CS641

Modern Cryptology

Lecture 17

Outline

1. Hashing

2. Public Key Infrastructure (PKI)

Problems with Digital Signatures

- Suppose the document to be signed is very long, and so we need to split it into k blocks $m = m_1 m_2 \cdots m_k$.
- Each block is signed separately, with signature s_i associated with block m_i.
- Ela can then take two such signed documents and do cut-and-paste to create signatures for a third document.
- In addition, signing multiple blocks consumes a lot of time as well.
- For these reasons, one would like to ideally sign only one block per document.

Problems with Digital Signatures

- Suppose the document to be signed is very long, and so we need to split it into k blocks $m = m_1 m_2 \cdots m_k$.
- Each block is signed separately, with signature s_i associated with block m_i.
- Ela can then take two such signed documents and do cut-and-paste to create signatures for a third document.
- In addition, signing multiple blocks consumes a lot of time as well.
- For these reasons, one would like to ideally sign only one block per document.

Problems with Digital Signatures

- Suppose the document to be signed is very long, and so we need to split it into k blocks $m = m_1 m_2 \cdots m_k$.
- Each block is signed separately, with signature s_i associated with block m_i.
- Ela can then take two such signed documents and do cut-and-paste to create signatures for a third document.
- In addition, signing multiple blocks consumes a lot of time as well.
- For these reasons, one would like to ideally sign only one block per document.

Problems with Digital Signatures

- Suppose the document to be signed is very long, and so we need to split it into k blocks $m = m_1 m_2 \cdots m_k$.
- Each block is signed separately, with signature s_i associated with block m_i.
- Ela can then take two such signed documents and do cut-and-paste to create signatures for a third document.
- In addition, signing multiple blocks consumes a lot of time as well.
- For these reasons, one would like to ideally sign only one block per document.

Problems with Digital Signatures

- Suppose the document to be signed is very long, and so we need to split it into k blocks $m = m_1 m_2 \cdots m_k$.
- Each block is signed separately, with signature s_i associated with block m_i.
- Ela can then take two such signed documents and do cut-and-paste to create signatures for a third document.
- In addition, signing multiple blocks consumes a lot of time as well.
- For these reasons, one would like to ideally sign only one block per document.

Hashing

- We need a function $h : \{0,1\}^* \mapsto \{0,1\}^\ell$ such that h maps two distinct documents to distinct strings of length ℓ with ℓ less than size of one block.
- This is impossible since there can be infinitely many documents but there are only 2^ℓ strings of length ℓ.
- If h is such that finding two documents that map to same output is hard, it can still work:
  - Since it is hard to find m and m' such that $h(m) = h(m')$, one would not encounter two such documents!
- Such functions are called hash functions.

Hashing

- We need a function $h : \{0,1\}^* \mapsto \{0,1\}^\ell$ such that h maps two distinct documents to distinct strings of length ℓ with ℓ less than size of one block.
- This is impossible since there can be infinitely many documents but there are only 2^ℓ strings of length ℓ.
- If h is such that finding two documents that map to same output is hard, it can still work:
  - Since it is hard to find m and m' such that $h(m) = h(m')$, one would not encounter two such documents!
- Such functions are called hash functions.

HASHING

- We need a function $h : \{0,1\}^* \mapsto \{0,1\}^\ell$ such that h maps two distinct documents to distinct strings of length ℓ with ℓ less than size of one block.
- This is impossible since there can be infinitely many documents but there are only 2^ℓ strings of length ℓ.
- If h is such that finding two documents that map to same output is hard, it can still work:
  - Since it is hard to find m and m' such that $h(m) = h(m')$, one would not encounter two such documents!
- Such functions are called hash functions.

Hashing

- We need a function $h : \{0,1\}^* \mapsto \{0,1\}^\ell$ such that h maps two distinct documents to distinct strings of length ℓ with ℓ less than size of one block.
- This is impossible since there can be infinitely many documents but there are only 2^ℓ strings of length ℓ.
- If h is such that finding two documents that map to same output is hard, it can still work:
  - Since it is hard to find m and m' such that $h(m) = h(m')$, one would not encounter two such documents!
- Such functions are called hash functions.

Hashing

- We need a function $h : \{0,1\}^* \mapsto \{0,1\}^\ell$ such that h maps two distinct documents to distinct strings of length ℓ with ℓ less than size of one block.
- This is impossible since there can be infinitely many documents but there are only 2^ℓ strings of length ℓ.
- If h is such that finding two documents that map to same output is hard, it can still work:
  - Since it is hard to find m and m' such that $h(m) = h(m')$, one would not encounter two such documents!
- Such functions are called hash functions.

Cryptographically Secure Hash Functions

Function h is a cryptographically secure hash function if h is easy-to-compute and following are hard:

1. Given m, find $m' \neq m$ such that $h(m) = h(m')$.
2. Given w, find m such that $h(m) = w$.
3. Find m and m' such that $h(m) = h(m')$.

- Third property is required to avoid the case when a signed document can be replaced by another one.
- Second property is useful in other applications.

Cryptographically Secure Hash Functions

Function h is a cryptographically secure hash function if h is easy-to-compute and following are hard:

1. Given m, find $m' \neq m$ such that $h(m) = h(m')$.
2. Given w, find m such that $h(m) = w$.
3. Find m and m' such that $h(m) = h(m')$.

- Third property is required to avoid the case when a signed document can be replaced by another one.
- Second property is useful in other applications.

Cryptographically Secure Hash Functions

Function h is a cryptographically secure hash function if h is easy-to-compute and following are hard:

1. Given m, find $m' \neq m$ such that $h(m) = h(m')$.
2. Given w, find m such that $h(m) = w$.
3. Find m and m' such that $h(m) = h(m')$.

- Third property is required to avoid the case when a signed document can be replaced by another one.
- Second property is useful in other applications.

Cryptographically Secure Hash Functions

Function h is a cryptographically secure hash function if h is easy-to-compute and following are hard:

1. Given m, find $m' \neq m$ such that $h(m) = h(m')$.
2. Given w, find m such that $h(m) = w$.
3. Find m and m' such that $h(m) = h(m')$.

- Third property is required to avoid the case when a signed document can be replaced by another one.
- Second property is useful in other applications.

Cryptographically Secure Hash Functions

Function h is a cryptographically secure hash function if h is easy-to-compute and following are hard:

1. Given m, find $m' \neq m$ such that $h(m) = h(m')$.
2. Given w, find m such that $h(m) = w$.
3. Find m and m' such that $h(m) = h(m')$.

- Third property is required to avoid the case when a signed document can be replaced by another one.
- Second property is useful in other applications.

Cryptographically Secure Hash Functions

Function h is a cryptographically secure hash function if h is easy-to-compute and following are hard:

1. Given m, find $m' \neq m$ such that $h(m) = h(m')$.
2. Given w, find m such that $h(m) = w$.
3. Find m and m' such that $h(m) = h(m')$.

- Third property is required to avoid the case when a signed document can be replaced by another one.
- Second property is useful in other applications.

Digital Signature via RSA and Hashing

- Anubha announces her public key (e, n) and she has corresponding private key d.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< n$.
- Signing: Anubha computes $s = h(m)^d \pmod{n}$.
- Verification: Given (m, s), Braj checks if $s = h(m)^e \pmod{n}$.
- Hardness of forgery follows as before and using the properties of h.

Digital Signature via RSA and Hashing

- Anubha announces her public key (e, n) and she has corresponding private key d.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< n$.
- Signing: Anubha computes $s = h(m)^d \pmod{n}$.
- Verification: Given (m, s), Braj checks if $s = h(m)^e \pmod{n}$.
- Hardness of forgery follows as before and using the properties of h.

Digital Signature via RSA and Hashing

- Anubha announces her public key (e, n) and she has corresponding private key d.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< n$.
- Signing: Anubha computes $s = h(m)^d \pmod{n}$.
- Verification: Given (m, s), Braj checks if $s = h(m)^e \pmod{n}$.
- Hardness of forgery follows as before and using the properties of h.

Digital Signature via RSA and Hashing

- Anubha announces her public key (e, n) and she has corresponding private key d.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< n$.
- Signing: Anubha computes $s = h(m)^d \pmod{n}$.
- Verification: Given (m, s), Braj checks if $s = h(m)^e \pmod{n}$.
- Hardness of forgery follows as before and using the properties of h.

Digital Signature via RSA and Hashing

- Anubha announces her public key (e, n) and she has corresponding private key d.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< n$.
- Signing: Anubha computes $s = h(m)^d \pmod{n}$.
- Verification: Given (m, s), Braj checks if $s = h(m)^e \pmod{n}$.
- Hardness of forgery follows as before and using the properties of h.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

Digital Signature via ECC and Hashing

- Anubha announces her public key (C, p, P, eP, t) and she has $t - e$ as private key.
  - g is an element of order t in the group and t is a prime number.
- Assume that a cryptographycally secure hash function h is available such that its output can be viewed as a number $< t$.
- Signing:
  - Anubha picks a random r, $1 < r < t$, and computes $rP = (a, b)$.
  - She computes $s = r^{-1}(h(m) + ae) \pmod{t}$.
  - Signature of document m is the pair (a, s).
- Verification:
  - Given document m and signature (a, s), Braj first computes $s' = s^{-1} \pmod{t}$.
  - Then he computes point $s'h(m)P + s'a(eP) = (a', b')$.
  - He accepts the signature if $a = a'$.

History of Hash Functions

- In 1980s, several hash functions were proposed but none were secure.
- In 1991, Ron Rivesh proposed MD5, which was found suitable and got adopted widely.
  - It produces 128-bit output.
- In 2005, MD5 was shown to be insecure by demonstrating two distinct messages that hash to same value.
  - This also made another similar algorithm, SHA-1, insecure.
- In 2006, NIST started a competition to select a new secure hash algorithm that culminated in SHA-3 being selected in 2012.

History of Hash Functions

- In 1980s, several hash functions were proposed but none were secure.
- In 1991, Ron Rivesh proposed MD5, which was found suitable and got adopted widely.
  - It produces 128-bit output.
- In 2005, MD5 was shown to be insecure by demonstrating two distinct messages that hash to same value.
  - This also made another similar algorithm, SHA-1, insecure.
- In 2006, NIST started a competition to select a new secure hash algorithm that culminated in SHA-3 being selected in 2012.

History of Hash Functions

- In 1980s, several hash functions were proposed but none were secure.
- In 1991, Ron Rivesh proposed MD5, which was found suitable and got adopted widely.
  - It produces 128-bit output.
- In 2005, MD5 was shown to be insecure by demonstrating two distinct messages that hash to same value.
  - This also made another similar algorithm, SHA-1, insecure.
- In 2006, NIST started a competition to select a new secure hash algorithm that culminated in SHA-3 being selected in 2012.

History of Hash Functions

- In 1980s, several hash functions were proposed but none were secure.
- In 1991, Ron Rivesh proposed MD5, which was found suitable and got adopted widely.
    - It produces 128-bit output.
- In 2005, MD5 was shown to be insecure by demonstrating two distinct messages that hash to same value.
    - This also made another similar algorithm, SHA-1, insecure.
- In 2006, NIST started a competition to select a new secure hash algorithm that culminated in SHA-3 being selected in 2012.

SHA-3

- Let $r, b, d \in \mathbb{Z}$ where $b > r$. Let $c = b - r$.
- Let $f, \{0,1\}^b \mapsto \{0,1\}^b$ be a permutation.
- Let m be the input document with $|m| = N$.
- Break m into blocks of r bits, by padding if necessary. Let $m = m_1 m_2 \cdots m_t$.
- Let $s_0 = 0^b$ and define $s_i = f(s_{i-1} \oplus m_i 0^c)$ for $1 \leq i \leq t$.
- Let z_i be first r bits of s_{t+i}, and $s_{t+i+1} = f(s_{t+i})$ for $0 \leq i < d/r$.
- Output $z_0 z_1 z_2 \cdots$ truncated to d bits.

SHA-3

- Let $r, b, d \in \mathbb{Z}$ where $b > r$. Let $c = b - r$.
- Let $f, \{0,1\}^b \mapsto \{0,1\}^b$ be a permutation.
- Let m be the input document with $|m| = N$.
- Break m into blocks of r bits, by padding if necessary. Let $m = m_1 m_2 \cdots m_t$.
- Let $s_0 = 0^b$ and define $s_i = f(s_{i-1} \oplus m_i 0^c)$ for $1 \leq i \leq t$.
- Let z_i be first r bits of s_{t+i}, and $s_{t+i+1} = f(s_{t+i})$ for $0 \leq i < d/r$.
- Output $z_0 z_1 z_2 \cdots$ truncated to d bits.

SHA-3

- Let $r, b, d \in \mathbb{Z}$ where $b > r$. Let $c = b - r$.
- Let $f, \{0,1\}^b \mapsto \{0,1\}^b$ be a permutation.
- Let m be the input document with $|m| = N$.
- Break m into blocks of r bits, by padding if necessary. Let $m = m_1 m_2 \cdots m_t$.
- Let $s_0 = 0^b$ and define $s_i = f(s_{i-1} \oplus m_i 0^c)$ for $1 \leq i \leq t$.
- Let z_i be first r bits of s_{t+i}, and $s_{t+i+1} = f(s_{t+i})$ for $0 \leq i < d/r$.
- Output $z_0 z_1 z_2 \cdots$ truncated to d bits.

SHA-3

- Let $r, b, d \in \mathbb{Z}$ where $b > r$. Let $c = b - r$.
- Let $f, \{0,1\}^b \mapsto \{0,1\}^b$ be a permutation.
- Let m be the input document with $|m| = N$.
- Break m into blocks of r bits, by padding if necessary. Let $m = m_1 m_2 \cdots m_t$.
- Let $s_0 = 0^b$ and define $s_i = f(s_{i-1} \oplus m_i 0^c)$ for $1 \leq i \leq t$.
- Let z_i be first r bits of s_{t+i}, and $s_{t+i+1} = f(s_{t+i})$ for $0 \leq i < d/r$.
- Output $z_0 z_1 z_2 \cdots$ truncated to d bits.

SHA-3

- Let $r, b, d \in \mathbb{Z}$ where $b > r$. Let $c = b - r$.
- Let $f, \{0,1\}^b \mapsto \{0,1\}^b$ be a permutation.
- Let m be the input document with $|m| = N$.
- Break m into blocks of r bits, by padding if necessary. Let $m = m_1 m_2 \cdots m_t$.
- Let $s_0 = 0^b$ and define $s_i = f(s_{i-1} \oplus m_i 0^c)$ for $1 \leq i \leq t$.
- Let z_i be first r bits of s_{t+i}, and $s_{t+i+1} = f(s_{t+i})$ for $0 \leq i < d/r$.
- Output $z_0 z_1 z_2 \cdots$ truncated to d bits.

SHA-3

- Let $r, b, d \in \mathbb{Z}$ where $b > r$. Let $c = b - r$.
- Let $f, \{0,1\}^b \mapsto \{0,1\}^b$ be a permutation.
- Let m be the input document with $|m| = N$.
- Break m into blocks of r bits, by padding if necessary. Let $m = m_1 m_2 \cdots m_t$.
- Let $s_0 = 0^b$ and define $s_i = f(s_{i-1} \oplus m_i 0^c)$ for $1 \leq i \leq t$.
- Let z_i be first r bits of s_{t+i}, and $s_{t+i+1} = f(s_{t+i})$ for $0 \leq i < d/r$.
- Output $z_0 z_1 z_2 \cdots$ truncated to d bits.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:

θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.

ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number $0, 1, 3, 6, 10, 15, \ldots$.

π: $a[3i+2j][i] \leftarrow a[i][j]$.

χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.

ι: XOR a round constant to string $a[0][0]$.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:

  θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.

  ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number $0, 1, 3, 6, 10, 15, \ldots$.

  π: $a[3i+2j][i] \leftarrow a[i][j]$.

  χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.

  ι: XOR a round constant to string $a[0][0]$.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:

θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.

ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number $0, 1, 3, 6, 10, 15, \ldots$.

π: $a[3i+2j][i] \leftarrow a[i][j]$.

χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.

ι: XOR a round constant to string $a[0][0]$.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:

  θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.

  ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number $0, 1, 3, 6, 10, 15, \ldots$.

  π: $a[3i+2j][i] \leftarrow a[i][j]$.

  χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.

  ι: XOR a round constant to string $a[0][0]$.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:
  - θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.
  - ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number 0, 1, 3, 6, 10, 15, ...
  - π: $a[3i+2j][i] \leftarrow a[i][j]$.
  - χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.
  - ι: XOR a round constant to string $a[0][0]$.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:

  θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.

  ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number 0, 1, 3, 6, 10, 15, ...

  π: $a[3i+2j][i] \leftarrow a[i][j]$.

  χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.

  ι: XOR a round constant to string $a[0][0]$.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:
  - θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.
  - ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number $0, 1, 3, 6, 10, 15, \ldots$.
  - π: $a[3i+2j][i] \leftarrow a[i][j]$.
  - χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.
  - ι: XOR a round constant to string $a[0][0]$.

SHA-3: Function f

- Typically, $b = 1600$, and each s_i is viewed as a 5×5 array of 64-bit strings.
- Let $a[i][j][k]$ denote the kth bit of string at (i, j)th location in array.
- Function f consists of 24 rounds of following five operations:

  θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus_{u=0}^{4} (a[u][j-1][k] \oplus a[u][j+1][k])$ where index arithmetic is modulo 5.

  ρ: Bitwise rotate each string $a[i][j]$ by a different triangular number $0, 1, 3, 6, 10, 15, \ldots$.

  π: $a[3i+2j][i] \leftarrow a[i][j]$.

  χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \wedge a[i][j+2][k])$.

  ι: XOR a round constant to string $a[0][0]$.

SHA-3: Parameter Values

- $d = 224$, $r = 1152$, $c = 448$
- $d = 256$, $r = 1088$, $c = 512$
- $d = 384$, $r = 832$, $c = 768$
- $d = 512$, $r = 576$, $c = 1024$

SHA-3: Parameter Values

- $d = 224$, $r = 1152$, $c = 448$
- $d = 256$, $r = 1088$, $c = 512$
- $d = 384$, $r = 832$, $c = 768$
- $d = 512$, $r = 576$, $c = 1024$

SHA-3: Parameter Values

- $d = 224$, $r = 1152$, $c = 448$
- $d = 256$, $r = 1088$, $c = 512$
- $d = 384$, $r = 832$, $c = 768$
- $d = 512$, $r = 576$, $c = 1024$

SHA-3: Parameter Values

- $d = 224$, $r = 1152$, $c = 448$
- $d = 256$, $r = 1088$, $c = 512$
- $d = 384$, $r = 832$, $c = 768$
- $d = 512$, $r = 576$, $c = 1024$

Bit Commitment

- Suppose there is an online contest that requires participants to solve a particularly difficult problem.
- Further suppose that Anubha has solved the problem and wishes to submit the solution to the organizing site.
- In order to ensure that solution does not get leaked, Anubha can encrypt the solution using public key of the site and submit.
- However, there is a risk that someone at the organizing site may leak the solution to others.
- Can this be avoided?

Bit Commitment

- Suppose there is an online contest that requires participants to solve a particularly difficult problem.
- Further suppose that Anubha has solved the problem and wishes to submit the solution to the organizing site.
- In order to ensure that solution does not get leaked, Anubha can encrypt the solution using public key of the site and submit.
- However, there is a risk that someone at the organizing site may leak the solution to others.
- Can this be avoided?

Bit Commitment

- Suppose there is an online contest that requires participants to solve a particularly difficult problem.
- Further suppose that Anubha has solved the problem and wishes to submit the solution to the organizing site.
- In order to ensure that solution does not get leaked, Anubha can encrypt the solution using public key of the site and submit.
- However, there is a risk that someone at the organizing site may leak the solution to others.
- Can this be avoided?

Bit Commitment

- Suppose there is an online contest that requires participants to solve a particularly difficult problem.
- Further suppose that Anubha has solved the problem and wishes to submit the solution to the organizing site.
- In order to ensure that solution does not get leaked, Anubha can encrypt the solution using public key of the site and submit.
- However, there is a risk that someone at the organizing site may leak the solution to others.
- Can this be avoided?

Bit Commitment

- Suppose there is an online contest that requires participants to solve a particularly difficult problem.
- Further suppose that Anubha has solved the problem and wishes to submit the solution to the organizing site.
- In order to ensure that solution does not get leaked, Anubha can encrypt the solution using public key of the site and submit.
- However, there is a risk that someone at the organizing site may leak the solution to others.
- Can this be avoided?

Bit Commitment

- Let m be the solution of Anubha, and h be a cryptographically secure hash function.
- Anubha can submit $h(m)$ to the site instead of m.
- After the deadline for submitting solutions is over, Anubha can send m.
- Organizers can easily verify that solution m corresponds to earlier submission $h(m)$.
- Given $w = h(m)$, it is hard for anyone to find a string m' such that $h(m') = w$, as per the second property of secure hash functions.

Bit Commitment

- Let m be the solution of Anubha, and h be a cryptographically secure hash function.
- Anubha can submit $h(m)$ to the site instead of m.
- After the deadline for submitting solutions is over, Anubha can send m.
- Organizers can easily verify that solution m corresponds to earlier submission $h(m)$.
- Given $w = h(m)$, it is hard for anyone to find a string m' such that $h(m') = w$, as per the second property of secure hash functions.

Bit Commitment

- Let m be the solution of Anubha, and h be a cryptographically secure hash function.
- Anubha can submit $h(m)$ to the site instead of m.
- After the deadline for submitting solutions is over, Anubha can send m.
- Organizers can easily verify that solution m corresponds to earlier submission $h(m)$.
- Given $w = h(m)$, it is hard for anyone to find a string m' such that $h(m') = w$, as per the second property of secure hash functions.

Bit Commitment

- Let m be the solution of Anubha, and h be a cryptographically secure hash function.
- Anubha can submit $h(m)$ to the site instead of m.
- After the deadline for submitting solutions is over, Anubha can send m.
- Organizers can easily verify that solution m corresponds to earlier submission $h(m)$.
- Given $w = h(m)$, it is hard for anyone to find a string m' such that $h(m') = w$, as per the second property of secure hash functions.

Bit Commitment

- Let m be the solution of Anubha, and h be a cryptographically secure hash function.
- Anubha can submit $h(m)$ to the site instead of m.
- After the deadline for submitting solutions is over, Anubha can send m.
- Organizers can easily verify that solution m corresponds to earlier submission $h(m)$.
- Given $w = h(m)$, it is hard for anyone to find a string m' such that $h(m') = w$, as per the second property of secure hash functions.

OUTLINE

1. HASHING

2. PUBLIC KEY INFRASTRUCTURE (PKI)

Authentication

The Authentication Problem

How does Braj ascertain identity of Anubha remotely?

- Anubha can share her public-key with Braj and then digitally sign communication with Braj to prove her identity.
- But this only proves that the sender has the private-key corresponding to public-key sent to Braj.
- What is the proof that public-key belongs to Anubha?

Authentication

The Authentication Problem

How does Braj ascertain identity of Anubha remotely?

- Anubha can share her public-key with Braj and then digitally sign communication with Braj to prove her identity.
- But this only proves that the sender has the private-key corresponding to public-key sent to Braj.
- What is the proof that public-key belongs to Anubha?

Authentication

The Authentication Problem

How does Braj ascertain identity of Anubha remotely?

- Anubha can share her public-key with Braj and then digitally sign communication with Braj to prove her identity.
- But this only proves that the sender has the private-key corresponding to public-key sent to Braj.
- What is the proof that public-key belongs to Anubha?

Authentication

The Authentication Problem

How does Braj ascertain identity of Anubha remotely?

- Anubha can share her public-key with Braj and then digitally sign communication with Braj to prove her identity.
- But this only proves that the sender has the private-key corresponding to public-key sent to Braj.
- What is the proof that public-key belongs to Anubha?

Certification Authorities

- We can have designated certification authorities who verify the identity of Anubha and certify by digitally signing Anubha's public-key.
- Then Braj can be certain that the public key is indeed from Anibha.
- However, how does Braj know that certification authority's signatures are correct?
- One possibility is to go to designated websites that have public key of authorities.
  - The problem is that the website may get hacked and public key replaced by a malicious one.

Certification Authorities

- We can have designated certification authorities who verify the identity of Anubha and certify by digitally signing Anubha's public-key.
- Then Braj can be certain that the public key is indeed from Anibha.
- However, how does Braj know that certification authority's signatures are correct?
- One possibility is to go to designated websites that have public key of authorities.
  - The problem is that the website may get hacked and public key replaced by a malicious one.

Certification Authorities

- We can have designated certification authorities who verify the identity of Anubha and certify by digitally signing Anubha's public-key.
- Then Braj can be certain that the public key is indeed from Anibha.
- However, how does Braj know that certification authority's signatures are correct?
- One possibility is to go to designated websites that have public key of authorities.
  - The problem is that the website may get hacked and public key replaced by a malicious one.

Certification Authorities

- We can have designated certification authorities who verify the identity of Anubha and certify by digitally signing Anubha's public-key.
- Then Braj can be certain that the public key is indeed from Anibha.
- However, how does Braj know that certification authority's signatures are correct?
- One possibility is to go to designated websites that have public key of authorities.
  - The problem is that the website may get hacked and public key replaced by a malicious one.

Certification Authorities

- We can have designated certification authorities who verify the identity of Anubha and certify by digitally signing Anubha's public-key.
- Then Braj can be certain that the public key is indeed from Anibha.
- However, how does Braj know that certification authority's signatures are correct?
- One possibility is to go to designated websites that have public key of authorities.
  - The problem is that the website may get hacked and public key replaced by a malicious one.

Certification Authorities

- In order to ensure correct public keys of certification authorities, we can have higher authorities who certify these keys with their digital signatures.
- But the problem remains: how to ensure that public keys of higher authorities are not compromised?
- We can have even higher authorities who certify it, and they ensure that their public keys are never compromised.
- This is exactly how public-key infrastructure is implemented.
- Root CAs are highest authorities that guarantee that their public key can never get compromised.
  - Only a few entities in the world are root CAs.
  - Examples: Symantec, DigiCert, Comodo

Certification Authorities

- In order to ensure correct public keys of certification authorities, we can have higher authorities who certify these keys with their digital signatures.
- But the problem remains: how to ensure that public keys of higher authorities are not compromised?
- We can have even higher authorities who certify it, and they ensure that their public keys are never compromised.
- This is exactly how public-key infrastructure is implemented.
- Root CAs are highest authorities that guarantee that their public key can never get compromised.
  - Only a few entities in the world are root CAs.
  - Examples: Symantec, DigiCert, Comodo

Certification Authorities

- In order to ensure correct public keys of certification authorities, we can have higher authorities who certify these keys with their digital signatures.
- But the problem remains: how to ensure that public keys of higher authorities are not compromised?
- We can have even higher authorities who certify it, and they ensure that their public keys are never compromised.
- This is exactly how public-key infrastructure is implemented.
- Root CAs are highest authorities that guarantee that their public key can never get compromised.
  - Only a few entities in the world are root CAs.
  - Examples: Symantec, DigiCert, Comodo

Certification Authorities

- In order to ensure correct public keys of certification authorities, we can have higher authorities who certify these keys with their digital signatures.
- But the problem remains: how to ensure that public keys of higher authorities are not compromised?
- We can have even higher authorities who certify it, and they ensure that their public keys are never compromised.
- This is exactly how public-key infrastructure is implemented.
- Root CAs are highest authorities that guarantee that their public key can never get compromised.
  - Only a few entities in the world are root CAs.
  - Examples: Symantec, DigiCert, Comodo

Certification Authorities

- In order to ensure correct public keys of certification authorities, we can have higher authorities who certify these keys with their digital signatures.
- But the problem remains: how to ensure that public keys of higher authorities are not compromised?
- We can have even higher authorities who certify it, and they ensure that their public keys are never compromised.
- This is exactly how public-key infrastructure is implemented.
- Root CAs are highest authorities that guarantee that their public key can never get compromised.
  - Only a few entities in the world are root CAs.
  - Examples: Symantec, DigiCert, Comodo

Certification Authorities

- In order to ensure correct public keys of certification authorities, we can have higher authorities who certify these keys with their digital signatures.
- But the problem remains: how to ensure that public keys of higher authorities are not compromised?
- We can have even higher authorities who certify it, and they ensure that their public keys are never compromised.
- This is exactly how public-key infrastructure is implemented.
- Root CAs are highest authorities that guarantee that their public key can never get compromised.
  - Only a few entities in the world are root CAs.
  - Examples: Symantec, DigiCert, Comodo

Certification Authorities

- In order to ensure correct public keys of certification authorities, we can have higher authorities who certify these keys with their digital signatures.
- But the problem remains: how to ensure that public keys of higher authorities are not compromised?
- We can have even higher authorities who certify it, and they ensure that their public keys are never compromised.
- This is exactly how public-key infrastructure is implemented.
- Root CAs are highest authorities that guarantee that their public key can never get compromised.
  - Only a few entities in the world are root CAs.
  - Examples: Symantec, DigiCert, Comodo

Certification Authorities

- Intermediate CAs get their public key signed by a root CA.
- Issuing CAs get their public key signed by an intermediate CA.
- Users get their public key signed by an issuing CA.
- The entire setup is referred as Public-key Infrastructure.

Certification Authorities

- Intermediate CAs get their public key signed by a root CA.
- Issuing CAs get their public key signed by an intermediate CA.
- Users get their public key signed by an issuing CA.
- The entire setup is referred as Public-key Infrastructure.

Certification Authorities

- Intermediate CAs get their public key signed by a root CA.
- Issuing CAs get their public key signed by an intermediate CA.
- Users get their public key signed by an issuing CA.
- The entire setup is referred as Public-key Infrastructure.

Certification Authorities

- Intermediate CAs get their public key signed by a root CA.
- Issuing CAs get their public key signed by an intermediate CA.
- Users get their public key signed by an issuing CA.
- The entire setup is referred as Public-key Infrastructure.