Legal Notice

The Zoom session for this class will be recorded and made available asynchronously on Canvas to registered students.
Announcements

1. HW 2 is due Wednesday!

2. HW 3 is out, due before class in 1.5 weeks, April 22.
Last time: Chosen plaintext attacks

This time: Fun with CBC mode, malleability and message integrity
Last time: Cipher block chaining (CBC) mode

1. IV has same length as block length.
2. \( c_i = \text{Enc}_k(c_{i-1} \oplus m_i) \)
3. Output \((IV, c_0, c_1, c_2, \ldots)\).

IV should be random.

CBC mode is CPA-secure, but suffers from implementation vulnerabilities.
Chosen Plaintext Attack against CBC mode

Assumption: \( E \) \text{ is a block cipher LPRP.} 

No information if \( p \) is wrong.

Rogaway 1995: CBC mode not secure against chosen plaintext attacks if attacker can see IV or previous block before choosing \( m \).

1. Attacker observes \( c_1, c_2, \ldots, c_{i-1}, c_i, \ldots, c_j. \)
2. Attacker wants to distinguish \( c_i \), stream is currently at \( c_j \).
3. Attacker guesses \( c_i \) plaintext is \( p \).
4. Attacker causes victim to encrypt \( c_j \oplus c_{i-1} \oplus p \).
5. Victim sends \( \text{Enc}_k(c_j \oplus (c_j \oplus c_{i-1} \oplus p)) = \text{Enc}_k(c_{i-1} \oplus p) \).
6. Attacker compares \( \text{Enc}_k(c_{i-1} \oplus p) \) to \( c_i \), match if \( p \) correct.

A slightly modified attack game where \( C \) chains encryptions.
Chosen Plaintext Attacks against CBC mode in practice

- Dai 2002: SSH2 chains ciphertext between ciphertext, server
- SSLv3, TLS 1.0 also

128 bits is a lot to brute force in one block
- In many cases, know a lot of plaintext

How to request encryptions?
- Javascript, Java can make cookie-bearing requests across domains

Duong, Rizzo 2011: BEAST (Browser Exploit Against SSL/TLS)
- Pad plaintexts to 1 unknown text byte per block
Q: How do you encrypt odd-length messages with a fixed-length block cipher?
A: Pad message somehow.

Example: PKCS 7 padding.
- If message is \( b \) bytes short of 128-bit block, append \( b \) bytes \( b b b ... b \) to make block multiple of 128.
- Decryptor checks and strips off padding.

Q: What do you do if padding is incorrect?
A: Throw an error?
Q: How do you encrypt odd-length messages with a fixed-length block cipher?
A: Pad message somehow.

**Example:** PKCS 7 padding.
- If message is $b$ bytes short of 128-bit block, append $b$ bytes $bbb \ldots b$ to make block multiple of 128.
- Decryptor checks and strips off padding.
Q: How do you encrypt odd-length messages with a fixed-length block cipher?
A: Pad message somehow.

Example: PKCS 7 padding.
• If message is $b$ bytes short of 128-bit block, append $b$ bytes $bbb\ldots b$ to make block multiple of 128.
• Decryptor checks and strips off padding.

Q: What do you do if padding is incorrect?
A: Throw an error?
Padding oracle attacks

Vaudenay 2002: CBC mode encryption is insecure if attacker can distinguish bad from good message padding

Padding Oracle Attack
Want to decrypt \((IV, c_1, \ldots, c_n) = Enc_k(m_1, \ldots, m_n)\).

Have oracle that given \((IV', c'_1, \ldots, c'_\ell)\) will:

1. Compute \((m'_1, \ldots, m'_\ell) = Dec_k(IV, c'_1, \ldots, c'_\ell)\)
2. Return

\[
\begin{cases}
\text{valid} & \text{if } m'_\ell \text{ ends in valid padding} \\
\text{invalid} & \text{if } m'_\ell \text{ doesn’t end in valid padding}
\end{cases}
\]
Vaudenay CBC padding oracle attack

1. Attacker sends \((IV \oplus 00 \ldots 00t, c_1) = F_k(IV \oplus m_1 \oplus 00 \ldots 00t)\) to decryption padding oracle.

\[
\begin{array}{ccc}
IV & \rightarrow & \Theta \rightarrow \frac{\text{valid if } m_1 \oplus 000 \ldots 0t = \ldots 1(\text{most likely})}{\text{invalid otherwise}} \\
& & \{0, 1, \ldots, 2^{t-1}\} \\
\end{array}
\]

oracle returns

2. Try all 256 values of \(t\).
3. Good probability of learning value
4. Iterate for successive bytes of ciphertext.
5. \(256n\) query complexity to learn \(n\) bytes of plaintext.
Message Malleability and Integrity

Independent of whether messages remain confidential, we want to ensure that they can't be modified by a third party in transit.
Message Authentication Codes

Solution: Add a special tag to ensure integrity of the message.
Cryptographic and non-cryptographic integrity checking

Non-cryptographic protocols also often include integrity checks:

- Ethernet uses CRC32, a 32-bit checksum
- TCP has a 16-bit checksum

These algorithms protect against *random* errors, but they are public and do not include keys so can be forged by a malicious adversary.
Cryptographic and non-cryptographic integrity checking

Non-cryptographic protocols also often include integrity checks:

- Ethernet uses CRC32, a 32-bit checksum
- TCP has a 16-bit checksum

These algorithms protect against random errors, but they are public and do not include keys so can be forged by a malicious adversary.

In order to protect against malicious forgery, cryptographic integrity checks must use a secret key.

Encryption does not suffice.  \[
\text{Enc}_{k}(m) = k \oplus m = c \\
\text{c \oplus garbage} \quad \text{Dec}_{k}(c \oplus \text{garbage}) = m \oplus g + x
\]
Message Authentication Codes

Definition

- Key generation algorithm generates $k$
- Tag generation: $\text{Mac}_k(m) \rightarrow t$
- Tag verification:

$$\text{Verify}_k(m, t) = \begin{cases} 
\text{accept} & \text{if tag is valid} \\
\text{reject} & \text{if tag is invalid}
\end{cases}$$

- Correctness: $\Pr[\text{Verify}_k(m, \text{Mac}_k(m)) = \text{accept}] = 1$
MAC Security: Existential MAC forgery

**Definition**
A MAC construction \((\text{Mac}, \text{Verify})\) is existentially unforgeable under a chosen-message attack if the probability that \(A\) wins is negligible.
“Strongly secure” MACs

A

oracle access to $\text{Mac}_k(\cdot)$

$m$

$\text{Mac}_k(m)$

$\text{C}$

generate random $k$

$(m, t)$

A wins if

- $\text{Verify}_k(m, t) = 1$
- $(m, t)$ not among signed pairs

This is equivalent to the previous definition for deterministic MACs. The adversary can query the target message but not return one of the responses.
Let $F$ be a PRF. Then we can construct a MAC as follows:

- **Key generation:** $k \in_R \{0, 1\}^n$
- **$\text{Mac}_k(m) = F_k(m)$**
- **$\text{Verify}_k(m, t) =$**
  \[
  \begin{cases} 
  \text{accept} & \text{if } F_k(m) = t \\
  \text{reject} & \text{otherwise}
  \end{cases}
  \]
Theorem

*The PRF MAC construction is secure.*

Proof.
Assume construction not secure, construct PRF distinguisher.

1. If $F$ is a truly random function, then
   \[ \Pr[A \text{ succeeds}] = 2^{-n} = \Pr[D(f) = 1] \]
2. If $F$ PRF, \( \Pr[D(F_k) = 1] = d > \text{negligible by assumption.} \)

\[
| \Pr[D(F_k) = 1] - \Pr[D(f) = 1] | = d - 2^{-n} > \text{negligible}
\]
CBC-MAC

Can use CBC construction to construct a MAC for arbitrary length messages.

- $k \in_R \{0, 1\}^n$
- Input $m = m_1 \ldots m_\ell$. To compute $\text{Mac}_k(m)$:
  1. $t_0 = 0^n$ (fixed, non-random value)
  2. For $i = 1, \ldots, \ell$ $t_i = F_k(t_{i-1} \oplus m_i)$
  3. Output last block $t_\ell$

Verify $k(m, t) = \begin{cases} 
\text{accept} & \text{if } \text{Mac}_k(m) = t \\
\text{reject} & \text{otherwise} 
\end{cases}$

Theorem
If $F$ is a secure PRF, then CBC-MAC is a secure fixed-length MAC for arbitrary-length messages
Chosen Ciphertext Attacks

CCA indistinguishability requirement:

- \( A \) can \( m_0, m_1 \)
- \( C = \text{Enc}_k(m_b) \) \( b \in \{0,1\} \)
- \( \text{oracle access to } \text{Enc}_k(\cdot) \)
- \( \text{Dec}_k(\cdot) \)
- \( \text{can't query } \text{Dec}_k(C) \)

\[ \begin{array}{c}
\rightarrow \\
\leftarrow \\
\rightarrow \quad b' \rightarrow \\
\end{array} \]

A succeeds if \( b' = b \)

Define "CCA-Secure" under a chosen ciphertext-attack

An efficient adversary \( A \)

\[ \Pr[A \text{ succeeds}] \leq \frac{1}{2} + \varepsilon \text{ negligible} \]

IND-CCA 1: "non-adaptive"
- Decryption oracle only queried prior to challenge ciphertext

IND-CCA 2: "adaptive"
- May make further calls to decryption oracle
How do you combine encryption and integrity checks?
(Prelude to authenticated encryption.)

Right answer in practice: Use a pre-defined authenticated encryption mode w/ AES.

For historical purposes:
Intuition: Use encryption for CPA-security + MAC to keep attacker from mauling ciphertext into something useful.

Options:
Encrypt-then-MAC?
\[ c = \text{Enc}_{k_e}(m) \quad t = \text{Mac}_{k_m}(m) \quad \text{send } (c, t) \]

Mac-then-encrypt?
\[ t = \text{Mac}_{k_m}(m) \quad c = \text{Enc}_{k_e}(m || t) \quad \text{send } c \]

Encrypt-then-Mac? **CCA Secure**
\[ c = \text{Enc}_{k_e}(m) \quad t = \text{Mac}_{k_m}(c) \quad \text{send } (c, t) \]
Reminder, HW 2 is due before next lecture so we can talk about it in lecture!