Authenticated Encryption

Adam O’Neill
Based on http://cseweb.ucsd.edu/~mihir/cse207/
Motivation

In practice we often want both privacy and authenticity.

**Example:** A doctor wishes to send medical information $M$ about Alice to the medical database. Then

- We want data privacy to ensure Alice’s medical records remain confidential.
- We want authenticity to ensure the person sending the information is really the doctor and the information was not modified in transit.

We refer to this as authenticated encryption.
Syntax

Syntactically, an authenticated encryption scheme is just a symmetric encryption scheme $\mathcal{AE} = (K, \mathcal{E}, \mathcal{D})$ where
Security

- The same notion of privacy applies, namely IND-CPA
Security

- The same notion of privacy applies, namely IND-CPA

- For authenticity, the adversary’s goal is to get the receiver to accept a “non-authentic” ciphertext (i.e., not actually transmitted by the sender)
Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and $A$ an adversary.

<table>
<thead>
<tr>
<th>Game INTCTXT$_{\mathcal{AE}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>procedure Initialize</strong></td>
</tr>
<tr>
<td>$K \leftarrow \mathcal{K}$ ; $S \leftarrow \emptyset$</td>
</tr>
<tr>
<td><strong>procedure Enc($M$)</strong></td>
</tr>
<tr>
<td>$C \leftarrow \mathcal{E}_K(M)$</td>
</tr>
<tr>
<td>$S \leftarrow S \cup {C}$</td>
</tr>
<tr>
<td>Return $C$</td>
</tr>
</tbody>
</table>

**procedure Finalize($C$)**

$M \leftarrow \mathcal{D}_K(C)$

if $(C \notin S \land M \neq \bot)$ then
    return true
Else return false

The int-ctxt advantage of $A$ is

$$\text{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(A) = \Pr[\text{INTCTXT}_{\mathcal{AE}}^A \Rightarrow \text{true}]$$
Plain Encryption: CBC$

\begin{align*}
\textbf{Alg } \mathcal{E}_K(M) \\
C[0] &\leftarrow \{0, 1\}^n \\
\text{For } i = 1, \ldots, m \text{ do} \\
C[i] &\leftarrow E_K(C[i - 1] \oplus M[i]) \\
\text{Return } C
\end{align*}

\begin{align*}
\textbf{Alg } \mathcal{D}_K(C) \\
\text{For } i = 1, \ldots, m \text{ do} \\
M[i] &\leftarrow E_K^{-1}(C[i]) \oplus C[i - 1] \\
\text{Return } M
\end{align*}

**Question:** Is CBC$ encryption INT-CTXT secure?
Encryption with Redundancy

Let $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be our block cipher and $h: \{0, 1\}^* \rightarrow \{0, 1\}^n$ a redundancy function. Let $SE = (K, E', D')$ be CBC$ encryption and define the encryption with redundancy scheme $AE = (K, E, D)$ via

**Alg $E_K(M)$**

$M[1] \ldots M[m] \leftarrow M$

$M[m + 1] \leftarrow h(M)$

$C \leftarrow E_K'(M[1] \ldots M[m]M[m + 1])$

return $C$

**Alg $D_K(C)$**

$M[1] \ldots M[m]M[m + 1] \leftarrow D'_K(C)$

if $(M[m + 1] = h(M))$ then

return $M[1] \ldots M[m]$

else return $\perp$
Attacks

_adv_**adversary** _A_

\[
M[1] \overset{\$}{\leftarrow} \{0, 1\}^n; M[2] \leftarrow h(M[1])
\]


Return \(C[0]C[1]C[2]\)

This attack succeeds for any (not secret-key dependent) redundancy function \(h\).
Build an authenticated encryption scheme $AE = (K, E, D)$ by combining
- a given IND-CPA symmetric encryption scheme $SE = (K', E', D')$
- a given PRF $F : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^n$

<table>
<thead>
<tr>
<th></th>
<th>CBC$$-AES</th>
<th>CTR$$-AES</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMAC-SHA1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECBC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Build an authenticated encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ by combining

- a given IND-CPA symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}', \mathcal{E}', \mathcal{D}')$
- a given PRF $F : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^n$

A key $K = K_e \| K_m$ for $\mathcal{AE}$ always consists of a key $K_e$ for $\mathcal{SE}$ and a key $K_m$ for $F$:

\[
\text{Alg } K \\
K_e \leftarrow^\$ \mathcal{K}'; \ K_m \leftarrow^\$ \{0, 1\}^k \\
\text{Return } K_e \| K_m
\]
Generic Composition

The order in which the primitives are applied is important. Can consider

<table>
<thead>
<tr>
<th>Method</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt-and-MAC (E&amp;M)</td>
<td>SSH</td>
</tr>
<tr>
<td>MAC-then-encrypt (MtE)</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>Encrypt-then-MAC (EtM)</td>
<td>IPSec</td>
</tr>
</tbody>
</table>
Encrypt-and-MAC

$AE = (K, E, D)$ is defined by

| Alg $E_{K_e||K_m}(M)$ | Alg $D_{K_e||K_m}(C'||T)$ |
|------------------------|---------------------------|
| $C' \leftarrow E_{K_e}^*(M)$ | $M \leftarrow D_{K_e}^*(C')$ |
| $T \leftarrow F_{K_m}(M)$ | If $(T = F_{K_m}(M))$ then return $M$ |
| Return $C'||T$ | Else return $\perp$ |

<table>
<thead>
<tr>
<th>Security</th>
<th>Achieved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND-CPA</td>
<td>NO</td>
</tr>
<tr>
<td>INT-CTX</td>
<td>NO</td>
</tr>
</tbody>
</table>

Suppose you can modify or decipher to another encryption of the same message.
MAC-then-Encrypt

\( \mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D}) \) is defined by

\[
\begin{align*}
\text{Alg } & \ \mathcal{E}_{K_e\|K_m}(M) \\
T & \leftarrow F_{K_m}(M) \\
C & \leftarrow \mathcal{E}_{K_e}'(M\|T) \\
\text{Return } & \ C
\end{align*}
\]

\[
\begin{align*}
\text{Alg } & \ \mathcal{D}_{K_e\|K_m}(C) \\
M\|T & \leftarrow \mathcal{D}'_{K_e}(C) \\
\text{If } & (T = F_{K_m}(M)) \text{ then return } M \\
\text{Else return } & \bot
\end{align*}
\]

<table>
<thead>
<tr>
<th>Security</th>
<th>Achieved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND-CPA</td>
<td>Yes</td>
</tr>
<tr>
<td>INT-CTXT</td>
<td>No</td>
</tr>
</tbody>
</table>
POODLE Attack on SSL 3.0

Concretely, the cipher $E_{MtE} = (E_{MtE}, D_{MtE})$ obtained from applying MtE to randomized CBC mode encryption and a secure MAC works as follows:

- $E_{MtE}( (k_e, k_m), m)$: First use the MAC signing algorithm to compute a fixed-length tag $t \leftarrow S(k_m, m)$ for $m$. Next, encrypt $m \parallel t$ with randomized CBC encryption: pad the message and then encrypt in CBC mode using key $k_e$ and a random IV. Thus, the following data is encrypted to generate the ciphertext $c$:

  \[
  \begin{array}{c|c|c}
  \text{message } m & \text{tag } t & \text{pad } p \\
  \end{array}
  \]

  (9.8)

  Notice that the tag $t$ does not protect the integrity of the pad. We will exploit this to break CPA security using a chosen ciphertext attack.

- $D_{MtE}( (k_e, k_m), c)$: Run CBC decryption to obtain the plaintext data in (9.8). Next, remove the pad $p$ by reading the last byte in (9.8) and removing that many more bytes from the data (i.e., if the last byte is 3 then that byte is removed plus 3 additional bytes). Next, verify the MAC tag and if valid return the remaining bytes as the message. Otherwise, output reject.
POODLE Attack on SSL 3.0

Suppose the adversary intercepts a valid ciphertext \( c := E_{M\text{tE}}( (k_e, k_m), m) \) for some unknown message \( m \). The length of \( m \) is such that after a MAC tag \( t \) is appended to \( m \) the length of \( (m \parallel t) \) is a multiple of 16 bytes. This means that a full padding block of 16 bytes is appended during CBC encryption and the last byte of this pad is 15. Then the ciphertext \( c \) looks as follows:

\[
\begin{array}{c}
\text{IV} \quad \text{encryption of } m \quad \text{encrypted tag} \quad \text{encrypted pad} \\
\hline
\text{c}[0] \quad \text{c}[1] \quad \ldots \quad \text{c}[\ell - 1] \quad \text{c}[\ell] \\
\end{array}
\]

Let us first show that the adversary can learn something about \( m[0] \) (the first 16-byte block of \( m \)). This will break semantic security of \( E_{M\text{tE}} \). The attacker prepares a chosen ciphertext query \( \hat{c} \) by replacing the last block of \( c \) with \( c[1] \). That is,

\[
\hat{c} := \begin{array}{c}
\text{encrypted pad?} \\
\hline
\text{c}[0] \quad \text{c}[1] \quad \ldots \quad \text{c}[\ell - 1] \quad \text{c}[1] \\
\end{array}
\]

By definition of CBC decryption, decrypting the last block of \( \hat{c} \) yields the 16-byte plaintext block

\[
v := D(k_e, c[1]) \oplus c[\ell - 1] = m[0] \oplus c[0] \oplus c[\ell - 1].
\]
Breaking SSL 3.0

- Browser and server have a shared secret called a cookie
Breaking SSL 3.0

• Browser and server have a shared secret called a cookie

• Browser sends to serve GET: path, cookie
Breaking SSL 3.0

- Browser and server have a shared secret called a cookie.
- Browser sends to serve GET: path, cookie.
- Attacker sends Javascript program to browser making it issue request GET: /AA, cookie.
Breaking SSL 3.0

- Browser and server have a shared secret called a cookie

- Browser sends to serve GET: path, cookie

- Attacker sends Javascript program to browser making it issue request GET: /AA, cookie

- Using previous attack can learn one byte of the cookie
Breaking SSL 3.0

• Browser and server have a shared secret called a cookie

• Browser sends to serve GET: path, cookie

• Attacker sends Javascript program to browser making it issue request GET: /AA, cookie

• Using previous attack can learn one byte of the cookie

• Then shift the cookie right one byte by making Javascript program issue request GET: /AAA, cookie …
Encrypt-then-MAC

$AE = (K, E, D)$ is defined by

<table>
<thead>
<tr>
<th>$\text{Alg } E_{K_e|K_m}(M)$</th>
<th>$\text{Alg } D_{K_e|K_m}(C'|T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C' \leftarrow E'_{K_e}(M)$</td>
<td>$M \leftarrow D'_{K_e}(C')$</td>
</tr>
<tr>
<td>$T \leftarrow F_{K_m}(C')$</td>
<td>If $(T = F_{K_m}(C'))$ then return $M$</td>
</tr>
<tr>
<td>Return $C'|T$</td>
<td>Else return ⊥</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security</th>
<th>Achieved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND-CPA</td>
<td>yes</td>
</tr>
<tr>
<td>INT-CTXT</td>
<td>yes</td>
</tr>
</tbody>
</table>
Theorem

Encrypt-then-MAC is INT-CTXT-secure assuming PRF-security of $F$:

**Theorem:** Let $\mathcal{SE} = (\mathcal{K}', \mathcal{E}', \mathcal{D}')$ be a symmetric encryption scheme. Let $F : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a family of functions. Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be obtained by composing $\mathcal{SE}$ and $F$ in the Encrypt-then-MAC combination. Let $A$ be an int-ctxt adversary against $\mathcal{AE}$ make $q_e$ $\text{Enc}$ queries and having running time $t$. Then we can construct a prf-adversary $B$ against $F$ such that

$$\text{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(A) \leq \text{Adv}^\text{prf}_F (B) + \frac{1}{2^n}.$$

$B$ makes $q_e$ queries to its $\text{Fn}$ oracle and has running time $t$ plus some overhead.
Proof

Adversary B

\[ K_e \leftarrow R' \]

Run A

When A makes Enc query \( m \)

\[ c \leftarrow E'(K_e, m) \]

\( t' \leftarrow Fn(c) \)

return \( c_{llt} \)

Until A outputs \( c'_{llt'} \)

If \( c'_{llt'} \) is new and \( Fn(c') = t' \)
Theorem

Encrypt-then-MAC is IND-CPA-secure assuming IND-CPA-security of $SE'$:

**Theorem:** Let $SE' = (K', E', D')$ be a symmetric encryption scheme. Let $F : \{0, 1\}^k \times \{0, 1\}^* \to \{0, 1\}^n$ be a family of functions. Let $AE = (K, E, D)$ be obtained by composing $SE$ and $F$ in the Encrypt-then-MAC combination. Let $A$ be an ind-cpa adversary against $AE$ make $q$ LR queries and having running time $t$. Then we can construct an ind-cpa adversary $B$ against $SE'$ such that

$$\text{Adv}_{AE}^{\text{ind-cca}}(A) \leq \text{Adv}_{SE'}^{\text{ind-cca}}(B).$$

$B$ makes $q$ queries to its LR oracle and has running time $t$ plus some overhead.
Proof

Adversary $B_L^R(\cdot, \cdot)$
## Generic Composition in Practice

<table>
<thead>
<tr>
<th>AE in</th>
<th>is based on</th>
<th>which in general is</th>
<th>and in this case is</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td>E&amp;M</td>
<td>insecure</td>
<td>secure</td>
</tr>
<tr>
<td>SSL</td>
<td>MtE</td>
<td>insecure</td>
<td>insecure</td>
</tr>
<tr>
<td>SSL + RFC 4344</td>
<td>MtE</td>
<td>insecure</td>
<td>secure</td>
</tr>
<tr>
<td>IPSec</td>
<td>EtM</td>
<td>secure</td>
<td>secure</td>
</tr>
<tr>
<td>WinZip</td>
<td>EtM</td>
<td>secure</td>
<td>insecure</td>
</tr>
</tbody>
</table>

**Why?**

- Encodings
- Specific “E” and “M” schemes
- For WinZip, disparity between usage and security model
SSH2 encryption uses inter-packet chaining which is insecure [D, BKN]. RFC 4344 [BKN] proposed fixes that render SSH provably IND-CPA + INT-CTXT secure. Fixes recommended by Secure Shell Working Group and included in OpenSSH since 2003. Fixes included in PuTTY since 2008.
AE in SSL

- SSL uses MtE, which is not secure in general
AE in SSL

- SSL uses MtE, which is not secure in general
- We saw an attack on SSL 3.0, but this can be fixed if there is no padding
AE in SSL

- SSL uses MtE, which is not secure in general

- We saw an attack on SSL 3.0, but this can be fixed if there is no padding

- Both SSL 3.0 and TLS 1.0 actually use a defective variant of CBC$
AEAD

The goal has evolved into Authenticated Encryption with Associated Data (AEAD) [Ro].

- Associated Data (AD) is authenticated but not encrypted
- Schemes are nonce-based (and deterministic)

Sender

- $C \leftarrow \mathcal{E}_K(N, AD, M)$
- Send $(N, AD, C)$

Receiver

- Receive $(N, AD, C)$
- $M \leftarrow \mathcal{D}_K(N, AD, C)$

Sender must never re-use a nonce.

But when attacking integrity, the adversary may use any nonce it likes.
OCB


\( S = \text{PMAC}_K(AD) \) using separate tweaks.

Output may optionally be truncated.

Some complications (not shown) for non-full messages.

Optional in IEEE 802.11i
1-pass vs. 2-pass

• Many 1-pass schemes are patented (Jutla, Gilgor and Donescu, Rogaway)
1-pass vs. 2-pass

• Many 1-pass schemes are patented (Jutla, Gilgor and Donescu, Rogaway)

• 2-pass schemes offer tailored composition of specific base schemes with a single key
MtE-based but single key throughout. CTR-ENC is nonce-based counter mode encryption, and CBC-MAC is the basic CBC MAC. Ciphertext is $C \parallel T$. In NIST SP 800-38C, IEEE 802.11i.
Critiques

• Not “online”: message and AD lengths need to be known in advance
Critiques

• Not “online”: message and AD lengths need to be known in advance

• Can’t preprocess static AD
Critiques

• Not “online”: message and AD lengths need to be known in advance

• Can’t preprocess static AD

• Nonce length depends on message length and the former decreases as the latter increases
Critiques

- Not “online”: message and AD lengths need to be known in advance
- Can’t preprocess static AD
- Nonce length depends on message length and the former decreases as the latter increases
- Awkward/unnecessary parameters
Critiques

• Not “online”: message and AD lengths need to be known in advance

• Can’t preprocess static AD

• Nonce length depends on message length and the former decreases as the latter increases

• Awkward/unnecessary parameters

• Complex encodings
EtM-based but single key throughout. CTR-ENC is nonce-based counter mode encryption. Online; can pre-process static $AD$; always 128-bit nonce; simple; same performance as CCM. In ANSI C12.22.
CTR-ENC is nonce-based counter mode encryption. CWC-HASH is a AU polynomial-based hash. $K_H$ is derived from $K$ via $E$. Parallelizable; 300K gates for 10 Gbit/s (ASIC at 130 nanometers); Roughly same software speed as CCM, EAX, but can be improved via precomputation.
CTR-ENC is nonce-based counter mode encryption. GCM-HASH is a AL polynomial-based hash. $K_H$ is derived from $K$ via $E$. Can be used as a MAC. In NIST SP 800-38D.
Performance Comparisons x32

- CCM
- GCM
- OCB
- ECB

Gladman’s C code
Performance Comparisons x64

Gladman’s C code
• Which scheme to use? Depends on:
AE Today

• Which scheme to use? Depends on:
  • performance, patent requirements, single or multiple keys, standards compliance
AE Today

• Which scheme to use? Depends on:
  • performance, patent requirements, single or multiple keys, standards compliance

• Remains the most important practical goal
• Which scheme to use? Depends on:
  • performance, patent requirements, single or multiple keys, standards compliance

• Remains the most important practical goal

• Ongoing CAESAR competition, https://competitions.cr.yp.to/caesar.html
Schemes

**Generic composition:** E&M, MtE, EtM extend and again EtM is the best but others work too under appropriate conditions [NRS14].

1-pass schemes: IAPM [J], XCBC/XEBC [GD], OCB [RBBK, R]

2-pass schemes: CCM [FHW], EAX [BRW], CWC [KVW], GCM [MV]

Stream cipher based: Helix [FWSKLK], SOBER-128 [HR]

- 1-pass schemes are fast
- 2-pass schemes are patent-free
- Stream cipher based schemes are fast
TLS

- Designed by Netscape in 1994, called Secure Socket Layer (SSL) protocol
• Designed by Netscape in 1994, called Secure Socket Layer (SSL) protocol

• Primarily used to protect web traffic, but is general
TLS

- Designed by Netscape in 1994, called Secure Socket Layer (SSL) protocol
- Primarily used to protect web traffic, but is general
- SSL 3.0 in 1995 changed the protocol to use different keys for encryption and MAC
TLS

- Designed by Netscape in 1994, called Secure Socket Layer (SSL) protocol
- Primarily used to protect web traffic, but is general
- SSL 3.0 in 1995 changed the protocol to use different keys for encryption and MAC
- IETF designed TLS 1.0 in 1999, 1.3 in use today, by far the most widely used security protocol
• TLS setup uses public-key techniques and agrees on a shared key used (with additional $$) to derive $K_{bs}$ and $K_{sb}$
• TLS setup uses public-key techniques and agrees on a shared key used (with additional $$) to derive $K_{bs}$ and $K_{sb}$

• TLS 1.3 uses a nonce-based AEAD encryption scheme, negotiated during TLS setup.
• TLS setup uses public-key techniques and agrees on a **shared key** used (with additional $$) to derive $K_{bs}$ and $K_{sb}$

• TLS 1.3 uses a nonce-based AEAD encryption scheme, negotiated during TLS setup.

• The scheme takes the following arguments:
• TLS setup uses public-key techniques and agrees on a shared key used (with additional $$) to derive $K_{bs}$ and $K_{sb}$

• TLS 1.3 uses a nonce-based AEAD encryption scheme, negotiated during TLS setup.

• The scheme takes the following arguments:
  • Shared key $K_{bs}$ or $K_{sb}$
• TLS setup uses public-key techniques and agrees on a shared key used (with additional $$) to derive $K_{bs}$ and $K_{sb}$

• TLS 1.3 uses a nonce-based AEAD encryption scheme, negotiated during TLS setup.

• The scheme takes the following arguments:
  • Shared key $K_{bs}$ or $K_{sb}$
  • Plaintext data: up to $2^{14}$ bytes
• TLS setup uses public-key techniques and agrees on a shared key used (with additional $$) to derive $K_{bs}$ and $K_{sb}$

• TLS 1.3 uses a nonce-based AEAD encryption scheme, negotiated during TLS setup.

• The scheme takes the following arguments:

  • **Shared key** $K_{bs}$ or $K_{sb}$

  • **Plaintext data**: up to $2^{14}$ bytes

  • **Nonce**: 8 bytes of longer
The AEAD cipher outputs a ciphertext $c$ which is then formatted into an encrypted TLS record as follows:

<table>
<thead>
<tr>
<th>type</th>
<th>version</th>
<th>length</th>
<th>ciphertext $c$</th>
</tr>
</thead>
</table>


The AEAD cipher outputs a ciphertext $c$ which is then formatted into an encrypted TLS record as follows:

<table>
<thead>
<tr>
<th>type</th>
<th>version</th>
<th>length</th>
<th>ciphertext $c$</th>
</tr>
</thead>
</table>

- In TLS 1.3 length is sent in the clear!
The AEAD cipher outputs a ciphertext $c$ which is then formatted into an encrypted TLS record as follows:

<table>
<thead>
<tr>
<th>type</th>
<th>version</th>
<th>length</th>
<th>ciphertext $c$</th>
</tr>
</thead>
</table>

- In TLS 1.3 length is sent in the clear!
- Sequence number prevents replay attacks
SSH

• Popular command line tool for securely exchanging info with remote host
SSH

- Popular command line tool for securely exchanging info with remote host

- SSHv1 released in 1995, had many issues:
SSH

• Popular command line tool for securely exchanging info with remote host

• SSHv1 released in 1995, had many issues:
  • used a cyclic redundancy check
SSH

- Popular command line tool for securely exchanging info with remote host

- SSHv1 released in 1995, had many issues:
  - used a cyclic redundancy check
  - IV in CBC encryption set to zero
SSH

- Popular command line tool for securely exchanging info with remote host

- SSHv1 released in 1995, had many issues:
  - used a cyclic redundancy check
  - IV in CBC encryption set to zero
  - same key in both directions
SSH

- Popular command line tool for securely exchanging info with remote host

- SSHv1 released in 1995, had many issues:
  - used a cyclic redundancy check
  - IV in CBC encryption set to zero
  - same key in both directions

- SSHv2 released in 1996 tried to fix these issues
MAC. A MAC is computed over a sequence-number and the plaintext data in the thick box in Fig. 9.3. Here sequence-number is a 32-bit sequence number that is initialized to zero for the first packet, and is incremented by one after every packet. SSHv2 can use one of a number of MAC algorithms, but HMAC-SHA1-160 must be supported.

When an encrypted packet is received the decryption algorithm works as follows: first it decrypts the packet-length field using either $k_u$ or $k_s$. Next, it reads that many more packets from the network plus as many additional bytes as needed for the integrity tag. Next it decrypts the rest of the ciphertext and verifies validity of the integrity tag. If valid, it removes the pad and returns the plaintext message.

Although SSH uses encrypt-and-MAC, which is not generally secure, we show in Exercise 9.10 that for certain combinations of cipher and MAC, including the required ones in SSHv2, encrypt-and-MAC provides authenticated encryption.

SSH boundary hiding via length encryption. An interesting aspect of SSHv2 is that the encryption algorithm encrypts the packet length field, as shown in Fig. 9.3. The motivation for this is to ensure that if a sequence of encrypted SSH packets are sent over an insecure network as a stream of bytes, then an eavesdropper should be unable to determine the number of packets sent or their lengths. This is intended to frustrate certain traffic analysis attacks that deduce information about the plaintext from its size.

Hiding message boundaries between consecutive encrypted messages is outside the requirements addressed by authenticated encryption. In fact, many secure AEAD modes do not provide this level of secrecy. TLS 1.0, for example, sends the length of the every record in the clear making it easy to detect boundaries between consecutive encrypted records. Enhancing authenticated encryption

Figure 9.3: An SSHv2 packet
MAC. A MAC is computed over a sequence-number and the plaintext data in the thick box in Fig. 9.3. Here sequence-number is a 32-bit sequence number that is initialized to zero for the first packet, and is incremented by one after every packet. SSHv2 can use one of a number of MAC algorithms, but HMAC-SHA1-160 must be supported.

When an encrypted packet is received the decryption algorithm works as follows: first it decrypts the packet-length field using either $k_u$ or $k_s$. Next, it reads that many more packets from the network plus as many additional bytes as needed for the integrity tag. Next it decrypts the rest of the ciphertext and verifies validity of the integrity tag. If valid, it removes the pad and returns the plaintext message.

Although SSH uses encrypt-and-MAC, which is not generally secure, we show in Exercise 9.10 that for certain combinations of cipher and MAC, including the required ones in SSHv2, encrypt-and-MAC provides authenticated encryption.

SSH boundary hiding via length encryption. An interesting aspect of SSHv2 is that the encryption algorithm encrypts the packet length field, as shown in Fig. 9.3. The motivation for this is to ensure that if a sequence of encrypted SSH packets are sent over an insecure network as a stream of bytes, then an eavesdropper should be unable to determine the number of packets sent or their lengths. This is intended to frustrate certain traffic analysis attacks that deduce information about the plaintext from its size.

Hiding message boundaries between consecutive encrypted messages is outside the requirements addressed by authenticated encryption. In fact, many secure AEAD modes do not provide this level of secrecy. TLS 1.0, for example, sends the length of the every record in the clear making it easy to detect boundaries between consecutive encrypted records. Enhancing authenticated encryption

- Plaintext := packet-length || pad-length || message|| pad
Plaintext := packet-length || pad-length || message| pad

Encrypt using randomized CBC mode

Gray area is encrypted; Boxed area is authenticated by integrity tag

Figure 9.3: An SSHv2 packet

3. MAC. A MAC is computed over a sequence-number and the plaintext data in the thick box in Fig. 9.3. Here sequence-number is a 32-bit sequence number that is initialized to zero for the first packet, and is incremented by one after every packet. SSHv2 can use one of a number of MAC algorithms, but HMAC-SHA1-160 must be supported.

When an encrypted packet is received the decryption algorithm works as follows: first it decrypts the packet-length field using either $k_u$ or $k_s$. Next, it reads that many more packets from the network plus as many additional bytes as needed for the integrity tag. Next it decrypts the rest of the ciphertext and verifies validity of the integrity tag. If valid, it removes the pad and returns the plaintext message.

Although SSH uses encrypt-and-MAC, which is not generally secure, we show in Exercise 9.10 that for certain combinations of cipher and MAC, including the required ones in SSHv2, encrypt-and-MAC provides authenticated encryption.

SSH boundary hiding via length encryption. An interesting aspect of SSHv2 is that the encryption algorithm encrypts the packet length field, as shown in Fig. 9.3. The motivation for this is to ensure that if a sequence of encrypted SSH packets are sent over an insecure network as a stream of bytes, then an eavesdropper should be unable to determine the number of packets sent or their lengths. This is intended to frustrate certain traffic analysis attacks that deduce information about the plaintext from its size.

Hiding message boundaries between consecutive encrypted messages is outside the requirements addressed by authenticated encryption. In fact, many secure AEAD modes do not provide this level of secrecy. TLS 1.0, for example, sends the length of every record in the clear making it easy to detect boundaries between consecutive encrypted records. Enhancing authenticated encryption
Gray area is encrypted; Boxed area is authenticated by integrity tag

Figure 9.3: An SSHv2 packet

- Plaintext := `packet-length || pad-length || message| pad`
- Encrypt using randomized CBC mode
- Various MACs supported, must support HMAC-SHA1
• When an encrypted packet is received the decryption alg:
• When an encrypted packet is received the decryption alg:
  • Decrypts packet-length using the shared key
• When an encrypted packet is received the decryption alg:
  
  • Decrypts packet-length using the shared key
  
  • Reads many more packets plus as many additional bytes needed for the integrity tag
When an encrypted packet is received the decryption alg:

- Decrypts packet-length using the shared key
- Reads many more packets plus as many additional bytes needed for the integrity tag
- Decrypts remainder of ciphertext and verifies validity of the integrity tag
When an encrypted packet is received the decryption alg:

- Decrypts packet-length using the shared key
- Reads many more packets plus as many additional bytes needed for the integrity tag
- Decrypts remainder of ciphertext and verifies validity of the integrity tag
- If valid, remove tag and return message
• When an encrypted packet is received the decryption alg:
  • Decrypts packet-length using the shared key
  • Reads many more packets plus as many additional bytes needed for the integrity tag
  • Decrypts remainder of ciphertext and verifies validity of the integrity tag
  • If valid, remove tag and return message

• **Non-atomic decryption** leads to attack! [APW’09]
When an encrypted packet is received the decryption alg:

- Decrypts packet-length using the shared key
- Reads many more packets plus as many additional bytes needed for the integrity tag
- Decrypts remainder of ciphertext and verifies validity of the integrity tag
- If valid, remove tag and return message

Non-atomic decryption leads to attack! [APW’09]

Also attacks using timing and keystrokes [SWT’01]
802.11b WEP

• IEEE standard for wireless communication
802.11b WEP

- IEEE standard for wireless communication
- Designed to provide data privacy at the level of a wired network
802.11b WEP

• IEEE standard for wireless communication

• Designed to provide data privacy at the level of a wired network

• Uses either 128-bit or 40-bit keys (to support US export restrictions at the time)
802.11b WEP

Let $m$ be an 802.11b cleartext frame. The first few bits of $m$ encode the length of $m$. To encrypt an 802.11b frame $m$ the sender picks a 24-bit IV and computes:

$$c \leftarrow (m \parallel \text{CRC}(m)) \oplus \text{RC4}(\text{IV} \parallel k)$$
$$c_{\text{full}} \leftarrow (\text{IV}, c)$$

The WEP encryption process is shown in Fig. 9.4. The receiver decrypts by first computing $c \oplus \text{RC4}(\text{IV} \parallel k)$ to obtain a pair $(m, s)$. The receiver accepts the frame if $s = \text{CRC}(m)$ and rejects it otherwise.

![Figure 9.4: WEP Encryption](image-url)
802.11b WEP

• Attacks from:
802.11b WEP

- Attacks from:
  - IV collisions
802.11b WEP

• Attacks from:

  • IV collisions

  • Related keys - $1 \parallel k$, $2 \parallel k$,… used for stream cipher - gives attacks after about a million WEP frames sent
802.11b WEP

- Attacks from:
  - IV collisions
  - Related keys - $1 \parallel k$, $2 \parallel k$, ... used for stream cipher - gives attacks after about a million WEP frames sent
  - Malleability of CBC
802.11b WEP

- Attacks from:
  - IV collisions
  - Related keys - $1 \parallel k$, $2 \parallel k$, … used for stream cipher - gives attacks after about a million WEP frames sent
  - Malleability of CBC
  - Others…
IPSec

• Provides confidentiality and integrity for Internet IP packets
IPSec

- Provides confidentiality and integrity for Internet IP packets
- First released in 1998 and updated in 2005
IPSec

• Provides confidentiality and integrity for Internet IP packets

• First released in 1998 and updated in 2005

• Enables virtual private networks (VPNs) over public Internet channels
IPSec

- Provides confidentiality and integrity for Internet IP packets
- First released in 1998 and updated in 2005
- Enables virtual private networks (VPNs) over public Internet channels
  - Uses “west” and “east” gateways to encrypt and decrypt messages from different machines
Gray area is encrypted
Boxed area is authenticated by integrity tag

Figure 9.6: Cleartext IPv4 packet and an IPsec ESP packet
• Every IPsec encapsulated security payload (ESP) endpoint maintains a security association database (SAD)
• Every IPsec encapsulated security payload (ESP) endpoint maintains a security association database (SAD)

• SAD record (an SA) contains many connection-specific parameters and is identified by a 32-bit security parameters index (SPI)
• Every IPsec encapsulated security payload (ESP) endpoint maintains a security association database (SAD)

• SAD record (an SA) contains many connection-specific parameters and is identified by a 32-bit security parameters index (SPI)

• The recipient locates an appropriate SA in its SAD
• Every IPsec encapsulated security payload (ESP) endpoint maintains a security association database (SAD)

• SAD record (an SA) contains many connection-specific parameters and is identified by a 32-bit security parameters index (SPI)

• The recipient locates an appropriate SA in its SAD

• Sequence number enables detecting/discarding duplicates
• Every IPsec encapsulated security payload (ESP) endpoint maintains a security association database (SAD)

• SAD record (an SA) contains many connection-specific parameters and is identified by a 32-bit security parameters index (SPI)

• The recipient locates an appropriate SA in its SAD

• Sequence number enables detecting/discarding duplicates

• Traffic flow confidentiality (dummy packets) supported by appending dummy bytes to the payload
• Every IPsec encapsulated security payload (ESP) endpoint maintains a security association database (SAD)

• SAD record (an SA) contains many connection-specific parameters and is identified by a 32-bit security parameters index (SPI)

• The recipient locates an appropriate SA in its SAD

• Sequence number enables detecting/discard ing duplicates

• Traffic flow confidentiality (dummy packets) supported by appending dummy bytes to the payload

• One issue: Provides CPA-secure encryption without an integrity check