Entity Authentication and Key Agreement

CSE 651
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 secret key

Alice and Bob share a secret key $k$.

Protocol

(0. Alice $\rightarrow$ Bob: "I'm Alice")
1. Alice $\leftarrow$ Bob: a random challenge $r$.
2. Alice $\rightarrow$ Bob: $y = \text{MAC}_k(r)$.
3. Bob computes $y' = \text{MAC}_k(r)$ and checks if $y = y'$.

Or

1. Alice $\leftarrow$ Bob: a random challenge $r$.
2. Alice $\rightarrow$ Bob: $y = E_k(r)$.
3. Bob checks if $D_k(y) = r$. 
Parallel sessions attack

Alice \quad Eve \quad Bob

\[ y = MAC_k(r) \]
**Countermeasure**

Alice \[\rightarrow\] Eve \[\rightarrow\] Bob

\[ y = \text{MAC}_k (r|\text{Alice}) \]
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: $y = \text{MAC}_k (\text{ID}(\text{Alice}) \ || \ r)$.
3. Bob computes $y' = \text{MAC}_k (\text{ID}(\text{Alice}) \ || \ r)$
   and checks if $y = y'$.

Or

1. Alice $\leftarrow$ Bob: a random challenge $r$.
2. Alice $\rightarrow$ Bob: $y = E_k (\text{ID}(\text{Alice}) \ || \ r)$.
3. Bob checks if $D_k (y) = \text{ID}(\text{Alice}) \ || \ r$. 
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: $y_1 = \text{MAC}_k(\text{ID}(\text{Alice}) \| r_1)$ and $r_2$.
3. Alice $\leftarrow$ Bob: $y_2 = \text{MAC}_k(\text{ID}(\text{Bob}) \| r_2)$.
4. Alice and Bob verify each other's response.
Man-in-the-middle attack

Alice   Eve   Bob

$\mathsf{MAC}_k(A \parallel r_1), r_2$

$\mathsf{MAC}_k(B \parallel r_2)$
Countermeasure

Alice → Eve → Bob

$\text{MAC}_k (A \parallel r_1 \parallel r_2)$, $r_2$

$\text{MAC}_k (B \parallel r_2)$
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: $y_1 = \text{MAC}_k (\text{ID}(\text{Alice}) \parallel r_1 \parallel r_2)$ and $r_2$.
3. Alice $\leftarrow$ Bob: $y_2 = \text{MAC}_k (\text{ID}(\text{Bob}) \parallel r_2)$.
4. Alice and Bob verify each other's response.
Public-key mutual authentication

Protocol (secure):

1. Alice ← Bob: a random challenge $r_1$.
2. Alice → Bob: $y_1 = \text{Sign}_{pr(\text{Alice})}(\text{ID}(\text{Bob}) \parallel r_1 \parallel r_2)$ and $r_2$.
3. Alice ← Bob: $y_2 = \text{Sign}_{pr(\text{Bob})}(\text{ID}(\text{Alice}) \parallel r_2)$.
4. Alice and Bob verify each other's response.
Key Agreement/Distribution
Two levels of keys

- **Master (long-lived) keys:** (asymmetric) keys used for entity authentication and session key agreement/distribution.
- **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.
Discrete Logarithm

- Let $p$ be a large prime number.
- $\mathbb{Z}_p^* = \{1, 2, 3, ..., p - 1\}$. $\varphi(p) = p - 1$.
- An element $\alpha \in \mathbb{Z}_p^*$ is called a generator of $\mathbb{Z}_p^*$ if $\text{ord}(\alpha) = \varphi(p)$.
- In this case, $\mathbb{Z}_p^* = \{\alpha^0, \alpha^1, \alpha^2, ..., \alpha^{p-2}\}$.
- For any $y \in \mathbb{Z}_p^*$, there is a unique exponent $x$ such that $y = \alpha^x \mod p$.
- This $x$ is called the discrete logarithm of $y$ modulo $p$, and is denoted by $\log_{\alpha} y \mod p$.
- Given $y \in \mathbb{Z}_p^*$, computing $\log_{\alpha} y \mod p$ is a hard problem.
- Diffie-Hellman algorithm is based on the hardness of this problem.
Example


- $\mathbb{Z}_p^* = \{1, 2, 3, ..., 12\}$. $\alpha = 2$ is a generator.

  \[
  \begin{align*}
  x &:\quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \\
  2^x &:\quad 1 \quad 2 \quad 4 \quad 8 \quad 3 \quad 6 \quad 12 \quad 11 \quad 9 \quad 5 \quad 10 \quad 7 \quad 1
  \end{align*}
  \]

- Given $2^x = 11$, what is the value of $x$?

- That is, $\log_2 11 \mod 13 = ?$

- There is no efficient algorithm for this.
**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 $\alpha \in \mathbb{Z}_p^*$.
  2. Alice $\rightarrow$ Bob: $\alpha^a \mod p$, where $a \in \mathbb{R} \mathbb{Z}_{p-1}$.
  3. Alice $\leftarrow$ Bob: $\alpha^b \mod p$ where $b \in \mathbb{R} \mathbb{Z}_{p-1}$.
  4. They agree on the key: $\alpha^{ab} \mod p$.

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

Combine Diffie-Hellman with this protocol

- Alice uses $\alpha^a$ for $r_1$.
- Bob uses $\alpha^b$ 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 \( \alpha \in \mathbb{Z}_p^* \) as in DH key agreement.

1. A \( \rightarrow \) B: \( r_1 = \alpha^a \), where \( a \in \mathbb{Z}_{p-1}^\ast \).

2. A \( \leftarrow \) B: \( r_2 = \alpha^b \), \( y_1 = \text{Sign}_{pr(B)}(B \parallel r_1 \parallel r_2) \), where \( b \in \mathbb{Z}_{p-1}^\ast \).

3. A \( \rightarrow \) B: \( y_2 = \text{Sign}_{pr(A)}(A \parallel r_2 \parallel r_1) \).

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 → B: a random challenge $r_1$.
2. A ← B: $y_1 = \text{Sign}_{pr(B)}(A \parallel r_1 \parallel r_2)$, $r_2$.

3. A → B: $y_2 = \text{Sign}_{pr(A)}(B \parallel r_2)$.
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.
Public-key management

- To verify a signature, you need to know the signer's public key.
- A common practice is to include a certificate with the signature.
- A certificate is issued by a certification authority, and contains:
  - certificate owner's name/identifier
  - certificate issuer's name/identifier
  - period of validity
  - certificate owner's public key information:
    - public key, algorithm identifier (e.g., RSA), associated parameters
  - issuer's signature algorithm identifier, including associated parameters
  - certificate issuer's signature
  - other information
Obtaining a certificate

- There are commercial certification authorities.
- Just google "certificate authority" and you will find many websites where you can purchase a certificate online.

- Browsers are equipped with a list of certificate authorities, and can verify certificates signed by them.