Outline: Crypto Part 2

• **Symmetric Key Cryptography**
  • Hash functions and MACs
  • Authenticated Encryption (and Block Ciphers)

• **Asymmetric (Public) Key Cryptography**
  • Public-Key Encryption
  • Digital Signatures

• **Hybrid Encryption:** Building secure channels from scratch*
Recall: Integrity (Message Authentication Codes)

Provide integrity by attaching a MAC (tag $T$) to each message ($D$), where the tag is:
1) Short string that validates the message $D$
2) Unforgeable (can’t create) without knowing secret key $K$

$$T \leftarrow \text{MAC}_K(D)$$

Check: $T = \text{MAC}_K(D)$?
Definition: A hash function is a deterministic function \( H(\ldots) \) that maps arbitrary strings to fixed-length outputs.

Properties of a secure hash function:

1. One-way function: given \( H(M) \), can’t find \( M \)
2. Collision resistance: can’t find \( M \neq M' \) such that \( H(M) = H(M') \)
3. Second-preimage resistance: given \( H(M) \), can’t find \( M' \) s.t. \( H(M') = H(M) \)

- Note: Very different from hashes used in data structures!
Why are hash collisions bad?

The binary should hash to 3477a3498234f.

MD5(0100001001) = 3477a3498234f

Hashes to 3477a3498234f, so let's install!
# Practical Hash Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Output Len (bits)</th>
<th>Broken?</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>1993</td>
<td>128</td>
<td>Super-duper broken</td>
</tr>
<tr>
<td>SHA-1</td>
<td>1994</td>
<td>160</td>
<td>Yes</td>
</tr>
<tr>
<td>SHA-2 (SHA-256)</td>
<td>1999</td>
<td>256</td>
<td>No</td>
</tr>
<tr>
<td>SHA-2 (SHA-512)</td>
<td>2009</td>
<td>512</td>
<td>No</td>
</tr>
<tr>
<td>SHA-3</td>
<td>2019</td>
<td>&gt;=224</td>
<td>No</td>
</tr>
</tbody>
</table>
Hash Functions are **not** MACs

Both functions map long inputs to short outputs… but hash func’s do not use a key: Attackers can compute hash of any message they want (not unforgeable)

**Intuition**: a MAC is like a hash function, that only the someone w/ the key can evaluate.
Building MACs from Hash Functions

**Goal:** Build a secure MAC out of a good hash function.

**Construction:** \( \text{MAC}(K, D) = H(K \| D) \)  

*Warning: Broken*

- Totally insecure if \( H = \text{MD5}, \text{SHA1}, \text{SHA-256}, \text{SHA-512} \)

**Secure MAC:** Use standard HMAC function  
\[
\text{MAC}(K, D) = H( K \oplus \text{opad} \| H( K \oplus \text{ipad} \| D ) )
\]

NEVER Design your own crypto algorithms, always use standard libraries!
Length Extension Attack on Insecure MACs

Construction: $\text{MAC}(K, D) = H(K \ || \ D)$

Adversary goal: Find new message $D'$ and a valid tag $T'$ for $D'$

In other words: Given $T=H(K \ || \ D)$, find $T'=H(K \ || \ D')$ without knowing $K$.

- Attack: Can craft $D' = D \ || \ XYZ$, with some string $XYZ$ that consists of (1) substr that attacker can freely choose and (2) substr to make attack work

In Assignment 3: Break this construction!
Outline: Crypto Part 2

• **Symmetric Key Cryptography**
  • Hash functions and MACs
  • Authenticated Encryption (and Block Ciphers)

• **Asymmetric (Public) Key Cryptography**
  • Public-Key Encryption
  • Digital Signatures

• **Hybrid Encryption**: Building secure channels from scratch*
## Four Cryptography Problems / Tools

<table>
<thead>
<tr>
<th>Confidentiality</th>
<th>Authenticity/Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symmetric Encryption</strong></td>
<td><strong>Message Authentication Code (MAC)</strong></td>
</tr>
<tr>
<td>Security: Ciphertext reveals <em>nothing</em> about plaintext message</td>
<td>Security: Tag for new msg is impossible to compute without secret key</td>
</tr>
</tbody>
</table>

- **Pre-shared key?**
  - Yes
  - No

- **Security Goal**
  - Breached
  - Confidentiality

- **Security**
  - Ciphertext reveals *nothing* about plaintext message
  - Tag for new msg is impossible to compute without secret key
Authenticated Encryption algorithms provide both confidentiality and integrity.

- One approach: Built using a good stream cipher and a MAC.
  - Ex: Salsa20 with HMAC-SHA2

- Best solution: Use ready-made Authenticated Encryption
  - Ex: AES-GCM is the standard
Brief Detour: AES & Block Ciphers

Blockciphers: common crypto building block for solving many problems.

**Informal definition:** A blockcipher is essentially a substitution cipher with a very large alphabet and a very compact key.

Typical parameters:
Alphabet = \(\{0,1\}^{128}\)
Key length = 16 bytes.

Can build many higher-level protocols from a good blockcipher.
Advanced Encryption Standard (AES)

- NIST ran competition to develop standard encryption algorithms in 1997
- Several submissions, *Rijndael* chosen and standardized

Rijmen and Daemen
- Block length $n = 128$
- Key length $k = 128, 192, 256$
- 10 rounds of “substitution-permutation network”
- Break msg $M$ into blocks and encrypt each block

\[
M \oplus P_1 \oplus K_1 \oplus P_2 \oplus K_2 \oplus P_3 \oplus \ldots
\]
Blockcipher Security (Confidentiality)

- AES is thought to be a good “Pseudorandom Permutation”

- Outputs all look random and independent, even when inputs are maliciously controlled.
- Formal definition in CS284.
Advanced Encryption Standard (AES)

- AES is now the gold standard blockcipher
  - Very fast; Intel & AMD CPU chips have built-in AES instructions

- AES has different *modes* of operation
  - Some common modes: ECB, CTR, CBC, GCM
  - ECB: do not use – insecure!!
  - CTR & CBC do not provide integrity
  - GCM: authenticated encryption (both conf & integrity)
Outline: Crypto Part 2

• **Symmetric Key Cryptography**
  • Hash functions and MACs
  • Authenticated Encryption (and Block Ciphers)

• **Asymmetric (Public) Key Cryptography**
  • Public-Key Encryption
  • Digital Signatures

• **Hybrid Encryption:** Building secure channels from scratch*
**Motivation:** If two people do not have a pre-shared secret key, can they send private messages in the presence of an attacker?

<table>
<thead>
<tr>
<th>Security Goal</th>
<th>Confidentiality</th>
<th>Authenticity/Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-shared key?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes (“Symmetric”)</td>
<td>Symmetric Encryption</td>
<td>Message Authentication Code (MAC)</td>
</tr>
<tr>
<td>No (“Asymmetric”)</td>
<td>Public-Key Encryption</td>
<td>Digital Signatures</td>
</tr>
</tbody>
</table>
Why do we need Public-Key Cryptography?

**Motivation:** If two people do not have a pre-shared secret key, can they send private messages in the presence of an attacker?

Formally impossible (in some sense): No difference between receiver and adversary.
Why do we need Public-Key Cryptography?

**Motivation:** If two people do **not** have a pre-shared secret key, can they send private messages in the presence of an attacker?

- **Diffie and Hellman in 1976:** Yes!
  - Turing Award, 2015

- **Rivest, Shamir, Adleman in 1978:** Yes, differently!
  - Turing Award, 2002

- **Cocks, Ellis, Williamson in 1969, at GCHQ:** Yes...
A public-key encryption scheme consists of three algorithms: KeyGen, Encrypt, and Decrypt.

**KeyGen**
- Outputs two keys. PK published openly, and SK kept secret.

**Encrypt (PK, M)**
- Uses PK and M to produce a ciphertext C.

**Decrypt (SK, C)**
- Uses SK and C to recover M.
Public-Key Encryption

**Goal:** Passive Attacker, knows algorithm implementations (Enc, Dec) and PK, but the ciphertext $C$ reveals nothing about the plaintext message $M$

- Attacker might also have partial knowledge, e.g., other $(M^*, C^*)$ pairs
- Encryption (symmetric too) not even allowed to reveal if a message repeated!
Public Key Encryption Schemes: RSA

Key Generation:

- Pick \( p \) and \( q \) be large random prime numbers (around \( 2^{1024} \))
- Compute \( N \leftarrow pq \)
- Set \( e \) to a default value (\( e = 3 \) and \( e = 65537 \) are common)
- Compute \( d \) such that \( ed = 1 \mod (p - 1)(q - 1) \)
- Output:
  - Public key \( pk = (N, e) \)
  - Secret key \( sk = (N, d) \)

Example:

- \( p = 5, q = 11, N = 55 \)
- \( e = 3, d = 27 \)
Plain RSA Encryption

\( PK = (N, e) \quad SK = (N, d) \quad \text{where} \quad N = pq, \ ed = 1 \mod (\phi(N)) \)

Encryption & Decryption:

\[
\text{Enc}((N, e), x) = x^e \mod N \\
\text{Dec}((N, d), y) = y^d \mod N
\]

Note: Taking modular roots is believed to be computational hard

Using number theory from CMSC 27100, can show:

\[
\text{Dec(Enc}((N, e), x)) = (x^e)^d = x \mod N
\]

Never use directly as encryption!  

Warning: Broken
Best Known Attack on RSA: Factoring

- Factoring $N$ allows recovery of secret key… can compute $\phi(N) = (p - 1)(q - 1)$
- Challenges posted publicly by RSA Laboratories

<table>
<thead>
<tr>
<th>Bit-length of $N$</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>1993</td>
</tr>
<tr>
<td>478</td>
<td>1994</td>
</tr>
<tr>
<td>515</td>
<td>1999</td>
</tr>
<tr>
<td>768</td>
<td>2009</td>
</tr>
<tr>
<td>795</td>
<td>2019</td>
</tr>
</tbody>
</table>

- Recommended bit-length today: 2048 or greater
- Note that fast factoring algorithms force such a large key.
  - 512-bit $N$ defeats naive factoring
Outline: Crypto Part 2

• **Symmetric Key Cryptography**
  • Hash functions and MACs
  • Authenticated Encryption (and Block Ciphers)

• **Asymmetric (Public) Key Cryptography**
  • Public-Key Encryption
  • Digital Signatures

• **Hybrid Encryption**: Building secure channels from scratch*
A digital signature scheme consists of three algorithms:

- **KeyGen**: Outputs two keys. PK published openly, and SK kept secret.
- **Sign**: Uses SK to produce a “signature” σ on M.
- **Verify**: Uses PK to check if signature σ is valid for M.
Scheme satisfies **unforgeability** if an Adversary (who knows $PK$) cannot to fool Bob into accepting $(M', \sigma')$ that Alice has not sent.
“Plain” RSA Signature with No Encoding

KeyGen is same as regular RSA:

\[ PK = (N, e) \quad SK = (N, d) \quad \text{where} \quad N = pq, ed = 1mod\phi(N) \]

\( e = 3 \) is common for fast verification.

\[ \text{Sign}((N, d), M) = M^d modN \]

\[ \text{Verify}((N, e), M, \sigma): \sigma^e = MmodN? \]
“Plain” RSA Weaknesses

Assume $e=3$.

Sign($((N,d), M) = M^d \mod N$

Verify($((N,3), M, \sigma): \sigma^3 = M \mod N$)

To forge a signature on message $M'$: Find number $\sigma'$ such that $(\sigma')^3 = M' \mod N$

**Trivial Attack:** Easy to forge signature for $M'=1$: Take $\sigma'=1$:

$$(\sigma')^3 = 1^3 = 1 = M' \mod N$$

**Cube-M weakness:** For any $M'$ that is a perfect cube, it is easy to forge.

Attack: Signature $\sigma' = \sqrt[3]{M'}$, i.e. the usual cube root of $M'$

**Example:** To forge on $M'=8$, which is a perfect cube, set $\sigma'=2$.

$$(\sigma')^3 = 2^3 = 8 = M' \mod N$$

(Intuition: If cubing does not “wrap modulo $N$”, then it is easy to un-do.)
More “Plain” RSA Weaknesses

Assume $e=3$.

Sign($((N,d),M)$) = $M^d \mod N$

Verify($((N,3),M,\sigma)$): $\sigma^3 = M \mod N$?

To forge a signature on message $M'$: Find number $\sigma'$ such that $(\sigma')^3 = M' \mod N$.

**Malleability weakness:** If $\sigma$ is a valid signature for $M$, then it is easy to forge a signature for new msg $M' = (8M \mod N)$.

Given $(M, \sigma)$, compute forgery $(M', \sigma')$ as

$$M' = (8 \times M \mod N), \text{ and } \sigma' = (2 \times \sigma \mod N)$$

This is a valid pair because: Verify($((N,3), M', \sigma')$) checks:

$$(\sigma')^3 = (2 \times \sigma \mod N)^3 = (2^3 \times \sigma^3 \mod N) = (2^3 \times M \mod N) = 8M \mod N$$

$\sigma^3 = M \mod N$ because $\sigma$ is valid sig. on $M$. 

\[\checkmark\]
Secure RSA Signatures with Encodings

\[ PK = (N, e) \quad SK = (N, d) \quad \text{where} \quad N = pq, ed = 1 \mod \phi(N) \]

\[ \text{Sign}((N, d), M) = (\text{encode}(M))^d \mod N \]

\[ \text{Verify}((N, e), M, \sigma): \sigma^e = \text{encode}(M) \mod N? \]

encode maps bit strings to numbers between 0 and N

**Encoding must be chosen with extreme care.**

Broken
Authentication via Digital Signatures

- “Challenge – Response” Protocol
- This and similar ideas used in SSH, TLS, etc.

Blase
PK,SK

Hey it’s me, your user

Really? Prove it by signing \( r \)

\( \sigma = \text{Sign}(SK, r) \)

Server

Pick random bytes \( r \)

Blase’s PK

Verify(PK, r, \( \sigma \))?
Digital Signature Summary

As with all crypto schemes:
do not build your own signature schemes!

- Plain RSA signatures are very broken!
- Several secure RSA options in widely deployed libraries available:
  - PKCS#1 v.1.5 is widely used, in TLS, and fine if implemented correctly
  - Full-Domain Hash and PSS should be preferred
- There are also other signature schemes that aren’t based on RSA (e.g., DSA/ECDSA)
Outline: Crypto Part 2

• **Symmetric Key Cryptography**
  • Hash functions and MACs
  • Authenticated Encryption (and Block Ciphers)

• **Asymmetric (Public) Key Cryptography**
  • Public-Key Encryption
  • Digital Signatures

• **Hybrid Encryption**: Building secure channels from scratch*
Why not use asymmetric crypto for everything?

Answer

Symmetric key crypto algorithms are MUCH faster

<table>
<thead>
<tr>
<th>Security Goal</th>
<th>Confidentiality</th>
<th>Authenticity/Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-shared key?</td>
<td>Symmetric Encryption</td>
<td>Message Authentication Code (MAC)</td>
</tr>
<tr>
<td>Yes (&quot;Symmetric&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No (&quot;Asymmetric&quot;)</td>
<td>Public-Key Encryption</td>
<td>Digital Signatures</td>
</tr>
</tbody>
</table>
Hybrid Encryption: Real-world Secure Channels

Strategy:

1. Alice & Bob use a key exchange protocol to share their secret key(s)

2. Alice & Bob then use symmetric authenticated encryption (fast) for all their msg’s

```
Key Exchange

AES-GCM(K, M₁)

AES-GCM(K, M₂)

AES-GCM(K, M₃)

...
```
Key Exchange Protocols

Options

1. Use public-key crypto algorithms (RSA encryption & signatures)

2. Use dedicated key exchange algorithms (Diffie-Hellman):
   Faster & recommended approach (e.g., TLS, SSH)
Key Exchange using Public Key Crypto

**Goal:** Establish secret key $K$ to use with Authenticated Encryption.

$(\text{KeyGen}, \text{Enc}, \text{Dec})$ is a public-key encryption scheme.

- Pick random AES key $K$
- $C = \text{Enc}(PK, K)$
- $K$ is the message
- $K \leftarrow \text{Dec}(SK, C)$
- $\text{AES-GCM}(K, M_1)$
Key Exchange using Public Key Crypto

Q: How do we make this secure against an active attacker?
A: Certificates w/ Signatures (Next Lecture)

Pick random AES key $K$

$PK, SK \leftarrow \text{Keygen}$

$K \leftarrow \text{Dec}(SK, C)$

$C = \text{Enc}(PK', K)$

$C = \text{Enc}(PK, K)$

AES-GCM($K, M_1$)

$K$
The End