Arrangement for the following weeks

- I will be away, back on Tuesday 23 October 2018
- Dr Fang will lecture the topic **malicious software** on 25 September 2018 (next week)
- There will be **no lectures on 9 October and 16 October**
- By the end of the week I will publish spec for Assignment one (10%) on `https://github.com/czhang-fm/computer-security-JNU`

Key Distribution

If a given cipher has the following property:

$$E_{K_A}(E_{K_B}(M)) = E_{K_B}(E_{K_A}(M))$$

We are able to use the property for key distribution.

Key Distribution

A generates a shared key K

$A \longrightarrow B : E_{K_A}(K)$

$B \longrightarrow A : E_{K_B}(E_{K_A}(K))$

A decrypts with K_A, by computing
$D_{K_A}(E_{K_B}(E_{K_A}(K))) = D_{K_A}(E_{K_A}(E_{K_B}(K))) = E_{K_B}(K)$

$A \longrightarrow B : E_{K_B}(K)$

B computes $D_{K_B}(E_{K_B}(K)) = K$.

Key Distribution

If a given cipher has the following property:

$$E_{K_A}(E_{K_B}(M)) = E_{K_B}(E_{K_A}(M))$$

However, very few good ciphers satisfy the above property

Key Distribution

If a given cipher has the following property:

$$E_{K_A}(E_{K_B}(M)) = E_{K_B}(E_{K_A}(M))$$

However, very few good ciphers satisfy the above property

- Review the commutative property of known ciphers
- Caesar's cipher, substitution cipher, Viginère cipher?

Modular Arithmetic

- $x \equiv y \; mod \; n$ if there exists k such that $x = k \times n + y$

  I.e., x and y have the same remainder if divided by n
- For every integer x, there exists $m \in \{0, 1, 2, \ldots n-1\}$ such that $x \equiv m \; mod \; n$

  I.e., $x \; mod \; n = m$.

Modular Arithmetic

- $[(x \ mod \ n) + (y \ mod \ n)] \ mod \ n = (x + y) \ mod \ n$
- $[(x \ mod \ n) - (y \ mod \ n)] \ mod \ n = (x - y) \ mod \ n$
- $[(x \ mod \ n) \times (y \ mod \ n)] \ mod \ n = (x \times y) \ mod \ n$
- $(x \ mod \ n)^k \ mod \ n = (x^k) \ mod \ n$

Modular Arithmetic

- $[(x \ mod \ n) + (y \ mod \ n)] \ mod \ n = (x + y) \ mod \ n$
- $[(x \ mod \ n) - (y \ mod \ n)] \ mod \ n = (x - y) \ mod \ n$
- $[(x \ mod \ n) \times (y \ mod \ n)] \ mod \ n = (x \times y) \ mod \ n$
- $(x \ mod \ n)^k \ mod \ n = (x^k) \ mod \ n$

Tutorial: prove these laws

Discrete Logarithm

- Let n be a large prime

  g is a **generator** modulo n, if
  $g \ mod \ n$, $g^2 \ mod \ n$, $\dots$, $g^{n-1} \ mod \ n$
  enumerates all members in $\{1, 2, 3, \dots n-1\}$
- Given n prime, a generator modulo n always exists
- Let n be 5, then 3 is a generator modulo n
  - $3 \ mod \ 5 = 3$, $3^2 \ mod \ 5 = 4$, $3^3 \ mod \ 5 = 2$, $3^4 \ mod \ 5 = 1$

Discrete Logarithm

- Let n be a large prime

  g is a **generator** modulo n, if
  $g \ mod \ n, \ g^2 \ mod \ n, \ \ldots, \ g^{n-1} \ mod \ n$
  enumerates all members in $\{1, 2, 3, \ldots n-1\}$
- Given n prime, a generator modulo n always exists
- Let n be 5, then 3 is a generator modulo n
  - $3 \ mod \ 5 = 3$, $3^2 \ mod \ 5 = 4$, $3^3 \ mod \ 5 = 2$, $3^4 \ mod \ 5 = 1$
- **Discrete Logarithm Problem**: From $g^i \ mod \ n$, compute i

  This is a hard problem

Diffie-Hellman key exchange

- A and B agree on a large prime q and generator a, presumably known by everyone else (including attackers)

  A generates random value X_A

  B generates random value X_B

Diffie-Hellman key exchange

- A and B agree on a large prime q and generator a, presumably known by everyone else (including attackers)

  A generates random value X_A

  B generates random value X_B
- $A \longrightarrow B : a^{X_A}\ mod\ q$
- $B \longrightarrow A : a^{X_B}\ mod\ q$

Diffie-Hellman key exchange

- A and B agree on a large prime q and generator a, presumably known by everyone else (including attackers)

  A generates random value X_A

  B generates random value X_B
- $A \longrightarrow B : a^{X_A} \ mod \ q$
- $B \longrightarrow A : a^{X_B} \ mod \ q$
- A computes $(a^{X_B} \ mod \ q)^{X_A} \ mod \ q = a^{X_A X_B} \ mod \ q$

  B computes $(a^{X_A} \ mod \ q)^{X_B} \ mod \ q = a^{X_A X_B} \ mod \ q$

Diffie-Hellman key exchange

- A and B agree on a large prime q and generator a, presumably known by everyone else (including attackers)

  A generates random value X_A

  B generates random value X_B
- $A \longrightarrow B : a^{X_A}\ mod\ q$
- $B \longrightarrow A : a^{X_B}\ mod\ q$
- A computes $(a^{X_B}\ mod\ q)^{X_A}\ mod\ q = a^{X_A X_B}\ mod\ q$

  B computes $(a^{X_A}\ mod\ q)^{X_B}\ mod\ q = a^{X_A X_B}\ mod\ q$

Security of the protocol depends on the hardness of discrete logarithm — from $a^{X_A}\ mod\ q$, it is hard to compute X_A.

Public Key Cryptography

A public key cryptosystem is

- a pair of keys, K (public key) and K^{-1} (private key)
- a pair of efficiently computable functions E and S

Public Key Cryptography

A public key cryptosystem is

- a pair of keys, K (public key) and K^{-1} (private key)
- a pair of efficiently computable functions E and S
- given a message M, we have
  - $S(E(M, K), K^{-1}) = M$,
    (**encryption**, **confidentiality**)
  - $E(S(M, K^{-1}), K) = M$,
    (used for **digital signature**, **authenticity**, **nonrepudiation**)

Public Key Cryptography

A public key cryptosystem is

- a pair of keys, K (public key) and K^{-1} (private key)
- a pair of efficiently computable functions E and S
- given a message M, we have
  - $S(E(M, K), K^{-1}) = M$,
    (**encryption**, **confidentiality**)
  - $E(S(M, K^{-1}), K) = M$,
    (used for **digital signature**, **authenticity**, **nonrepudiation**)
- from K, it's hard to compute K^{-1}

Public Key Cryptography

Alice wants to send a (short) secret message to Bob

Public Key Cryptography

Alice wants to send a (short) secret message to Bob

Bob generates K_B and K_B^{-1} and publishes K_B

$A \longrightarrow B : E(M, K_B)$

Bob decrypts the message by computing $S(E(M, K_B), K_B^{-1}) = M$

Public Key Cryptography

Alice wants to send a (short) secret message to Bob

Bob generates K_B and K_B^{-1} and publishes K_B

$A \longrightarrow B : E(M, K_B)$

Bob decrypts the message by computing $S(E(M, K_B), K_B^{-1}) = M$

- Eve, the adversary, cannot easily compute M without Bob's private key K_B^{-1}

Public Key Cryptography

Alice wants to send a (short) secret message to Bob

Bob generates K_B and K_B^{-1} and publishes K_B

$A \longrightarrow B : E(M, K_B)$

Bob decrypts the message by computing $S(E(M, K_B), K_B^{-1}) = M$

- Eve, the adversary, cannot easily compute M without Bob's private key K_B^{-1}

Comparing to symmetric key encryption/decryption, public key cryptography takes longer to compute, and usually with longer keys. E.g., RSA uses 1024 to 4096 bits of key length, comparing to DES (56 bits) and AES (128-256 bits).

Key Distribution in Public Key Cryptography

1. Alice generate a secret key K to be shared with Bob
2. $A \longrightarrow B : E(\text{“}the\ shared\ key\ is\ K", K_B)$
3. $A \longrightarrow B : E_K(M)$

Key Distribution in Public Key Cryptography

1. Alice generate a secret key K to be shared with Bob
2. $A \longrightarrow B : E(\text{"}\textit{the shared key is } K\text{"}, K_B)$
3. $A \longrightarrow B : E_K(M)$

- Does not require two parties to have a shared key
- K can be randomly generated
- The first step requires public key cryptography, but the first message is short
- The longer message M is encrypted/decrypted using faster symmetric cipher with shared key K

Key Distribution in Public Key Cryptography

1. Alice generate a secret key K to be shared with Bob
2. $A \longrightarrow B : E(\text{``}\textit{the shared key is } K\text{''}, K_B)$
3. $A \longrightarrow B : E_K(M)$

- Does not require two parties to have a shared key
- K can be randomly generated
- The first step requires public key cryptography, but the first message is short
- The longer message M is encrypted/decrypted using faster symmetric cipher with shared key K
- How is Alice sure that K_B belongs to Bob?
- How is Bob sure that M is from Alice?

Combining Signatures with Encryption

Alice wants to send a signed message to Bob, but doesn't want Eve to learn the contents:

1. $A \rightarrow B : E(S(M, K_A^{-1}), K_B)$
2. B computes $S(E(S(M, K_A^{-1}), K_B), K_B^{-1} = S(M, K_A^{-1})$
3. B then verifies $E(S(M, K_A^{-1}), K_A) = M$,
   B then knows M is from A

Combining Signatures with Encryption

Why not encrypt then sign?
I.e., $A \rightarrow B : S(E(M, K_B), K_A^{-1})$

Combining Signatures with Encryption

Why not encrypt then sign?
I.e., $A \rightarrow B : S(E(M, K_B), K_A^{-1})$

- If Bob wants to convince someone Alice signed M, he would have to reveal K_B^{-1}
- Signature on text not comprehensive to the signer might not be legally valid.
- If Eve intercepts the message, Eve can extract $E(M, K_B)$, then send $S(E(M, K_B), K_{Eve}^{-1})$ and claim the message is from her

Combining Signatures with Encryption

In practice, digital signatures are always applied on hashed consumption

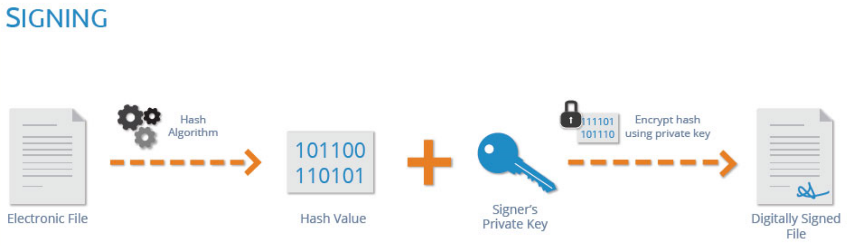
One way hash functions

- A hash function h can be applied to a block of data of any size and produces a fixed-length output.
  I.e., $h : \Sigma^* \rightarrow \Sigma^n$ with fixed n bits.
- h is *one-way* or *pre-image resistant* if given $h(M)$, it is hard to find value M
- h is *weak collision resistant* if it is hard to find some y with $h(y) = h(x)$
- h is *strong collision-resistant* if it is hard to find two different messages M_1 and M_2 such that $h(M_1) = h(M_2)$.

One way hash functions

- A hash function h can be applied to a block of data of any size and produces a fixed-length output.
  I.e., $h : \Sigma^* \rightarrow \Sigma^n$ with fixed n bits.
- h is *one-way* or *pre-image resistant* if given $h(M)$, it is hard to find value M
- h is *weak collision resistant* if it is hard to find some y with $h(y) = h(x)$
- h is *strong collision-resistant* if it is hard to find two different messages M_1 and M_2 such that $h(M_1) = h(M_2)$.
- Examples
  - MD2, MD4, MD5 of 128 bits by Ron Rivest, MIT
  - SHA-1 (160 bits), SHA-2 (SHA-256 and SHA-512), SHA-3
  - hashes derived from Block ciphers such as DES and RSA

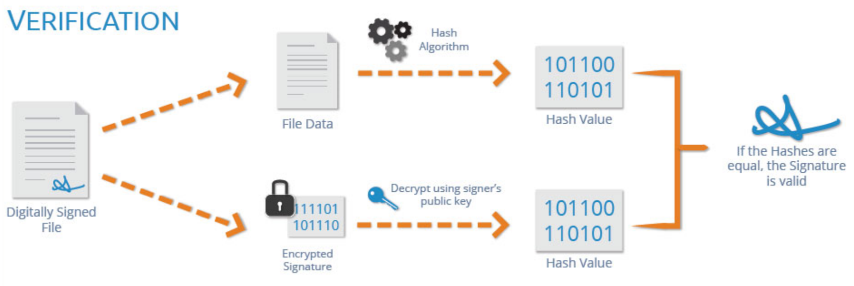
Use of hash functions in Digital Signatures



Alice's signature on M is

$$sig_A(M) = (M, S(h(M), K_A^{-1})) = (M, sig_{M,A})$$

Use of hash functions in Digital Signatures



I.e., to check if the following is true

$$h(M) == E(sig_{M,A}, K_A)$$

Security of Digital Signatures

Recall

$$sig_A(M) = (M, S(h(M), K_A^{-1}))$$

- **Authentication, unforgeability**: the adversary doesn't know K_A^{-1}, and cannot forge $S(h(M), K_A^{-1})$
- **Integrity**: it is hard to compute $S(h(M'), K_A^{-1})$ from $S(h(M), K_A^{-1})$ with $M \neq M'$
- **Nonrepudiation**: $S(h(M), K_A^{-1})$ can only be made by Alice

Why digital signatures are applied on hashed values

Why digital signatures are applied on hashed values

- Public key encryption/decryption is usually much slower than hash computation.

Why digital signatures are applied on hashed values

- Public key encryption/decryption is usually much slower than hash computation.
- Reordering attack: Most public key are block ciphers. If a message M is too long, one will have to break $M = M_1 M_2 \ldots$, and produces $Sig_1, Sig_2, \ldots$. An attacker can swap or drop some of chunks.

Statistics of Birthdays (today's quiz)

Q1 How many people need to be at a party for the probability that at least one of them has the same birthday as you to be $> \frac{1}{2}$?

Q2 How many people need to be at a party for there to be at least two with the same birthday?

Statistics of Birthdays (today's quiz)

Q1 How many people need to be at a party for the probability that at least one of them has the same birthday as you to be $> \frac{1}{2}$?

Q2 How many people need to be at a party for there to be at least two with the same birthday?

Answer to Q1: 253

Statistics of Birthdays (today's quiz)

Q1 How many people need to be at a party for the probability that at least one of them has the same birthday as you to be $> \frac{1}{2}$?

Q2 How many people need to be at a party for there to be at least two with the same birthday?

Answer to Q1: 253

Answer to Q2: 23

Statistics of Birthdays (today's quiz)

Q1 How many people need to be at a party for the probability that at least one of them has the same birthday as you to be $> \frac{1}{2}$?

Q2 How many people need to be at a party for there to be at least two with the same birthday?

Answer to Q1: 253

Answer to Q2: 23

For a hash function, *strong collision-resistant* is usually harder to achieve than *weak collision-resistant*

Birthday Attack on Hashed Signature

1. Bob prepare two contracts
   - Contract1 "Bob will pay Alice \$10"
   - Contract2 "Alice will pay Bob \$10,000"
2. Bob prepares a large number of Contract1 and Contract2, differing only with respect to spacing
3. Bob hashes all these versions, looking for a version V_1 of Contract1 and a versin V_2 of Contract2, such that $h(V_1) = h(V_2)$
4. Bob asks Alice to sign V_1

Birthday Attack on Hashed Signature

1. Bob prepare two contracts
   - Contract1 "Bob will pay Alice $10"
   - Contract2 "Alice will pay Bob $10,000"
2. Bob prepares a large number of Contract1 and Contract2, differing only with respect to spacing
3. Bob hashes all these versions, looking for a version V_1 of Contract1 and a versin V_2 of Contract2, such that $h(V_1) = h(V_2)$
4. Bob asks Alice to sign V_1

Moral: if someone asks you to sign a contract, add some random spaces before you do so

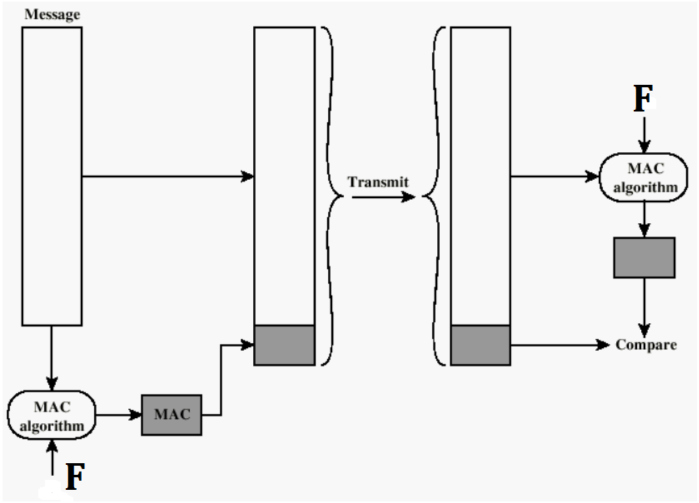
To wrap up the application of public key cryptography

- A pair of keys K_A and K_A^{-1}, where K_A is made public
- Confidentiality: others use K_A to encrypt, Alice uses K_A^{-1} to decrypt, which protects against passive attack (eavesdropping)
- Integrity and authentication: provided by the use of K_A^{-1} for making a digital signature, which protects against the forge of a message (it's hard to forge a $S(M', K_A^{-1})$ given $S(M, K_A^{-1})$)
- Hash function improves efficiency, avoid certain attacks, while also provide integrity (hard to find M' such that $h(M) = h(M')$)
- In a protocol, public key cryptography can be used to establish a one-time session (symmetric) key

Quiz: Message Authentication

Alice and Bob are physically separated and only connected via the Internet. Enumerate all possible ways for Alice to convince Bob that a message M is from her.

Message Authentication Code



- Given M, compute $F(M)$ of fixed length, output $M||F(M)$
- The mechanism similar to Cyclic Redundancy Check (CRC)

Message Authentication Code

- MAC with a symmetric cipher with shared key K: given $M = M_1 M_2 \ldots M_n$, use the last block $E_K(M_n)$ in CBC (Cipher Block Chaining) mode.
- MAC with a secure hash function h: use $h(M)$.

Message Authentication Code

- MAC with a symmetric cipher with shared key K: given $M = M_1 M_2 \dots M_n$, use the last block $E_K(M_n)$ in CBC (Cipher Block Chaining) mode.
- MAC with a secure hash function h: use $h(M)$.

How are desired security properties (authenticity) guaranteed?

Message Authentication Code

- MAC with a symmetric cipher with shared key K: given $M = M_1 M_2 \ldots M_n$, use the last block $E_K(M_n)$ in CBC (Cipher Block Chaining) mode.
- MAC with a secure hash function h: use $h(M)$.

How are desired security properties (authenticity) guaranteed?

- MAC with a symmetric block cipher: rely on the strength of the cipher as well as shared keys
- MAC with a secure hash function h: rely on strong collision resistance

Lamport Signature

- Generate 256 pairs of random numbers, each number being 256 bits in size, that is, a total of $2 \times 256 \times 256$ bits in total, stored as a two dimensional array $S[0][0]$, $S[0][1], \ldots$, $S[0][255]$, $S[1][0]$, $S[1][1]$, $\ldots$, $S[0][255]$. This is kept secret as the private key.
- The public key is the collection of the hashed values $h(S[0][0])$, $h(S[0][1]), \ldots$, which is also stored as a two dimensional array $P[0][0]$, $P[0][1], \ldots$, $P[0][255]$, $P[1][0]$, $P[1][1]$, $\ldots$, $P[0][255]$.

Lamport Signature

- Generate 256 pairs of random numbers, each number being 256 bits in size, that is, a total of $2 \times 256 \times 256$ bits in total, stored as a two dimensional array $S[0][0]$, $S[0][1], \ldots$, $S[0][255]$, $S[1][0]$, $S[1][1]$, $\ldots$, $S[0][255]$. This is kept secret as the private key.
- The public key is the collection of the hashed values $h(S[0][0])$, $h(S[0][1]), \ldots$, which is also stored as a two dimensional array $P[0][0]$, $P[0][1], \ldots$, $P[0][255]$, $P[1][0]$, $P[1][1]$, $\ldots$, $P[0][255]$.
- To sign a message M, we first hash the message to a 256-bit hash sum $h(M)$. Then, for each bit in the hash, based on the value of the bit, pick one number from the corresponding pairs of numbers that comprise the private key (i.e., if the bit is 0, the first number is chosen, and if the bit is 1, the second is chosen). This produces a sequence of 256 random numbers. As each number is itself 256 bits long the total size of the signature will be $8KB$.

Lamport Signature

- Generate 256 pairs of random numbers, each number being 256 bits in size, that is, a total of $2 \times 256 \times 256$ bits in total, stored as a two dimensional array $S[0][0]$, $S[0][1], \dots$, $S[0][255]$, $S[1][0]$, $S[1][1]$, $\dots$, $S[0][255]$. This is kept secret as the private key.
- The public key is the collection of the hashed values $h(S[0][0])$, $h(S[0][1]), \dots$, which is also stored as a two dimensional array $P[0][0]$, $P[0][1], \dots$, $P[0][255]$, $P[1][0]$, $P[1][1]$, $\dots$, $P[0][255]$.
- To sign a message M, we first hash the message to a 256-bit hash sum $h(M)$. Then, for each bit in the hash, based on the value of the bit, pick one number from the corresponding pairs of numbers that comprise the private key (i.e., if the bit is 0, the first number is chosen, and if the bit is 1, the second is chosen). This produces a sequence of 256 random numbers. As each number is itself 256 bits long the total size of the signature will be $8KB$.
- This is a one-time signature scheme. How it works and why the level of security decreases if it is used for the second time and the third time?