Authenticated Encryption and
Cryptographic Network Protocols

David Brumley
dbrumley@cmu.edu
Carnegie Mellon University
Some Straw Men
TCP/IP (highly abstracted)

Source

packet

dest=80 data

TCP/IP Stack

data

Webserver (port = 80)

Bob (port = 25)

Destination Machine
Encrypted with CBC and random IV

encrypted packets with key $k$

Source

<table>
<thead>
<tr>
<th>IV</th>
<th>dest</th>
<th>msg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{V1}$</td>
<td>80</td>
<td>msg a</td>
</tr>
<tr>
<td>$I_{V2}$</td>
<td>25</td>
<td>msg b</td>
</tr>
</tbody>
</table>

Destination Machine

Webserver (port = 80)

Bob (port = 25)
Example Tampering Attack

Encrypted with CBC and random IV

Eve can change destination
(easy with CBC and rand IV)
Example Tampering Attack

Encrypted with CBC and random IV

Eve can change destination (easy with CBC and rand IV)
How?

CBC encryption:
\[ D(k, c[0]) \oplus IV_1 = \text{“dest=80”} \]

Attack:
\[ IV_2 = IV_1 \oplus 000...80 \oplus 000...25 \]

oxor out “80” and xor in “1026”
An Attack Using Only Network Access

**Example:**
Remote terminal app where each keystroke encrypted with CTR mode

![Diagram showing network attack](image)
An Attack Using Only Network Access

Example:
Remote terminal app where each keystroke encrypted with CTR mode

\[
\text{checksum}(\text{hdr}, d) = t \oplus \text{checksum}(\text{hdr}, d \oplus s)
\]

\[
\Rightarrow \quad \text{Even can find } d \text{ for many realistic checksums}^*
\]

* potentially not for TCP checksum
The Story So Far

**Confidentiality**: semantic security against a CPA attack
- Examples: Using CBC with a PRP, AES

**Integrity**: security against existential forgery
- Examples: CBC-MAC, NMAC, PMAC, HMAC

Now: security against *tampering*
- Integrity + Confidentiality!
The lesson

CPA security *cannot* guarantee secrecy under *active* attacks.

- Integrity Only
- Integrity + Secrecy
- Integrity + Secrecy

- Secure MAC
- Secure MAC + Secure Cipher
- *Authenticated Encryption*
Motivating Question: Which is Best?

Encryption Key = $K_E$; MAC key = $k_I$

Option 1: SSL (MAC-then-encrypt)

Option 2: IPsec (Encrypt-then-MAC)

Option 3: SSH (Encrypt-and-MAC)
Authenticated Encryption
An **authenticated encryption** system \((E,D)\) is a cipher where

As usual: \[E: K \times M \times N \rightarrow C\]

but \[D: K \times C \times N \rightarrow M \cup \{\perp\}\]

**Security**: the system must provide

– Semantic security under CPA attack, **and**

– **ciphertext integrity**. The attacker cannot create a new ciphertext that decrypts properly.
Ciphertext Integrity

For $b = \{0, 1\}$, define $\text{EXP}(0)$ and $\text{EXP}(1)$ as:

Def: $(E,D)$ has ciphertext integrity iff for all “efficient” $A$:

$$\text{Adv}_{\text{CI}}[A,I] = \Pr \left[ \text{Chal. outputs 1} \right] < \varepsilon$$
Authenticated Encryption

Def: cipher \((E,D)\) provides \textit{authenticated encryption (AE)} if it is

(1) semantically secure under CPA, and
(2) has ciphertext integrity

Counter-example: CBC with rand. IV does not provide AE

– \(D(k, \cdot)\) never outputs \(\bot\), hence adv. always wins ciphertext integrity game
Implication 1: Authenticity

Attacker cannot fool Bob into thinking a message was sent from Alice

\[
\begin{align*}
\text{Alice} & \quad m_1, \ldots, m_q \quad \text{Eve} \\
\quad k & \quad c_i = E(k, m_i) \quad c \\
\text{Bob} & \quad c \notin \{c_1, \ldots, c_q\}
\end{align*}
\]

⇒ if \( D(k, c) \neq \bot \) Bob guaranteed message is from someone who knows \( k \) (but could be a replay)
Implication 2

Authenticated encryption ⇒

Security against \textit{chosen ciphertext attack}
Chosen Ciphertext Attacks
Def: A CCA adversary has the capability to get ciphertexts of their choosing decrypted.

Don’t want them to learn \( m' \) ... or even just whether an ACK occurred.

Eve sees \( c \) and \( m \)
The Lunchtime CCA Attack

It’s Lunchtime!

Alice’s Computer

Encryption Program

k

Encrypted File 1

Encrypted File 2

21
The Lunchtime CCA Attack

Alice’s Computer

Eve’s Encrypted File 1

Eve’s Encrypted File 2

Encrypted File 1

Encrypted File 2

Eve

Encryption Program

k
802.11b WEP: how not to do it

\[
\text{IV} \xrightarrow{\text{PRG}(\text{IV} \parallel k)} \text{m} \parallel \text{CRC}(m) \xrightarrow{\oplus} \text{ciphertext}
\]
Active attacks

**Fact:** CRC is linear, i.e.
\[ \forall m, p: \; \text{CRC}(m \oplus p) = \text{CRC}(m) \oplus \text{F}(p) \]

WEP ciphertext:

```
IV | dest-port = 80 | data | CRC
---|----------------|------|-----
```

Attacker:

```
000...00...... XX......0000 | F(XX)
```

\[ XX = 25 \oplus 80 \]

Upon decryption CRC is valid, but ciphertext is changed !!
Chosen Ciphertext Security

Adversaries Power: both CPA and CCA
  – Can obtain the encryption of arbitrary messages
  – Can decrypt ciphertexts of his choice

Adversaries Goal: break semantic security
Let ENC = (E,D) over (K,M,C).
For b = \{0,1\}, define EXP(0) and EXP(1)

\[\text{Chal.} \quad k \leftarrow K\]

\begin{align*}
\text{for } i=1,\ldots,q: \\
&\text{(1) CPA query:} \\
&\quad m_{i,0}, m_{i,1} \in M: \quad |m_{i,0}| = |m_{i,1}| \\
&\quad c_i \leftarrow E(k, m_{i,b})
\end{align*}

\begin{align*}
&\text{(2) CCA query:} \\
&\quad c_i \in C: \quad c_i \notin \{c_1, \ldots, c_{i-1}\} \\
&\quad m_i \leftarrow D(k, c_i)
\end{align*}

\[b' \in \{0,1\}\]
CCA Game Definition

Let $\text{ENC} = (E, D)$ over $(K, M, C)$.
For $b = \{0,1\}$, define $\text{EXP}(0)$ and $\text{EXP}(1)$

$(1)$ CPA query:

$m_0, m_1 \in M : |m_0| = |m_1|$

$b \leftarrow K$

for $i=1,...,q$:

$
 \text{CPA query:} \\
 m_i \leftarrow E(k, m_i, b)$

$(2)$ CCA query:

$c_i \in C : c_i \notin \{c_1, ..., c_{i-1}\}$

$b' \in \{0,1\}$

$m_i \leftarrow D(k, c_i)$

$\text{ENC} = (E, D)$ is CCA secure iff

$\text{Adv}[A, \text{ENC}] = |\Pr[\text{Exp}(0) = 1] - \Pr[\text{Exp}(1) = 1]| < \epsilon$
Example: CBC is not CCA Secure

Chal. k←K

Adv.

m₀, m₁ : |m₀| = |m₁| = 1

c ← E(k, m_b) = (IV, c[0])
c' = (IV ⊕ 1, c[0])

D(k, c') = m_b ⊕ 1
Thm: Let \((E,D)\) be a cipher that provides AE. Then \((E,D)\) is CCA secure!

In particular, for any \(q\)-query eff. A there exist eff. \(B_1, B_2\) s.t.

\[
\text{Adv}_{\text{CCA}}[A,E] \leq 2q \cdot \text{Adv}_{\text{CI}}[B_1,E] + \text{Adv}_{\text{CPA}}[B_2,E]
\]

AE implies CCA security!
So What?

Authenticated encryption assures security against:

– A passive adversary (CPA security)
– An active adversary that can even decrypt some ciphertexts (CCA security)

Limitations:

– Does not protect against replay
– Assumes no other information other than message/ciphertext pairs can be learned.
  • Timing attacks out of scope
  • Power attacks out of scope
  • ...
AE Constructions

Cipher + MAC = security
History

Pre 2000: Crypto API’s provide *separate* MAC and encrypt primitives
  – Example: Microsoft Cryptographic Application Programming Interface (MS-CAPI) provided HMAC and CBC + IV
  – Every project had to combine primitives in their own way

2000: Authenticated Encryption
  – Bellare and Namprempre in Crypto, 2000
  – Katz and Yung in FSE, 2000
Motivating Question: Which is Best?

Encryption Key = $K_E$; MAC key = $k_I$

Option 1: SSL (MAC-then-encrypt)

Option 2: IPsec (Encrypt-then-MAC)

Option 3: SSH (Encrypt-and-MAC)
Theorems

Let (E,D) by a CPA secure cipher and (S,V) a MAC secure against existential forgery. Then:

1. Encrypt-then-MAC *always* provides authenticated encryption
2. MAC-then-encrypt *may* be insecure against CCA attacks
   – however, when (E,D) is rand-CTR mode or rand-CBC, MAC-then-encrypt provides authenticated encryption
Standards

GCM: CTR mode encryption then CW-MAC
CCM: CBC-MAC then CTR mode (802.11i)
EAX: CTR mode encryption then CMAC

All are nonce-based.

All support *Authenticated Encryption with Associated Data (AEAD).*
An example API (OpenSSL)

```c
int AES_GCM_Init(AES_GCM_CTX *ain, 
                 unsigned char *nonce,  unsigned long noncelen, 
                 unsigned char *key,    unsigned int klen )

int AES_GCM_EncryptUpdate(AES_GCM_CTX *a, 
                           unsigned char *aad,   unsigned long aadlen, 
                           unsigned char *data,  unsigned long datalen, 
                           unsigned char *out,   unsigned long *outlen)
```
MAC Security -- an explanation

Recall: MAC security required an attacker given \((m, t)\) couldn’t find a different \(t’\) such that \((m, t')\) is a valid MAC.

Why? Suppose not: \((m, t) \rightarrow (m, t')\)

Then Encrypt-then-MAC would not have Ciphertext Integrity!!
## Performance

From Crypto++ 5.6.0 [Wei Dai]

<table>
<thead>
<tr>
<th>AE Cipher</th>
<th>Code Size</th>
<th>Speed (MB/sec)</th>
<th>Raw Cipher</th>
<th>Raw Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES/GCM</td>
<td>Large</td>
<td>108</td>
<td>AES/CTR</td>
<td>139</td>
</tr>
<tr>
<td>AES/CCM</td>
<td>smaller</td>
<td>61</td>
<td>AES/CBC</td>
<td>109</td>
</tr>
<tr>
<td>AES/EAX</td>
<td>smaller</td>
<td>61</td>
<td>AES/CMAC</td>
<td>109</td>
</tr>
<tr>
<td>AES/OCB*</td>
<td>small</td>
<td>129</td>
<td>HMAC/SHA1</td>
<td>147</td>
</tr>
</tbody>
</table>

* OCB mode may have patent issues. Speed extrapolated from Ted Kravitz’s results.
Summary

Encrypt-then-MAC
- Provides integrity of CT
- Plaintext integrity
- If cipher is malleable, we detect invalid CT
- MAC provides no information about PT since it’s over the encryption

MAC-then-Encrypt
- No integrity of CT
- Plaintext integrity
- If cipher is malleable, can change message w/o detection
- MAC provides no information on PT since encrypted

Encrypt-and-MAC
- No integrity on CT
- Integrity of PT can be verified
- If cipher is malleable, contents of CT can be altered; should detect at PT level
- May reveal info about PT in the MAC (e.g., MAC of same messages are the same)
Wrapup

• Authenticated Encryption
  – Chosen Ciphertext Attack (CCA) and CCA-secure ciphers
  – AE game = CCA + CPA secure

• Encrypt-then-MAC always right
  – Don’t roll your own
Questions?
END
Case Study: TLS
*Certificates* bind a public key to a user
**Certificate Authority** (CA) binds certificate to person

- **CA Signature**
- **Certificate parameters**
Alice Sends:
User ID || public key || ...
Alice Generates and Gives:
User ID || public key || ...

CA Computes:
\[ D = H(\text{User ID || public key || ...}) \]
\[ \text{Sig} = \text{Sign}(D, \text{CA private key}) \]
Gives Alice \text{Sig}
Alice Generates and Gives:
User ID || public key || ...

CA Computes:
\[ D = H(\text{User ID} \ || \ \text{public key} \ || \ ...) \]
\[ \text{Sig} = \text{Sign}(D, \text{Serial}, \text{CA private key}) \]
Gives Alice <Sig, Serial>

Alice’s Certificate
[User ID || public key || ...] || CA Name || Serial || Sig || <add. params>
X.509 Certificates
TLS and SSL

• Transport Layer Security (TLS)
  – Secure socket layer (SSL) predecessor
  – originally developed by Netscape
  – version 3 designed with public input
  – RFC 2246

• Uses TCP to provide a reliable end-to-end service
# Protocol Stack

<table>
<thead>
<tr>
<th>Application Layer</th>
<th>SSL</th>
<th>Transport Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Telnet</td>
<td>...</td>
</tr>
<tr>
<td>Handshake</td>
<td>Change Cipher</td>
<td>Alert</td>
</tr>
<tr>
<td>SSL Record Protocol</td>
<td>TCP</td>
<td>IP</td>
</tr>
<tr>
<td>IP</td>
<td>TCP</td>
<td>SSL</td>
</tr>
</tbody>
</table>

## Protocol Stack Description

- **Application Layer**
  - HTTP
  - Telnet
  - ...  

- **SSL Layer**
  - Handshake
  - Change Cipher
  - Alert
  - SSL Record Protocol

- **Transport Layer**
  - TCP
  - IP
Session Establishment

1. ClientHello
2. ServerHello
3. ClientKeyExchange

Encrypt with symmetric cipher using shared secret

Supported MAC’s and ciphers
Protocol Record

Application Data

Fragment

Compress

MAC

Encrypt

Prepend Hdr

HTTP | Telnet | ...
---|---|---
Handshake | Change Cipher | Alert
SSL Record Protocol
TCP
IP
Other Fields

Change cipher: Re-initiate handshake protocol, e.g., to re-negotiate the keying material used for encryption

Alert: Signal warning or fatal problem

- Fatal: unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
- Warning: close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown
Detailed Protocol

Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.
Unidirectional keys: $k_{b \rightarrow s}, k_{s \rightarrow b}$

Stateful encryption:

- Each side maintains two 64-bit counters: $\text{ctr}_{b \rightarrow s}, \text{ctr}_{s \rightarrow b}$
- Init. to 0 when session started. $\text{ctr}++$ for every record.
- Purpose: replay defense
TLS Record Encryption
(CBC AES-128, HMAC-SHA1)

Browser side $\text{enc}(k_{b\to s}, \text{data}, \text{ctr}_{b\to s})$:

- **step 1:** $\text{tag} \leftarrow S(k_{\text{mac}}, [++\text{ctr}_{b\to s} || \text{header} || \text{data}])$
- **step 2:** pad $[\text{header} || \text{data} || \text{tag}]$ to AES block size
- **step 3:** CBC encrypt with $k_{\text{enc}}$ and new random IV
- **step 4:** prepend header

$k_{b\to s} = (k_{\text{mac}}, k_{\text{enc}})$
TLS Record Decryption
(CBC AES-128, HMAC-SHA1)

Server side \( \text{dec}(k_{b \rightarrow s}, \text{record}, \text{ctr}_{b \rightarrow s}) : \)

step 1: CBC decrypt record using \( k_{\text{enc}} \)
step 2: check pad format, send \text{bad_record_mac} if invalid
step 3: check tag on \([++\text{ctr}_{b \rightarrow s} || \text{header} || \text{data}]\)
send \text{bad_record_mac} if invalid

Provides authenticated encryption
(provided no other info. is leaked during decryption)
TLS Record Decryption
(CBC AES-128, HMAC-SHA1)

Server side \(\text{dec}(k_{b\rightarrow s}, \text{record, ctr}_{b\rightarrow s})\) :

step 1: CBC decrypt record using \(k_{\text{enc}}\)

step 2: check pad format, send \text{decryption_failed} if invalid

step 3: check tag on \([++\text{ctr}_{b\rightarrow s} || \text{header} || \text{data}]\)
send \text{bad_record_mac} if invalid

V1.1 Bug:
Only difference is error messages
Server side $\text{dec}(k_{b\rightarrow s}, \text{record}, \text{ctr}_{b\rightarrow s})$:

step 1: CBC decrypt record using $k_{\text{enc}}$
step 2: check pad format, abort if invalid
step 3: check tag, abort if invalid

Two different types of errors: bad pad vs. bad MAC

**Padding Attack**: Attacker submits ciphertext and learns if last byte of plaintext are a valid pad
In older TLS 1.0:
padding oracle due to different alert messages.
Valid paddings:
– 0x01 for 1 byte padding
– 0x02 0x02 for 2 byte padding
– 0x03 0x03 0x03 for 3 byte padding
– ....
Using a Padding Oracle with CBC

Example:
Attacker has ciphertext $c = (c[0], c[1], c[2])$ and wants $m[1]$. We’ll show you how to get last byte of $m[1]$. (Full break possible)
Step 1: Throw Away $c[2]$
Step 2: Guess and Check if Padding Valid

Let $g$ be our guess for the last byte of $m[1]$

$D(k, \cdot) \oplus c[0] \oplus c[1] \oplus g \oplus 0x01$

- if last-byte = $g$: valid pad
- otherwise: invalid pad

*note MAC will fail, but we get the byte.
Using a Padding Oracle

Attack: submit \((IV, c'[0], c[1])\) to padding oracle
⇒ attacker learns if last byte = g

Repeat with \(g = 0, 1, ..., 255\) to learn last byte of \(m[1]\)

Then use a \((0x02, 0x02)\) pad to learn the next byte and so on ...
Another TLS Bug Prior to 1.1

IV for CBC is predictable using chained IV
  – IV for next record is last ciphertext block of current record.
  – Not CPA secure (see block cipher lecture).

BEAST attack is a practical implementation
Other Problems

The TLS header leaks the length of TLS records
• Lengths can also be inferred by observing network traffic

For many web applications, leaking lengths reveals sensitive info:
• In tax preparation sites, lengths indicate the type of return being filed which leaks information about the user’s income
• In healthcare sites, lengths leaks what page the user is viewing
• In Google maps, lengths leaks the location being requested

No easy solution
Lesson

1. Encrypt-then-MAC would completely avoid many problem.
   – MAC is checked first and ciphertext discarded if invalid

2. MAC-then-CBC provides Authenticated Encryption, but padding oracle destroys it
Certificate Revocation

What to do if your keys are compromised.
Certificate Revocation

1. ClientHello
2. ServerHello
   (send cert., e.g., pub key e)

1. Check CA signature on key
2. ....
3. Accept key

What needs to happen here?
Certificate Revocation

1. ClientHello

2. ServerHello
   (send cert., e.g., pub key e)

Verification protocol

Verification Authority
Certificate Verification Protocols

• Expiration Date

• Certificate Revocation Lists (CRL) and Certificate Revocation Trees (CRT)

• OCSP – Online Cert Status Protocol
Efficient Certificate Revocation Lists
(kocher98)

VA creates CRL and signs using private key. Note key very powerful.

Note no private keys on server
Certificate Revocation Tree Generation

VASig = Sign(H_{\text{root}}, \text{VA signing key})

Revoked cert C_j sorted by serial

Verification Authority
1. Is Bob’s Cert $C_2$ revoked

2. $[C_1, H_2, H_6, \text{VASig}]$

Alice validates $C_2$ by:

- $H'_\text{root} = H(H(C_1, C_2), H_2, H_6)$
- $H' =?= H$
- VA Sig valid?

Size of Proof: $O(\log i)$
Online Cert Status Protocol

1. Request(Bob’s Cert)

2. Check DB

3. Response(
   Sign(Bob’s Cert {OK,BAD})
   VA Signing Key
)

Verification Authority

Implemented in IE7 (Vista+), Firefox, Safari (by default Lion+), Opera, Chrome