CSc 466/566

Computer Security

14 : Cryptography — Digital Signatures

Version: 2014/10/30 15:27:25

Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

Outline

Digital Signatures

- In this lecture we are going to talk about cryptographic hash functions (checksums) and digital signatures.
- We want to be able to
  1. Detect tampering: is the message we received the same as the message that was sent?
  2. Authenticate: did the message come from who we think it came from?

Digital Signatures...

- More specifically, we want to ensure:
  1. Nonforgeability: Eve should not be able to create a message that appears to come from Alice.
  2. Nonmutability: Eve should not be able to take a valid signature for one message from Alice, and apply it to another one.
  3. Nonrepudiation: Alice should not be able to claim she didn't sign a document that she did sign.

Digital Signatures...

- In the non-digital world, Alice would sign the document. We can do the same with digital signatures.

Digital Signatures...

- In the non-digital world, Alice would sign the document. We can do the same with digital signatures.

1. Alice encrypts her document M with her private key S_A, thereby creating a signature $S_{\mathrm{Alice}}(M)$.

Digital Signatures...

- In the non-digital world, Alice would sign the document. We can do the same with digital signatures.

1. Alice encrypts her document M with her private key S_A, thereby creating a signature $S_{\mathrm{Alice}}(M)$.
2. Alice sends M and the signature $S_{\mathrm{Alice}}(M)$ to Bob.

Digital Signatures...

- In the non-digital world, Alice would sign the document. We can do the same with digital signatures.

1. Alice encrypts her document M with her private key S_A, thereby creating a signature $S_{\mathrm{Alice}}(M)$.
2. Alice sends M and the signature $S_{\mathrm{Alice}}(M)$ to Bob.
3. Bob decrypts the document using Alice's public key, thereby verifying her signature.

Digital Signatures...

- This works because for many public key ciphers

$$\begin{aligned} D_{S_B}(E_{P_B}(M)) &= M \\ E_{P_B}(D_{S_B}(M)) &= M \end{aligned}$$

  i.e. we can reverse the encryption/decryption operations.
- That is, Bob can apply the decryption function to a message with his private key S_B, yielding the signature sig:

$$\mathrm{sig} \leftarrow D_{S_B}(M)$$

- Then, anyone else can apply the encryption function to sig to get the message back. Only Bob (who has his secret key) could have generated the signature:

$$E_{P_B}(\mathrm{sig}) = M$$

Digital Signatures…

Bob

Alice

$\boxed{M}$

Bob sent M?

Digital Signatures...

Bob

Alice

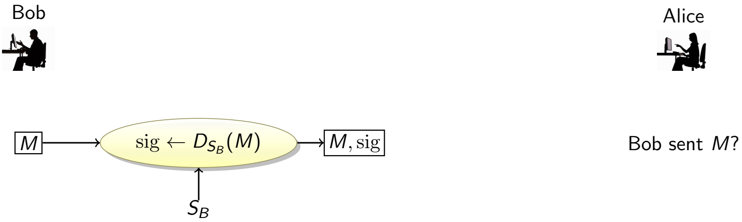
$\boxed{M} \longrightarrow \mathrm{sig} \leftarrow D_{S_B}(M)$

$\uparrow$

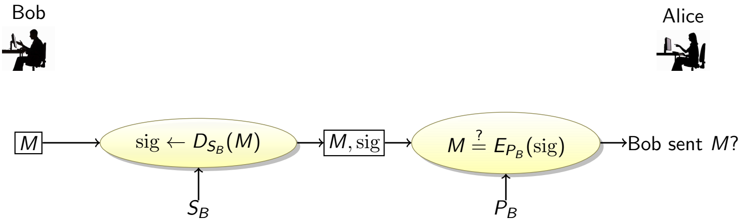
S_B

Bob sent M?

Digital Signatures...



Digital Signatures...



Outline

RSA Signature Scheme

RSA Signature Scheme

1. Alice applies the decryption function to her document M with her private key S_A, thereby creating a signature $S_{\mathrm{Alice}}(M)$.

RSA Signature Scheme

1. Alice applies the decryption function to her document M with her private key S_A, thereby creating a signature $S_{\mathrm{Alice}}(M)$.
2. Alice sends M and the signature $S_{\mathrm{Alice}}(M)$ to Bob.

RSA Signature Scheme

1. Alice applies the decryption function to her document M with her private key S_A, thereby creating a signature $S_{\mathrm{Alice}}(M)$.
2. Alice sends M and the signature $S_{\mathrm{Alice}}(M)$ to Bob.
3. Bob applies the encryption function to the document using Alice's public key, thereby verifying her signature.

RSA Encryption: Algorithm

- Bob (Key generation):

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p - 1)(q - 1)$.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Alice (encrypt and send a message M to Bob):

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (e, n)$.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (e, n)$.
  2. Compute $C = M^e \bmod n$.

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (e, n)$.
  2. Compute $C = M^e \bmod n$.
- Bob (decrypt a message C received from Alice):

RSA Encryption: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (e, n)$.
  2. Compute $C = M^e \bmod n$.
- Bob (decrypt a message C received from Alice):
  1. Compute $M = C^d \bmod n$.

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Bob (sign a secret message M):

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Bob (sign a secret message M):
  1. Compute $S = M^d \bmod n$.

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Bob (sign a secret message M):
  1. Compute $S = M^d \bmod n$.
  2. Send M, S to Alice.

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Bob (sign a secret message M):
  1. Compute $S = M^d \bmod n$.
  2. Send M, S to Alice.
- Alice (verify signature S received from Bob):

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Bob (sign a secret message M):
  1. Compute $S = M^d \bmod n$.
  2. Send M, S to Alice.
- Alice (verify signature S received from Bob):
  1. Receive M, S from Alice.

RSA Signature Scheme: Algorithm

- Bob (Key generation): As before.
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Bob (sign a secret message M):
  1. Compute $S = M^d \bmod n$.
  2. Send M, S to Alice.
- Alice (verify signature S received from Bob):
  1. Receive M, S from Alice.
  2. Verify that $M \stackrel{?}{=} S^e \bmod n$.

RSA Signature Scheme: Correctness

- We have:

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
  - $S = M^d \bmod n$.

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
  - $S = M^d \bmod n$.
  - $d = e^{-1} \bmod \phi(n) \Rightarrow de = 1 \bmod \phi(n)$.

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
  - $S = M^d \bmod n$.
  - $d = e^{-1} \bmod \phi(n) \Rightarrow de = 1 \bmod \phi(n)$.

Theorem (Corollary to Euler's theorem)

Let x be any positive integer that's relatively prime to the integer $n > 0$, and let k be any positive integer, then

$$x^k \bmod n = x^{k \bmod \phi(n)} \bmod n$$

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
  - $S = M^d \bmod n$.
  - $d = e^{-1} \bmod \phi(n) \Rightarrow de = 1 \bmod \phi(n)$.

Theorem (Corollary to Euler's theorem)

Let x be any positive integer that's relatively prime to the integer $n > 0$, and let k be any positive integer, then

$$x^k \bmod n = x^{k \bmod \phi(n)} \bmod n$$

- Alice wants to verify that $M \stackrel{?}{=} S^e \bmod n$.

$$S^e \bmod n \quad = \quad M^{de} \bmod n$$

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
  - $S = M^d \bmod n$.
  - $d = e^{-1} \bmod \phi(n) \Rightarrow de = 1 \bmod \phi(n)$.

Theorem (Corollary to Euler's theorem)

Let x be any positive integer that's relatively prime to the integer $n > 0$, and let k be any positive integer, then

$$x^k \bmod n = x^{k \bmod \phi(n)} \bmod n$$

- Alice wants to verify that $M \stackrel{?}{=} S^e \bmod n$.

$$S^e \bmod n \quad = \quad M^{de} \bmod n$$

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
  - $S = M^d \bmod n$.
  - $d = e^{-1} \bmod \phi(n) \Rightarrow de = 1 \bmod \phi(n)$.

Theorem (Corollary to Euler's theorem)

Let x be any positive integer that's relatively prime to the integer $n > 0$, and let k be any positive integer, then

$$x^k \bmod n = x^{k \bmod \phi(n)} \bmod n$$

- Alice wants to verify that $M \stackrel{?}{=} S^e \bmod n$.

$$\begin{aligned} S^e \bmod n &= M^{de} \bmod n \\ &= M^{de \bmod \phi(n)} \bmod n \end{aligned}$$

RSA Signature Scheme: Correctness

- We have:
  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
  - $S = M^d \bmod n$.
  - $d = e^{-1} \bmod \phi(n) \Rightarrow de = 1 \bmod \phi(n)$.

Theorem (Corollary to Euler's theorem)

Let x be any positive integer that's relatively prime to the integer $n > 0$, and let k be any positive integer, then

$$x^k \bmod n = x^{k \bmod \phi(n)} \bmod n$$

- Alice wants to verify that $M \stackrel{?}{=} S^e \bmod n$.

$$\begin{aligned} S^e \bmod n &= M^{de} \bmod n \\ &= M^{de \bmod \phi(n)} \bmod n \\ &= M^1 \bmod n = M \end{aligned}$$

RSA signature: Nonforgeability

- Nonforgeability: Eve should not be able to create a message that appears to come from Alice.

RSA signature: Nonforgeability

- Nonforgeability: Eve should not be able to create a message that appears to come from Alice.
- To forge a message M from Alice, Eve would have to produce

$$M^d \bmod n$$

without knowing Alice's private key d.

RSA signature: Nonforgeability

- Nonforgeability: Eve should not be able to create a message that appears to come from Alice.
- To forge a message M from Alice, Eve would have to produce

$$M^d \bmod n$$

without knowing Alice's private key d.
- This is equivalent to being able to break RSA encryption.

RSA signature: Nonmutability

- Nonmutability: Eve should not be able to take a valid signature for one message from Alice, and apply it to another message.

RSA signature: Nonmutability

- Nonmutability: Eve should not be able to take a valid signature for one message from Alice, and apply it to another message.
- RSA does not achieve nonmutability.

RSA signature: Nonmutability

- Nonmutability: Eve should not be able to take a valid signature for one message from Alice, and apply it to another message.
- RSA does not achieve nonmutability.
- Assume Eve has two valid signatures from Alice, on two messages M_1 and M_2:

$$\begin{aligned} S_1 &= M_1^d \bmod n \\ S_2 &= M_2^d \bmod n \end{aligned}$$

RSA signature: Nonmutability

- Nonmutability: Eve should not be able to take a valid signature for one message from Alice, and apply it to another message.
- RSA does not achieve nonmutability.
- Assume Eve has two valid signatures from Alice, on two messages M_1 and M_2:

$$\begin{aligned} S_1 &= M_1^d \bmod n \\ S_2 &= M_2^d \bmod n \end{aligned}$$

- Eve can then produce a new signature

$$\begin{aligned} S_1 \cdot S_2 &= (M_1^d \bmod n) \cdot (M_2^d \bmod n) \\ &= (M_1 \cdot M_2)^d \bmod n \end{aligned}$$

  This is a valid signature for the message $M_1 \cdot M_2$!

RSA signature: Nonmutability

- Nonmutability: Eve should not be able to take a valid signature for one message from Alice, and apply it to another message.
- RSA does not achieve nonmutability.
- Assume Eve has two valid signatures from Alice, on two messages M_1 and M_2:

$$
\begin{aligned}
S_1 &= M_1^d \bmod n \\
S_2 &= M_2^d \bmod n
\end{aligned}
$$

- Eve can then produce a new signature

$$
\begin{aligned}
S_1 \cdot S_2 &= (M_1^d \bmod n) \cdot (M_2^d \bmod n) \\
&= (M_1 \cdot M_2)^d \bmod n
\end{aligned}
$$

  This is a valid signature for the message $M_1 \cdot M_2$!
- Not usually a problem since we normally sign hashes.

Outline

Elgamal: Encryption Algorithm

- Bob (Key generation):

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p - 2$.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_B = (p, g, y)$ is Bob's Elgamal public key.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_B = (p, g, y)$ is Bob's Elgamal public key.
  - $S_B = x$ is Bob' Elgamal private key.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
     - $P_B = (p, g, y)$ is Bob's Elgamal public key.
     - $S_B = x$ is Bob' Elgamal private key.
- Alice (encrypt and send a message M to Bob):

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p - 2$.
  4. Compute $y = g^x \bmod p$.
     - $P_B = (p, g, y)$ is Bob's Elgamal public key.
     - $S_B = x$ is Bob' Elgamal private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (p, g, y)$.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p - 2$.
  4. Compute $y = g^x \bmod p$.
  - $P_B = (p, g, y)$ is Bob's Elgamal public key.
  - $S_B = x$ is Bob' Elgamal private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (p, g, y)$.
  2. Pick a random number k between 1 and $p - 2$.

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_B = (p, g, y)$ is Bob's Elgamal public key.
  - $S_B = x$ is Bob' Elgamal private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (p, g, y)$.
  2. Pick a random number k between 1 and $p-2$.
  3. Compute the ciphertext $C = (a, b)$:

$$\begin{aligned} a &= g^k \bmod p \\ b &= My^k \bmod p \end{aligned}$$

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_B = (p, g, y)$ is Bob's Elgamal public key.
  - $S_B = x$ is Bob' Elgamal private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (p, g, y)$.
  2. Pick a random number k between 1 and $p-2$.
  3. Compute the ciphertext $C = (a, b)$:

$$\begin{aligned} a &= g^k \bmod p \\ b &= My^k \bmod p \end{aligned}$$

- Bob (decrypt a message $C = (a, b)$ received from Alice):

Elgamal: Encryption Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
     - $P_B = (p, g, y)$ is Bob's Elgamal public key.
     - $S_B = x$ is Bob' Elgamal private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (p, g, y)$.
  2. Pick a random number k between 1 and $p-2$.
  3. Compute the ciphertext $C = (a, b)$:

$$\begin{aligned} a &= g^k \bmod p \\ b &= My^k \bmod p \end{aligned}$$

- Bob (decrypt a message $C = (a, b)$ received from Alice):
  1. Compute $M = b(a^x)^{-1} \bmod p$.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p - 2$.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p - 2$.
  4. Compute $y = g^x \mod p$.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_A = (p, g, y)$ is Alice's Elgamal public key.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_A = (p, g, y)$ is Alice's Elgamal public key.
  - $S_A = x$ is Alice' Elgamal private key.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
     - $P_A = (p, g, y)$ is Alice's Elgamal public key.
     - $S_A = x$ is Alice' Elgamal private key.
- Alice (sign message M and send to Bob):

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.

  - $P_A = (p, g, y)$ is Alice's Elgamal public key.
  - $S_A = x$ is Alice' Elgamal private key.
- Alice (sign message M and send to Bob):
  1. Pick a random number k.

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
     - $P_A = (p, g, y)$ is Alice's Elgamal public key.
     - $S_A = x$ is Alice' Elgamal private key.
- Alice (sign message M and send to Bob):
  1. Pick a random number k.
  2. Compute the signature $S = (a, b)$:

$$\begin{aligned} a &= g^k \bmod p \\ b &= k^{-1}(M - xa) \bmod (p-1) \end{aligned}$$

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_A = (p, g, y)$ is Alice's Elgamal public key.
  - $S_A = x$ is Alice' Elgamal private key.
- Alice (sign message M and send to Bob):
  1. Pick a random number k.
  2. Compute the signature $S = (a, b)$:

$$\begin{aligned} a &= g^k \bmod p \\ b &= k^{-1}(M - xa) \bmod (p-1) \end{aligned}$$

- Bob (verify the signature $S = (a, b)$ received from Alice):

Elgamal: Signature Algorithm

- Alice (Key generation): As before.
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_A = (p, g, y)$ is Alice's Elgamal public key.
  - $S_A = x$ is Alice' Elgamal private key.
- Alice (sign message M and send to Bob):
  1. Pick a random number k.
  2. Compute the signature $S = (a, b)$:

$$\begin{aligned} a &= g^k \bmod p \\ b &= k^{-1}(M - xa) \bmod (p-1) \end{aligned}$$

- Bob (verify the signature $S = (a, b)$ received from Alice):
  1. Verify $y^a \cdot a^b \bmod p \stackrel{?}{=} g^M \bmod p$.

Elgamal Signature Algorithm: Correctness

- We have:

$$\begin{aligned} y &= g^x \bmod p \\ a &= g^k \bmod p \\ b &= k^{-1}(M - xa) \bmod (p-1) \end{aligned}$$

- Show that $y^a \cdot a^b \bmod p = g^M \bmod p$.

$$\begin{aligned} y^a a^b \bmod p &= (g^x \bmod p)^a ((g^k \bmod p)^(k^{-1}(M - xa) \bmod (p-1)) \text{ m} \\ &= g^{xa} g^{kk^{-1}(M-xa) \bmod (p-1)} \bmod p \\ &= g^{xa} g^{(M-xa) \bmod (p-1)} \bmod p \\ &= g^{xa} g^{M-xa} \bmod p \\ &= g^{xa+M-xa} \bmod p \\ &= g^M \bmod p \end{aligned}$$

Elgamal Signature Algorithm: Security

- We have:

$$\begin{aligned} y &= g^x \bmod p \\ a &= g^k \bmod p \\ b &= k^{-1}(M - xa) \bmod (p-1) \end{aligned}$$

- k is random $\Rightarrow$ b is random!
- To the adversary, b looks completely random.
- The adversary must compute k from $a = g^k \bmod p \Leftrightarrow$ compute discrete log!
- If Alice reuses $k \Rightarrow$ The adversary can compute the secret key.

Outline

Cryptographic Hash Functions

- Public key algorithms are too slow to sign large documents. A better protocol is to use a one way hash function also known as a cryptographic hash function (CHF).
- CHFs are checksums or compression functions: they take an arbitrary block of data and generate a unique, short, fixed-size, bitstring.

```
> echo "hello" | sha1sum
f572d396fae9206628714fb2ce00f72e94f2258f  -
> echo "hella" | sha1sum
1519ca327399f9d699afb0f8a3b7e1ea9d1edd0c  -
> echo "can't believe it's not butter!"|sha1sum
34e780e19b07b003b7cf1babba8ef7399b7f81dd  -
```

Signature Protocol

1. Bob computes a one-way hash of his document.

$$
\begin{aligned}
\text{hash} &\leftarrow h(M) \\
\text{sig} &\leftarrow E_{S_B}(\text{hash}) \\
D_{P_B}(\text{sig}) &\stackrel{?}{=} h(M)
\end{aligned}
$$

Signature Protocol

1. Bob computes a one-way hash of his document.
2. Bob decrypts the hash with his private key, thereby signing it.

$$
\begin{aligned}
\text{hash} &\leftarrow h(M) \\
\text{sig} &\leftarrow E_{S_B}(\text{hash}) \\
D_{P_B}(\text{sig}) &\stackrel{?}{=} h(M)
\end{aligned}
$$

Signature Protocol

1. Bob computes a one-way hash of his document.
2. Bob decrypts the hash with his private key, thereby signing it.
3. Bob sends the decrypted hash and the document to Alice.

$$\begin{aligned} \text{hash} &\leftarrow h(M) \\ \text{sig} &\leftarrow E_{S_B}(\text{hash}) \\ D_{P_B}(\text{sig}) &\stackrel{?}{=} h(M) \end{aligned}$$

Signature Protocol

1. Bob computes a one-way hash of his document.
2. Bob decrypts the hash with his private key, thereby signing it.
3. Bob sends the decrypted hash and the document to Alice.
4. Alice decrypts the hash Bob sent him, and compares it against a hash she computes herself of the document. If they are the same, the signature is valid.

$$\begin{aligned} \text{hash} &\leftarrow h(M) \\ \text{sig} &\leftarrow E_{S_B}(\text{hash}) \\ D_{P_B}(\text{sig}) &\stackrel{?}{=} h(M) \end{aligned}$$

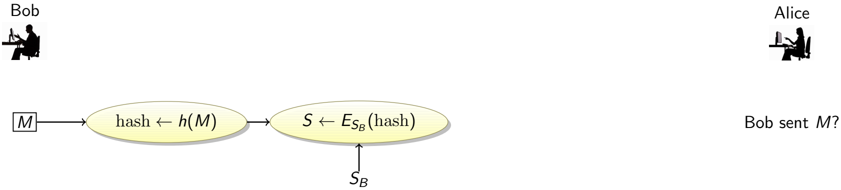
Signature Protocol...

- Advantage: the signature is short; defends against MITM attack.

Signature Protocol...

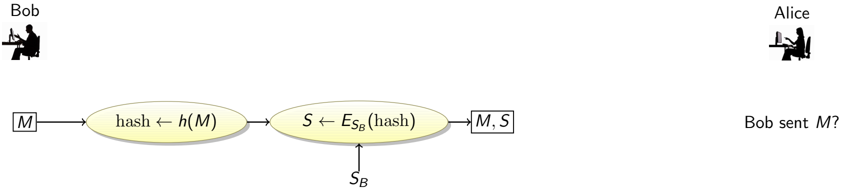
- Advantage: the signature is short; defends against MITM attack.

Signature Protocol. . .



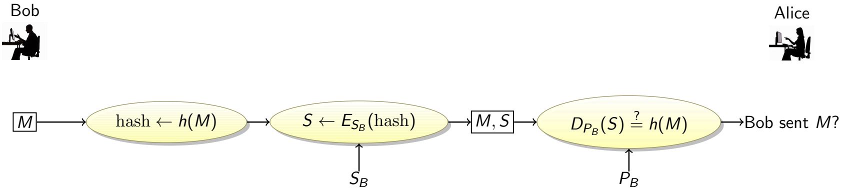
- Advantage: the signature is short; defends against MITM attack.

Signature Protocol...



- Advantage: the signature is short; defends against MITM attack.

Signature Protocol...



- Advantage: the signature is short; defends against MITM attack.

Cryptographic Hash Functions...

- CHFs should be
  1. deterministic
  2. one-way
  3. collision-resistant

  i.e., easy to compute, but hard to invert.
- I.e.
  - given message M, it's easy to compute $y \leftarrow h(M)$;
  - given a value y it's hard to compute an M such that $y = h(M)$.

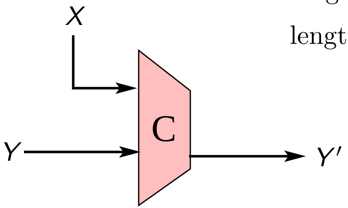
  This is what we mean by CHFs being one-way.

Weak vs. Strong Collision Resistance

- CHFs also have the property to be collision resistant.
- Weak collision resistance:
  - Assume you have a message M with hash value $h(M)$.
  - Then it should be hard to find a different message M' such that $h(M) = h(M')$.
- Strong collision resistance:
  - It should be hard to find two different message M_1 and M_2 such that $h(M_1) = h(M_2)$.
- Strong collisions resistance is hard to prove.

Merkle-Damgøard Construction

- Hash functions are often built on a compression function $C(X, Y)$:

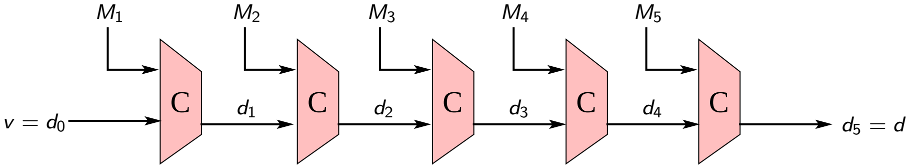


$\text{length}(X) = m$

$\text{length}(Y) = \text{length}(Y') = n$

- X is (a piece of) the message we're hashing.
- Y and Y' is the hash value we're computing.

Merkle-Damgård Construction...



- For long messages M we break it into pieces $M_1, \ldots, M_k$, each of size m.
- Our initial hash value is an initialization vector v.
- We then compress one M_i at a time, chaining it together on the previous hash value.

Outline

The Birthday Problem

- Given a group of n people, what is the probability that two share a birthday?
- Examine the probability that no two share a birthday: (let B_i be person i's birthday)
  - $n = 1 : 1$
  - $n = 2 : 364/365$
  - $n = 3$: probability that B_3 differs from both B_1 and B_2 and that none of the first two share a birthday: $363/365 * 364/365$
  - $n = 4$:, probability that B_4 differs from all of $B_{1...3}$ and that none of the first three share a birthday: $362/365 * (363/365 * 364/365)$
  - and so on ...

The Birthday Problem

- This generalizes to

$$\frac{365!}{365^n(365-n)!}$$

- It takes only 23 people to give greater than .5 probability that two people share a birthday in a domain with cardinality 365.
- For a domain with cardinality c, .5 probability is reached with approximately $1.2\sqrt{c}$ numbers.
- So what does this have to do with checksums?

The Birthday Problem...

- Assume our hash function H has b-bit output.

The Birthday Problem. . .

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.

The Birthday Problem...

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:

The Birthday Problem...

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:
    1. Eve generates large number of messages $m_1, m_2, \ldots$.

The Birthday Problem...

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:
  1. Eve generates large number of messages $m_1, m_2, \ldots$.
  2. She computes their hash values $H(m_1), H(m_2), \ldots$.

The Birthday Problem...

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:
  1. Eve generates large number of messages $m_1, m_2, \ldots$.
  2. She computes their hash values $H(m_1), H(m_2), \ldots$.
  3. She waits for two messages m_i and m_j such that $H(m_i) = H(m_j)$.

The Birthday Problem...

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:
  1. Eve generates large number of messages $m_1, m_2, \ldots$.
  2. She computes their hash values $H(m_1), H(m_2), \ldots$.
  3. She waits for two messages m_i and m_j such that $H(m_i) = H(m_j)$.
- Eve needs to generate $\approx 2^b$ inputs to find a collision, right?

The Birthday Problem. . .

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:
  1. Eve generates large number of messages $m_1, m_2, \ldots$.
  2. She computes their hash values $H(m_1), H(m_2), \ldots$.
  3. She waits for two messages m_i and m_j such that $H(m_i) = H(m_j)$.
- Eve needs to generate $\approx 2^b$ inputs to find a collision, right?
- Wrong! By the birthday paradox, it is likely that two messages will have the same hash value!

The Birthday Problem...

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:
  1. Eve generates large number of messages $m_1, m_2, \ldots$.
  2. She computes their hash values $H(m_1), H(m_2), \ldots$.
  3. She waits for two messages m_i and m_j such that $H(m_i) = H(m_j)$.
- Eve needs to generate $\approx 2^b$ inputs to find a collision, right?
- Wrong! By the birthday paradox, it is likely that two messages will have the same hash value!
- Security is $\approx 2^{b/2}$ not 2^b.

The Birthday Problem...

- Assume our hash function H has b-bit output.
- The number of possible hash values is 2^b.
- Attack:
  1. Eve generates large number of messages $m_1, m_2, \ldots$.
  2. She computes their hash values $H(m_1), H(m_2), \ldots$.
  3. She waits for two messages m_i and m_j such that $H(m_i) = H(m_j)$.
- Eve needs to generate $\approx 2^b$ inputs to find a collision, right?
- Wrong! By the birthday paradox, it is likely that two messages will have the same hash value!
- Security is $\approx 2^{b/2}$ not 2^b.
- Thus, a hash-function with 256-bit output has 128-bit security.

Birthday Attacks

- Little Billy wants to be the sole beneficiary of Grandma's will
- He prepares two message templates, like the one Charlie made, one being a field trip permission slip, and the other being a will in which Grandma bequeaths everything to her sweet grandson.
- Little Billy finds a pair of messages, one generated from each template, with equal checksums
- Little Billy has Grandma sign the field trip permission slip
- Little Billy now has a signature that checks out against the will he created
- Profit!!

Outline

Summary

- Digital signatures make a message tamper-proof and give us authentication and nonrepudiation
- They only show that it was signed by a specific key, however
- It's cheaper to sign a checksum of the message rather than the whole message
  - Cryptographic checksums are necessary to do this securely

Readings and References

- Chapter 8.1.7, 8.2.1, 8.5.2 in *Introduction to Computer Security*, by Goodrich and Tamassia.

Acknowledgments

Additional material and exercises have also been collected from these sources:

1. Matthew Landis, *620—Fall 2003—Cryptographic Checksums and Digital Signatures*.
2. RFC1321 (MD5), www.ietf.org/rfc/rfc1321.txt