

Lecture 9

October 11, 2021

Checksums

- Mathematical function to generate a set of k bits from a set of n bits (where $k \leq n$).
 - k is smaller than n except in unusual circumstances
- Example: ASCII parity bit
 - ASCII has 7 bits; 8th bit is “parity”
 - Even parity: even number of 1 bits
 - Odd parity: odd number of 1 bits

Example Use

- Bob receives “10111101” as bits.
 - Sender is using even parity; 6 1 bits, so character was received correctly
 - Note: could be garbled, but 2 bits would need to have been changed to preserve parity
 - Sender is using odd parity; even number of 1 bits, so character was not received correctly

Definition of Cryptographic Checksum

Cryptographic checksum $h: A \rightarrow B$:

1. For any $x \in A$, $h(x)$ is easy to compute
2. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$
3. It is computationally infeasible to find two inputs $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$
 - Alternate form (stronger): Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$.

Collisions

- If $x \neq x'$ and $h(x) = h(x')$, x and x' are a *collision*
 - Pigeonhole principle: if there are n containers for $n+1$ objects, then at least one container will have at least 2 objects in it.
 - Application: if there are 32 files and 8 possible cryptographic checksum values, at least one value corresponds to at least 4 files

Keys

- Keyed cryptographic checksum: requires cryptographic key
 - AES in chaining mode: encipher message, use last n bits. Requires a key to encipher, so it is a keyed cryptographic checksum.
- Keyless cryptographic checksum: requires no cryptographic key
 - SHA-512, SHA-3 are examples; older ones include MD4, MD5, RIPEM, SHA-0, and SHA-1 (methods for constructing collisions are known for these)

HMAC

- Make keyed cryptographic checksums from keyless cryptographic checksums
- h keyless cryptographic checksum function that takes data in blocks of b bytes and outputs blocks of l bytes. k' is cryptographic key of length b bytes
 - If short, pad with 0 bytes; if long, hash to length b
- $ipad$ is 00110110 repeated b times
- $opad$ is 01011100 repeated b times
- $HMAC-h(k, m) = h(k' \oplus opad || h(k' \oplus ipad || m))$
 - \oplus exclusive or, $||$ concatenation

Strength of HMAC- h

- Depends on the strength of the hash function h
- Attacks on HMAC-MD4, HMAC-MD5, HMAC-SHA-0, and HMAC-SHA-1 recover partial or full keys
 - Note all of MD4, MD5, SHA-0, and SHA-1 have been broken

Digital Signature

- Construct that authenticates origin, contents of message in a manner provable to a disinterested third party (a “judge”)
- Sender cannot deny having sent message (service is “nonrepudiation”)
 - Limited to *technical* proofs
 - Inability to deny one’s cryptographic key was used to sign
 - One could claim the cryptographic key was stolen or compromised
 - Legal proofs, *etc.*, probably required; not dealt with here

Common Error

- Symmetric: Alice, Bob share key k
 - Alice sends $m || \{m\}_k$ to Bob
 - $\{m\}_k$ means m enciphered with key k , $||$ means concatenation

Claim: This is a digital signature

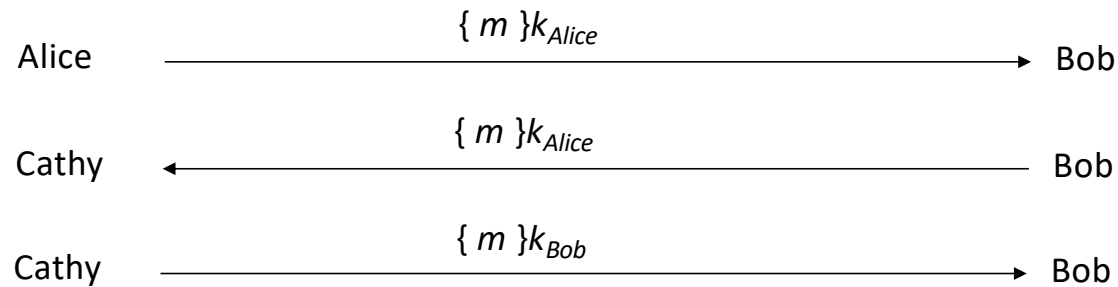
WRONG

This is not a digital signature

- Why? Third party cannot determine whether Alice or Bob generated message

Classical Digital Signatures

- Require trusted third party
 - Alice, Bob each share keys with trusted party Cathy
- To resolve dispute, judge gets $\{ m \}_{k_{Alice}}$, $\{ m \}_{k_{Bob}}$, and has Cathy decipher them; if messages matched, contract was signed



Public Key Digital Signatures

- Basically, Alice enciphers the message, or its cryptographic hash, with her private key
- In case of dispute or question of origin or whether changes have been made, a judge can use Alice's public key to verify the message came from Alice and has not been changed since being signed

RSA Digital Signatures

- Alice's keys are (e_{Alice}, n_{Alice}) (public key), d_{Alice} (private key)
 - In what follows, we use e_{Alice} to represent the public key

- Alice sends Bob

$$m || \{ m \}_{e_{Alice}}$$

- In case of dispute, judge computes

$$\{ \{ m \}_{e_{Alice}} \}_{d_{Alice}}$$

- and if it is m , Alice signed message
 - She's the only one who knows d_{Alice} !

RSA Digital Signatures

- Use private key to encipher message
 - Protocol for use is *critical*
- Key points:
 - Never sign random documents, and when signing, always sign hash and never document
 - Don't just encipher message and then sign, or vice versa
 - Changing public key and private key can cause problems
 - Messages can be forwarded, so third party cannot tell if original sender sent it to her

Attack #1

- Example: Alice, Bob communicating
 - $n_A = 262631, e_A = 154993, d_A = 95857$
 - $n_B = 288329, e_B = 22579, d_B = 138091$
- Alice asks Bob to sign 225536 so she can verify she has the right public key:
 - $c = m^{d_B} \bmod n_B = 225536^{138091} \bmod 288329 = 271316$
- Now she asks Bob to sign the statement AYE (002404):
 - $c = m^{d_B} \bmod n_B = 002404^{138091} \bmod 288329 = 182665$

Attack #1

- Alice computes:
 - new message NAY (130024) by $(002404)(225536) \bmod 288329 = 130024$
 - corresponding signature $(271316)(182665) \bmod 288329 = 218646$
- Alice now claims Bob signed NAY (130024), and as proof supplies signature 218646
- Judge computes $c^{e_B} \bmod n_B = 218646^{22579} \bmod 288329 = 130024$
 - Signature validated; Bob is toast

Preventing Attack #1

- Do not sign random messages
 - This would prevent Alice from getting the first message
- When signing, always sign the cryptographic hash of a message, not the message itself

Attack #2: Bob's Revenge

- Bob, Alice agree to sign contract LUR (112017)
 - But Bob really wants her to sign contract EWM (042212), but knows she won't
- Alice enciphers, then signs:
 - $(m^{e_B} \bmod n_A)^{d_A} \bmod n_A = (112017^{22579} \bmod 288329)^{95857} \bmod 262631 = 42390$
- Bob now changes his public key
 - Computes r such that $042212^r \bmod 288329 = 112017$; one such $r = 9175$
 - Computes $re_B \bmod \phi(n_B) = (9175)(22579) \bmod 287184 = 102661$
 - Replace public key with $(102661, 288329)$, private key with 161245
- Bob claims contract was EWM
- Judge computes:
 - $(42390^{154993} \bmod 262631)^{161245} \bmod 288329 = 042212$, which is EWM
 - Verified; now Alice is toast

Preventing Attack #2

- Obvious thought: instead of encrypting message and then signing it, sign the message and then encrypt it
 - May not work due to surreptitious forwarding attack
 - Idea: Alice sends Cathy an encrypted signed message; Cathy decipheres it, re-enciphers it with Bob's public key, and then sends message and signature to Bob – now Bob thinks the message came from Alice (right) and was intended for him (wrong)
- Several ways to solve this:
 - Put sender and recipient in the message; changing recipient invalidates signature
 - Sign message, encrypt it, then sign the result

El Gamal Digital Signature

- Relies on discrete log problem
 - Choose p prime, $g, d < p$; compute $y = g^d \bmod p$
- Public key: (y, g, p) ; private key: d
- To sign contract m :
 - Choose k relatively prime to $p-1$, and not yet used
 - Compute $a = g^k \bmod p$
 - Find b such that $m = (da + kb) \bmod p-1$
 - Signature is (a, b)
- To validate, check that
 - $y^a a^b \bmod p = g^m \bmod p$

Example

- Alice chooses $p = 262643$, $g = 9563$, $d = 3632$, giving $y = 274598$
- Alice wants to send Bob signed contract PUP (152015)
 - Chooses $k = 601$ (relatively prime to 262642)
 - This gives $a = g^k \bmod p = 9563^{601} \bmod 262643 = 202897$
 - Then solving $152015 = (3632 \times 202897 + 601b) \bmod 262643$ gives $b = 225835$
 - Alice sends Bob message $m = 152015$ and signature $(a,b) = (202897, 225835)$
- Bob verifies signature: $g^m \bmod p = 9563^{152015} \bmod 262643 = 157499$
and $y^a a^b \bmod p = 274598^{202897} 202897^{225835} \bmod 262643 = 157499$
 - They match, so Alice signed

Attack

- Eve learns k , corresponding message m , and signature (a, b)
 - Extended Euclidean Algorithm gives d , the private key
- Example from above: Eve learned Alice signed last message with $k = 5$
 $m = (da + kb) \bmod p-1 \Rightarrow 152015 = (202897d + 601 \times 225835) \bmod 262642$
giving Alice's private key $d = 3632$

Notation

- $X \rightarrow Y : \{ Z \parallel W \} k_{X,Y}$
 - X sends Y the message produced by concatenating Z and W enciphered by key $k_{X,Y}$, which is shared by users X and Y
- $A \rightarrow T : \{ Z \} k_A \parallel \{ W \} k_{A,T}$
 - A sends T a message consisting of the concatenation of Z enciphered using k_A , A 's key, and W enciphered using $k_{A,T}$, the key shared by A and T
- r_1, r_2 nonces (nonrepeating random numbers)

Session, Interchange Keys

- Alice wants to send a message m to Bob
 - Assume public key encryption
 - Alice generates a random cryptographic key k_s and uses it to encipher m
 - To be used for this message *only*
 - Called a *session key*
 - She enciphers k_s with Bob's public key k_B
 - k_B enciphers all session keys Alice uses to communicate with Bob
 - Called an *interchange key*
 - Alice sends $\{ m \}_{k_s} \{ k_s \}_{k_B}$

Benefits

- Limits amount of traffic enciphered with single key
 - Standard practice, to decrease the amount of traffic an attacker can obtain
- Prevents some attacks
 - Example: Alice will send Bob message that is either “BUY” or “SELL”. Eve computes possible ciphertexts $\{ \text{“BUY”} \} k_B$ and $\{ \text{“SELL”} \} k_B$. Eve intercepts enciphered message, compares, and gets plaintext at once

Key Exchange Algorithms

- Goal: Alice, Bob get shared key
 - Key cannot be sent in clear
 - Attacker can listen in
 - Key can be sent enciphered, or derived from exchanged data plus data not known to an eavesdropper
 - Alice, Bob may trust third party
 - All cryptosystems, protocols publicly known
 - Only secret data is the keys, ancillary information known only to Alice and Bob needed to derive keys
 - Anything transmitted is assumed known to attacker

Symmetric Key Exchange

- Bootstrap problem: how do Alice, Bob begin?
 - Alice can't send it to Bob in the clear!
- Assume trusted third party, Cathy
 - Alice and Cathy share secret key k_A
 - Bob and Cathy share secret key k_B
- Use this to exchange shared key k_s

Simple Protocol

Alice $\xrightarrow{\{\text{request for session key to Bob}\} k_A}$ Cathy

Alice $\xleftarrow{\{\ k_S \} k_A \ || \ \{\ k_S \} k_B}$ Cathy

Alice $\xrightarrow{\{\ k_S \} k_B}$ Bob

Problems

- How does Bob know he is talking to Alice?
 - Replay attack: Eve records message from Alice to Bob, later replays it; Bob may think he's talking to Alice, but he isn't
 - Session key reuse: Eve replays message from Alice to Bob, so Bob re-uses session key
- Protocols must provide authentication and defense against replay