   - $p,\ q$ large primes, with $q \mid p-1$;
   - G_q: the unique subgroup of order q of Z_p^*;
   - $g,\ h$: generators of G_q; h random;
   - $G_q = \{g^0, g^1, g^2, \ldots, g^{q-1}\} = \{h^0, h^1, h^2, \ldots, h^{q-1}\}$.
2. Commit ($S \to R$): $c = g^r h^m$, where $r \in_u Z_q$, and $m \in Z_q$ is the value being committed.
3. Reveal ($S \to R$): (r, m).
4. Verify: R accepts m if $c = g^r h^m$.

Security

1. (Unconditional) Hiding: For any m, $c = g^r h^m$ is uniformly distributed over G_q; hence, m is perfectly hidden from R.
2. (Computational) Binding: S can change her commitment iff she knows (r, m), (r', m'), $m \neq m'$, such that $g^r h^m = g^{r'} h^{m'} \Rightarrow g^{(r-r')(m'-m)^{-1}} = h$
$\Rightarrow \log_g h = (r - r')(m' - m)^{-1} \Rightarrow\!\Leftarrow$ DL assumption.

Note: computations like $g^r h^m$ are done modulo p;
exponents and logarithms are computed modulo q.

Q: What if we change the commitment to the following?

- $c := h^m$ (without using g^r)
- $c := g^{r+m}$ (namely, $g = h$)