AUTHENTICATED ENCRYPTION

VIET TUNG HOANG
1. AE and Its Security Definitions

2. Failed Ways to Build AE

3. Generic Compositions

4. Padding-Oracle Attack on SSL/TLS
Motivation: Challenge-Response Revisited

Question: Break this identification mechanism if encryption is CTR.
Solution: Authenticated Encryption

Transfer $5 to account 12345

Privacy

Encryption scheme

Authenticated Encryption
Achieve both of these aims

Authenticity

MAC

K
Authenticated Encryption (AE)
Emerged ~ 2000

Begin with two realizations

1. Authenticity is routinely needed/assumed
2. “Standard” privacy mechanisms don’t provide it

Provide an easier-to-correctly-use abstraction boundary
AE Syntax

Key Gen

\[ \mathcal{K} \rightarrow K \]

Encrypt

\[ M \rightarrow \mathcal{E} \rightarrow C \]

\[ K \rightarrow \mathcal{E} \]

Decrypt

\[ C \rightarrow \mathcal{D} \rightarrow \begin{cases} M \text{ or } \bot \end{cases} \]

Decryption may reject invalid ciphertexts
Defining Security for AE

- Use Left-or-Right security for privacy

Auth $\mathcal{E}$

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize()</td>
<td>$K \leftarrow K$</td>
</tr>
<tr>
<td>Return $\mathcal{E}_K(M)$</td>
<td></td>
</tr>
<tr>
<td>$\text{Enc}(M)$</td>
<td>Return $(\mathcal{D}_K(C') \neq \bot)$</td>
</tr>
</tbody>
</table>

$\text{Adv}^{\text{auth}}_{\mathcal{T}}(A) = \text{Pr}[\text{Auth}^A_{\mathcal{E}} \Rightarrow 1]$
1. AE and Its Security Definitions

2. Failed Ways to Build AE

3. Generic Compositions

4. Padding-Oracle Attack on SSL/TLS
Plain Encryption Doesn’t Provide Authenticity

Question: Does CBC provide authenticity?

Answer: No, because any ciphertext has valid decryption
A Bad Fix: CBC with Redundancy

On decryption, verify the decrypted last block is zero.

**Question:** Break the authenticity of this scheme with a single Enc query
An Attack

$\text{Enc}$

$C_0$ $C_1$ $C_2$ $C_3$

$M_1$ $0^n$

$C_0$ $C_1$ $C_2$

$E_K$ $E_K$ $E_K$

$C_1$ $C_2$ $C_3$

$E_K^{-1}$ $E_K^{-1}$

$C_0$ $M_1$ $0^n$
Complex Redundancy Doesn’t Help

Some (unkeyed) “redundancy” function, such as checksum

The redundancy is verified upon decryption

**Question:** Break the authenticity of this scheme with a single Enc query
An Attack

\[ E_K \]

\[ h(M) \]

\[ h(M_1) \]

\[ E_K^{-1} \]

\[ C_0 \]

\[ C_1 \]

\[ C_2 \]

\[ C_3 \]
24-bit IV is a part of the ciphertext

A Case Study: WEP

Used in IEEE WiFi standard
Attack 1: Exploiting Short IV

Assume all messages are of the same length, and fairly long

Goal: recover at least one message
Attack 1: Exploiting Short IV

Aim for an IV collision

For 24-bit IV’s, how many ctx to wait for collision prob ≈ 0.5?
Attack 1: Exploiting Short IV

Same IV, can recover $M_1 \oplus M_2$
Attack 2: Chop-Chop Attack

Goal: recover the underlying message by exploiting Dec queries
Attack 2: Chop-Chop Attack
Illustrated Via A Simpler Variant of WEP

Example: $\text{Parity}(10011) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 1$
Attack 2: Chop-Chop Attack

Illustrated For 4-bit Message
Decryption CloseUp

Compare with $\text{Parity}(M_1 M_2 M_3)$
Exploit Decryption Response

If valid, $M_1 \oplus M_2 \oplus M_3 \oplus M_4 = 0$
Exploit Decryption Response

If invalid, \( M_1 \oplus M_2 \oplus M_3 \oplus M_4 = 1 \)
Exploit Decryption Response

Learn $M_1 \oplus M_2 \oplus M_3 \oplus M_4$
Exploit Decryption Even Further

Learn $M_1 \oplus M_2 \oplus M_3$
Solve A System of Linear Equations

\[
\begin{align*}
M_1 \oplus M_2 \oplus M_3 \oplus M_4 &= \square \\
M_1 \oplus M_2 \oplus M_3 &= \square \\
M_1 \oplus M_2 &= \square \\
M_1 &= \square
\end{align*}
\]
Agenda

1. AE and Its Security Definitions

2. Failed Ways to Build AE

3. Generic Compositions

4. Padding-Oracle Attack on SSL/TLS
Constructing AE: Generic Composition

A good PRF, such as
Encrypted CBC-MAC

Privacy-only encryption
(such as CTR/CBC)

Compose them to build AE

<table>
<thead>
<tr>
<th>Method</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt-and-MAC</td>
<td>SSH</td>
</tr>
<tr>
<td>MAC-then-Encrypt</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>Encrypt-then-MAC</td>
<td>IPSec</td>
</tr>
</tbody>
</table>
Encrypt-and-MAC: Simple Composition

No privacy: encrypting the same message results in the same tag
No authenticity if one can modify $C$ such that decryption is unchanged.

<table>
<thead>
<tr>
<th>Privacy</th>
<th>Authenticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

for some bad encryption scheme

$E_{K_e}$

$F_{K_m}$

$M$

$C$

$T$
Encrypt-and-MAC in SSH

$M$

Encode

$\text{len}(M) \parallel \text{len}(\text{pad})$

$M$  pad

CBC

$C$

Privacy | Authenticity
---|---
Yes | Yes

$F_{K_m}$

$T$
MAC-then-Encrypt

Privacy | Authenticity
---|---
Yes | No

for some bad encryption scheme

No authenticity if one can modify $C$ such that decryption is unchanged.
MAC-then-Encrypt in TLS

\[ M \xrightarrow{F_{K_m}} T \xrightarrow{CBC} C \]

<table>
<thead>
<tr>
<th>Privacy</th>
<th>Authenticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Encrypt-then-MAC

\[ M \rightarrow E_{K_e} \rightarrow C \]

\[ F_{K_m} \rightarrow T \]

<table>
<thead>
<tr>
<th>Privacy</th>
<th>Authenticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
A Common Pitfall in Implementing EtM

Happened in ISO 1972 standard, and in RNCryptor of iOS

Forget to feed IV into MAC

Break auth with one query
Agenda

1. AE and Its Security Definitions
2. Failed Ways to Build AE
3. Generic Compositions
4. Padding-Oracle Attack on SSL/TLS
The Padding-Oracle Attack

“Lucky Thirteen” attack snarfs cookies protected by SSL encryption
Exploit is the latest to subvert crypto used to secure Web transactions.

Meaner POODLE bug that bypasses TLS crypto bites 10 percent of websites
Some of the world’s leading sites are vulnerable to an easier, more simplified attack.

Researchers poke hole in custom crypto built for Amazon Web Services
Even when engineers do everything by the book, secure crypto is still hard.

New TLS encryption-busting attack also impacts the newer TLS 1.3
Researchers discover yet another Bleichenbacher attack variation (yawn!).
Attack Model: Chosen Prefix Secret Suffix

Goal: Recover $M$
This Model Is Realistic: Attacking SSL

cookie with bank.com

Encrypted communication via SSL

visit

attacker.com

bank.com
This Model Is Realistic: Attacking SSL

Encrypted communication via SSL

Request resource
/AA at bank.com

cookie with bank.com

attacker.com

Bank of America

bank.com
This Model Is Realistic: Attacking SSL

\[ M \]

GET /AA cookie: \( M \)

encrypted

attacker.com

Bank of America

bank.com
This Model Is Realistic: Attacking SSL

Enc oracle

GET /AA cookie: $M$

encrypted

Dec oracle

bank.com
Encryption In SSL: MAC-then-Encrypt

\[ M \xrightarrow{MAC} M_T \xrightarrow{padding} CBC \xrightarrow{\text{CBC}} C \]
Padding In SSL Encryption

block length is 16 bytes

Consider byte strings only

31 bytes 1 byte

24 bytes 7 bytes 1 byte

16 bytes 15 bytes 1 byte
The Attack in Action
Illustration For Two-block Message

Aim: Recover the message byte by byte
Recover Last Byte of First Block

Enc

\[ \mathcal{E}_K(M_1M_2) \]

\[ C_0 \quad C_1 \quad C_2 \quad C_3 \quad C_4 \]

Empty string
Recover Last Byte of First Block

\[ \begin{align*}
C_0 & \quad C_1 \\
C_2 & \quad C_3 \\
C_4 & \quad C_0 \quad C_1 \quad C_2 \quad C_3 \quad C_1
\end{align*} \]
CBC Decryption

To pass MAC check, want the last byte of $V$ to be 15

$V = M_1 \oplus C_0 \oplus C_3$

Pass with prob $\sim 1/256$
If valid, last byte of $V = M_1 \oplus C_0 \oplus C_3$ is 15

Learn last byte of $M_1$
Exploit Decryption Output

After $t$ attempts, succeed with prob $\sim 1 - (1 - 1/256)^t$ times

If invalid, restart
Patching Via Different Padding

Secure if implement properly
Careless Implementation Leads To Attack

- **Secure** if return a single error signal
- **Broken** if tell what kind of error it is.
Scanning For Vulnerable Implementations

"Given final block not properly padded"

Spring boot  Eclipse  PKCS12  Javax  Java  IntelliJ  Android Studio  Images  AES

About 16,000 results (0.32 seconds)

Stack Overflow
https://stackoverflow.com › questions › given-final-bl...

**Given final block not properly padded - java**

Nov 8, 2011 — BadPaddingException: **Given final block not properly padded**. Such issues can arise if a bad key is used during decryption.

7 answers  Top answer: If you try to decrypt PKCS5-padded data with the wrong key, and then ...
Implementation Is Hard: **Timing Leakage**

Source: Canvel et al, CRYPTO 2003
How To Attack
Illustration For Two-block Message

\[ \mathcal{E}_K(M_1M_2) \]
Recover Last Byte of Second Block

$C_0 \rightarrow C_{1} \rightarrow C_{2} \rightarrow C_{3} \rightarrow C_{4}$

$\text{Enc} \rightarrow \text{Dec}$

$C_0 \rightarrow R \rightarrow C_2$

random
CBC Decryption

\[ V = M_2 \oplus C_1 \oplus R \]

If \( V \) ends with a zero byte

Bad tag signal