Symmetric Encryption 2: Integrity

With material from Dave Levin, Jon Katz, David Brumley
• Schedule updates
• Exams/scores
• No office hours today
  • Tomorrow at 4 instead
• P1, P2
Summing up (so far)

• Computational security
  • Adversary receives encryption of either $m_0$ or $m_1$
  • Can’t do better than guess which it is

• Secure PRF: Adversary can’t distinguish between PRF and actual random function

• Block ciphers: Secure when used properly (IVs!)
  • Multiple encryption modes
Message integrity and authentication
• Privacy and integrity are *orthogonal*
  
  • Up to now we’ve had privacy without integrity
  
  • Now we will do integrity without privacy
  
  • And later, both at once

• Reminder: Goal is to *detect* tampering
  
  • Not to prevent it!
Goal: Integrity

Eve should not be able to alter \( m \) without detection.

message \( m \): “curiouser and curiouser!”

message \( m' \): “curious and curious?”

ERROR!
Message authentication (MAC)

\[ t = \text{Sign}(m, k_s) \]

\[ \text{Verify}(m, t, k_s) \neq \text{true} \]

Only someone who knows \( k_s \) could have sent the message!
Non-repudiation

- A special case of authentication
- Only Alice can have sent the message
  - Bob could not have made it up
  - Alice cannot effectively deny having sent it
- Why would you want this?
MAC definition

- $t : T = S(k, m)$
- $V(k, m, t) = \text{yes or no}$
- $V(k, m, S(k, m)) = \text{yes}$
Straw example #1: CRC

- CRC = cyclic redundancy check
  - Binary division gives short, deterministic “summary” of data

- $S(k,m) = CRC(m)$

- What’s wrong with this plan?
MAC security

• Alice sends message m with tag t

• Attacker’s power: Chosen plaintext
  • Can observe correct \((m_i, t_i)\) pairs
  • Can use MAC oracle to get \(t_x\) for chosen \(m_x\)

• Attacker’s goal: Generate some valid \(m’, t’\) for \(m’\) not previously seen
  • \(m’\) does not have to make sense!

• Secure if: \(\Pr[V(k, m’, t’) = \text{yes}]\) is very small

Called: Existential forgery
Replay attacks

• Does a MAC prevent a replay attack?
  • NO — Must be prevented at a higher level
    • Application-dependent scenario
    • Nonce, timestamp, etc. (more later)
Straw example #2: Block cipher

• Suppose message is exactly one block
• \( t = S(k,m) = E(k,m) \)
  • \( t \) is 128 bits long under AES
• Is this secure? Why?
Security sketch

- Since $E(k,m)$ is a secure block cipher, can conceptually replace $E(k,m)$ with a random permutation.
- Seeing $E(k,m_1) \ldots E(k,m_n)$ doesn’t help predict unseen $m_{n+1}$
- Probability of a random guess is $1/2^L$
  - $L = \text{length of output tag (in bits)}$
  - Need to make sure $L$ is long enough!

But this only works for tiny messages!
Encrypted CBC (ECBC)

IV = 0

\[ m_1 \rightarrow E \quad m_2 \rightarrow E \quad m_3 \rightarrow E \quad m_4 \rightarrow E \]

Using key $k_0$

Using key $k_1$

Verify: Same algorithm as signing
ECBC vs. CBC

- Output only one block instead of many
  - Don’t need to recover the plaintext
  - AES => $2^{-128}$ chance of guessing

- We used two keys
  - Necessary to prevent existential forgery

- Both require serial computation
Why two keys?

- Attacker requests tag for message m (m₁ .. mₙ)
  - Get corresponding tag t = c[n]

- Attacker creates message m’ (one block long)
  - Request tag t’ for (t XOR m’)

- Resulting t’ is valid for m || m’

Uh oh.
MACs with Hashes
Hash functions

• A pseudorandom, one-way function
  • *Does not* require a key

• $H(m) = h$
  • Input $m =$ *pre-image*, can be arbitrary length
  • Output $h =$ *digest* or *hash*, fixed small length

• Generally very fast to compute
Cryptographic hash

• **Pre-image resistance**:
  • Given $H(m)$, it’s hard to find $m$

• **Collision resistant**:
  • Given $H(m)$, it’s hard to find $m'$ s.t.
    • $m' \neq m$
    • $H(m') = H(m)$
  • **Even more**: $\Pr[\text{any bit matching}] = 1/2$
Example hash functions

- MD5: Known collision attacks, still frequently used
- SHA-1, SHA-256, SHA-512, etc.
  - SHA1 is theoretically broken
- New SHA-3 (224, 256, 385, 512)
  - Public contest 2007-2012
  - Officially standardized August 2015
Hash-MAC

- Most widely used MAC on internet
- General idea: hash then PRF (short MAC)
  - Translate arbitrary message into one block
  - Works if H and E are both secure
Aside: Birthday paradox

• How likely 2 people in a room share a birthday?
  • Pr > 50% with 23 people!
  • Why? There are n^2 different pairs

• With X possibility space and n samples:
  • Pr[x_i = x_j] ~ 50% when n = X^{1/2}

• Upshot: May need to change keys frequently
Integrity vs. Authentication

• Recall: What is the difference?
  • Don’t forget non-repudiation

• Do symmetric MACs like ECBC and Hash-Mac give one, or both? Which?

• Problem: *More than one person* knows the key
Authenticated Encryption
• Previously:
  • Privacy / secrecy
  • Integrity
• Now: Both at once
Ciphertext integrity

• Maintain semantic secrecy under CPA attack

• Attacker **cannot create a new ciphertext** that decrypts properly!

Encryption oracle

Decrypt c or error

Ciphertext integrity IFF prob. of decryption without error is very small
CCA revisited

Eve can get a cipher text decrypted

c = E(k,m)

c' = forged cipher text
CCA game

- Attacker gets encryption oracle + decryption oracle
  - (Encryption oracle not shown)

**Challenge:**
Choose \( b = x \) or \( y \) at uniform random

**Encryption oracle**

\[
c_1 \ldots c_n
\]

\[
m_1 = D(k, c_1) \ldots m_n
\]

\[
m_x \text{ and } m_y \text{ (not in } m_1 \ldots m_n)\]

\[
c = E(k, m_b)
\]

**Eve:** (polynomial time)

Eve’s job: Guess whether \( x \) or \( y \) was picked. CCA-secure IFF no better than guessing
CBC is not CCA-secure

Challenge: Choose $b = x$ or $y$ at uniform random

$m_x$ and $m_y$

$|m_x| = |m_y| = 1$ blk

$c = E(k, m_b) = IV \| c[0]$

$c' = (IV \text{ xor } 1) \| c[0]$

$m' = D(k, c') = m_b \text{ xor } 1$

Decryption oracle

Eve

 Uh oh.
Ciphertext integrity (aka authenticated encryption) can protect against CCA!

Because only someone who knows $k$ can send a message that will decrypt properly.
Auth. Encr. limitations

- Does not protect against replay
- Does not protect against e.g. timing attacks
Constructing authenticated encryption
Three basic options

Can you guess?

- Encrypt and MAC
- MAC then encrypt
- Encrypt then MAC
Encrypt and MAC

- Send $E(m) \| \text{MAC}(m)$
- This is not secure b/c MAC may leak information about the message
  - Secrecy is not a MAC property
MAC then encrypt

- Send $E(m \parallel \text{MAC}(m))$
- This can be insecure in some combinations
  - Always follow standards!
Encrypt then MAC

- Send $E(m) \parallel \text{Mac}(E(m))$

- This is **always secure**! Intuition:
  - MAC reveals only info about ciphertext (OK)
  - MAC ensures ciphertext has not been tampered

\[\text{m} \xrightarrow{E_{k_E}} \text{c} \xrightarrow{S_{k_I}} \text{c t}\]
Key exchange
• Up to now, we have assumed Alice and Bob share a secret key

• How did that happen?

• How does this scale to many users?
One solution: Trusted third party (TTP)

- TTP is a bottleneck for every message
- TTP must be online at all times
- TTP can read every message
- Does not solve bootstrapping problem
Session keys and tickets

Used for Kerberos

- $k_{AT}$
- $K_{AB, \text{Ticket}}$
- $k_{AB}$
- “Hi”
- “Bob”
- $\text{Ticket} = E(K_{BT, \text{Alice||Bob||}k_{AB}})$ (fresh $k_{AB}$)

- TTP is a bottleneck for every message
- TTP must be online at all times
- TTP can read every message
- Does not solve bootstrapping problem
A (very) little number theory

- $p \Rightarrow$ a random, large prime number
- $g \Rightarrow$ a generator for $p$: $1 < g < p$
  - For all $k$ between 1 and $p$:
  - There is some $i$, s.t. $k = g^i \mod p$
- This is called discrete log
  - Easy: Given $p$, $g$, $x$, compute $y = g^x \mod p$
  - Believed hard: Given $p$, $g$, $y$, compute $x$
  - Candidate one-way function!
### Generator examples: $p = 7$

#### $g = 3$

<table>
<thead>
<tr>
<th>$3^1 \mod 7$</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3^2 \mod 7$</td>
<td>2</td>
</tr>
<tr>
<td>$3^3 \mod 7$</td>
<td>6</td>
</tr>
<tr>
<td>$3^4 \mod 7$</td>
<td>4</td>
</tr>
<tr>
<td>$3^5 \mod 7$</td>
<td>5</td>
</tr>
</tbody>
</table>
| $3^6 \mod 7$ | 1  | **YES**

#### $g = 2$

<table>
<thead>
<tr>
<th>$2^1 \mod 7$</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^2 \mod 7$</td>
<td>4</td>
</tr>
<tr>
<td>$2^3 \mod 7$</td>
<td>1</td>
</tr>
<tr>
<td>$2^4 \mod 7$</td>
<td>2</td>
</tr>
<tr>
<td>$2^5 \mod 7$</td>
<td>4</td>
</tr>
</tbody>
</table>
| $2^6 \mod 7$ | 1  | **NO**
Diffie-Hellman protocol

1. Compute $A = g^x \mod p$
2. $p$, $g$, $A$
3. Compute $B = g^y \mod p$
4. $B$
5. $k = B^x \mod p$
6. $k = A^y \mod p$

Successful secret key exchange!