Symmetric and Password-based encryption

CS642: Computer Security

Professor Ristenpart

http://www.cs.wisc.edu/~rist/

rist at cs dot wisc dot edu
Symmetric encryption

Key generation

Kg

Optional

R

M

Enc

K

Dec

C

C is a ciphertext

Correctness: \( D( K, E(K, M, R) ) = M \) with probability 1 over randomness used
In TLS symmetric encryption underlies the Record Layer

What security properties do we need from symmetric encryption?

1) **Confidentiality**: should not learn any information about M
2) **Authenticity**: should not be able to forge messages

Often referred to as Authenticated Encryption security
Active security of CBC mode

What about forging a message? Pick any $C_0'$, $C_1'$ ...

Better yet for any $D$: $C_0' \oplus D$. 
Hash functions and message authentication

Hash function $H$ maps arbitrary bit string to fixed length string of size $m$

- **MD5**: $m = 128$ bits
- **SHA-1**: $m = 160$ bits
- **SHA-256**: $m = 256$ bits

Some security goals:
- collision resistance: can’t find $M \neq M'$ such that $H(M) = H(M')$
- preimage resistance: given $H(M)$, can’t find $M$
- second-preimage resistance: given $H(M)$, can’t find $M'$ s.t. $H(M') = H(M)$
Hash function application example

Password hashing. Choose random salt and store (salt,h) where:

\[
\text{salt} \ || \ \text{pw} \quad \rightarrow \quad H \quad \rightarrow \quad h
\]

The idea: Attacker, given (salt,h), should not be able to recover pw

Or can they?

For each guess pw':
- If \( H(\text{salt} \ || \ \text{pw'}) = h \) then Ret pw'

Rainbow tables speed this up in practice by way of precomputation. Large salts make rainbow tables impractical
Optional. If no randomness, then called a Message Authentication Code (MAC)

Correctness: \( \text{Ver}(K, \text{Tag}(K,M,R)) = 1 \) with probability 1 over randomness used

Unforgeability: Attacker can’t find \( M', T \) such that \( V(K, M', T) = 1 \)
Recall PRF security

F: \(\{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^n\)

Security goal: \(F(K,M)\) is indistinguishable from random n-bit string for anyone without \(K\)

For \(M_1, M_2, \ldots, M_q\) chosen by adversary and distinct

\[
F(K,M_1), F(K,M_2), \ldots, F(K,M_q)
\]

\(U_1, U_2, \ldots, U_q\)

Adversary that adaptively chooses messages but is limited to “reasonable” \(q\) (e.g., \(q = 2^{40}\)) can’t distinguish between two vectors

This means outputs of \(F\) are \textit{unpredictable}:

Given \(F(K,M_1), F(K,M_2), \ldots, F(K,M_{q-1})\) no attacker can predict \(F(K,M_q)\) with probability \(1 / 2^n + \text{negligible}\)
Any PRF is a good MAC

Correctness: \( \text{Ver}(K, \text{Tag}(K,M,R)) = 1 \) with probability 1 over randomness used

Unforgeability: Attacker can’t find \( M',T \) such that \( V(K,M',T) = 1 \)
Any PRF is a good MAC

key generation picks uniform key for $F$

$R_k \rightarrow Kg$

$K \rightarrow F(K,M) \rightarrow T$

$F(K,M) = T? \rightarrow 0 \text{ or } 1$

How do we instantiate $F$?
Message authentication with HMAC

Use a hash function $H$ to build a MAC. Kg outputs uniform bit string $K$

$Tag(K,M) = HMAC(K,M)$ defined by:

$K \oplus \text{ipad} \ || \ M$ 

$K \oplus \text{opad} \ || \ h$

To verify a $M,T$ pair, check if $HMAC(K,M) = T$

Unforgeability holds if $H$ is a secure PRF when so-keyed
Build a new scheme from CBC and HMAC
Kg outputs CBC key K1 and HMAC key K2

Several ways to combine:
(1) encrypt-then-mac
(2) mac-then-encrypt
(3) encrypt-and-mac
Build a new scheme from CBC and HMAC
Kg outputs CBC key K1 and HMAC key K2

Several ways to combine:
(1) encrypt-then-mac
(2) mac-then-encrypt
(3) encrypt-and-mac

Thm. If encryption scheme provides confidentiality against passive attackers and MAC provides unforgeability, then Encrypt-then-MAC provides secure authenticated encryption
TLS record protocol: MAC-Encode-Encrypt (MEE)

Padding is not MAC’d. Implementations must handle padding checks very carefully.

MAC
- HMAC-MD5, HMAC-SHA1, HMAC-SHA256

Encrypt
- CBC-AES128, CBC-AES256, CBC-3DES, RC4-128
## Dedicated authenticated encryption schemes

<table>
<thead>
<tr>
<th>Attack</th>
<th>Inventors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCB (Offset Codebook)</td>
<td>Rogaway</td>
<td>One-pass</td>
</tr>
<tr>
<td>GCM (Galios Counter Mode)</td>
<td>McGrew, Viega</td>
<td>CTR mode plus specialized MAC</td>
</tr>
<tr>
<td>CWC</td>
<td>Kohno, Viega, Whiting</td>
<td>CTR mode plus Carter-Wegman MAC</td>
</tr>
<tr>
<td>CCM</td>
<td>Housley, Ferguson, Whiting</td>
<td>CTR mode plus CBC-MAC</td>
</tr>
<tr>
<td>EAX</td>
<td>Wagner, Bellare, Rogaway</td>
<td>CTR mode plus OMAC</td>
</tr>
</tbody>
</table>
Symmetric Encryption Advice

*Never* use CTR mode or CBC mode by themselves.

Passive security is almost never good enough!!

Encrypt-then-MAC better than MAC-then-Encrypt, Encrypt and MAC

Dedicated modes that have been analyzed thoroughly are also good.
Password-based symmetric encryption

\[ D(\text{pw}, E(\text{pw}, M, R)) = M \] with probability 1 over randomness used
Encrypt-then-MAC with CBC and HMAC

Ciphertext is $C, T$

How do we use with a pw?
Password-based Key Derivation (PBKDF)

PBKDF(pw,salt):

\[ \text{Truncate if needed} \]

\[ \text{repeat } c \text{ times} \]

\[ \text{K1} \]

\[ \text{K2} \]
PBKDF + Symmetric encryption yields PW-based encryption

**Enc(pw,M,R):**
- salt || R' = R
- K = PBKDF(pw,salt)
- C = Enc'(K,M,R')
- Return (salt,C)

**Dec(pw,C):**
- salt || C' = C
- K = PBKDF(pw,salt)
- M = Enc'(K,C')
- Return M

Here `Enc'` is a normal symmetric encryption scheme (CBC+HMAC)

Attacks?
### Recommendation:
It should not be a name, a slang word, or any word in the dictionary. It should not include any part of your name or your e-mail address.

Almost all of the 5000 most popular passwords, that are used by a share of 20% of the users, were just that – names, slang words, dictionary words or trivial passwords (consecutive digits, adjacent keyboard keys, and so on). The most common password among Rockyou.com account owners is "123456". The runner up is "12345". The following table depicts the top 20 common passwords in the database list:

<table>
<thead>
<tr>
<th>Rank</th>
<th>Password</th>
<th>Number of Users with Password (absolute)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123456</td>
<td>290731</td>
</tr>
<tr>
<td>2</td>
<td>12345</td>
<td>79078</td>
</tr>
<tr>
<td>3</td>
<td>123456789</td>
<td>76790</td>
</tr>
<tr>
<td>4</td>
<td>Password</td>
<td>61958</td>
</tr>
<tr>
<td>5</td>
<td>iloveyou</td>
<td>51622</td>
</tr>
<tr>
<td>6</td>
<td>princess</td>
<td>35231</td>
</tr>
<tr>
<td>7</td>
<td>rockyou</td>
<td>22588</td>
</tr>
<tr>
<td>8</td>
<td>1234567</td>
<td>21726</td>
</tr>
<tr>
<td>9</td>
<td>12345678</td>
<td>20553</td>
</tr>
<tr>
<td>10</td>
<td>abc123</td>
<td>17542</td>
</tr>
</tbody>
</table>

11. Nicole 17168
12. Daniel 16409
13. babygirl 16094
14. monkey 15294
15. Jessica 15162
16. Lovely 14950
17. michael 14898
18. Ashley 14329
19. 654321 13984
20. Qwerty 13856

From an Imperva study of released RockMe.com password database 2010.
Brute-force attacks

- Given known plaintext, ciphertext pair:
  - $M$ and $C = \text{Enc}(pw,M)$
- Enumerate a dictionary $D$ of possible passwords, in order of likelihood

**BruteForce1(M,C):**

1. $R \ || \ C' = C$
2. foreach $pw*$ in $D$
   - $C^* = \text{Enc}(pw^*,M,R)$
   - If $C^* = C'$ then
     - Return $pw*$

$R$ is salt $\ || $ IV in CBC-based modes
Both are public:
$C = \text{salt} \ || \ IV \ || \ C1 \ || \ ...$

Diagram:
- $E_{K1}$
- $M1$
- $IV$
- $C0$
- $C1$
Brute-force attacks

• Given known plaintext, ciphertext pair:
  – M and C = Enc(pw,M)

• Enumerate a dictionary D of possible passwords, in order of likelihood

BruteForce1(M,C):
R || C’ = C
foreach pw* in D do
  C* = Enc(pw*,M,R)
  If C* = C’ then
    Return pw*

BruteForce2(C):
foreach pw* in D do
  M* = Dec(pw*,C)
  If M* “looks right” then
    Return (pw*,M*)
PBKDF design attempts to slow down brute-force attacks

Iterating $c$ times should slow down attacks by factor of $c$

Salts:
- Different derived keys, even if same password
- Slows down attacks against multiple users
- Prevents precomputation attacks, if salts chosen correctly

Diagram:
```
pw || salt || 1       H       H       ...       H   K1
```
Truncate if needed
Say \( c = 4096 \). Generous back of envelope* suggests that in 1 second, can test 252 passwords and so a naïve brute-force:

<table>
<thead>
<tr>
<th>Password Complexity</th>
<th>Average Combinations</th>
<th>Time Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 numerical digits</td>
<td>( 10^6 = 1,000,000 )</td>
<td>(~ 3968) seconds</td>
</tr>
<tr>
<td>6 lower case alphanumeric digits</td>
<td>( 36^6 = 2,176,782,336 )</td>
<td>(~ 99) days</td>
</tr>
<tr>
<td>8 alphanumeric + 10 special symbols</td>
<td>( 72^8 = 722,204,136,308,736 )</td>
<td>(~ 33) million days</td>
</tr>
</tbody>
</table>

* I did the arithmetic...
802.11 WPA passwords

PMK = PBKDF( pw, ssid || ssidlength )
with c = 4096

PTK = H( PMK || ANonce || SNonce || AP
       MAC address || STA MAC address )

MIC = HMAC-MD5(PTK, 2^{nd} message)

So after sniffing one handshake by another party, we can mount offline brute force attack
802.11 WPA passwords

PMK = PBKDF( pw, ssid | ssidlength )
with c = 4096

PTK = H( PMK || ANonce || SNonce || AP MAC address || STA MAC address )

MIC = HMAC-MD5(PTK, 2nd message)

BruteForce(MIC, ANonce, SNonce, 2nd message):
foreach pw* in D do
  PMK* = PBKDF(pw*, ssid | ssidlength)
  PTK* = H(PMK* || ANonce || ... )
  MIC* = HMAC-MD5(PTK*, 2nd message)
  If MIC* = MIC then
    Return pw*
We can also use precomputation for common SSID’s

\[
\begin{align*}
\text{PMK} &= F(pw, ssid) \\
\text{MIC} &= G(\text{PMK}, \text{data})
\end{align*}
\]

\[
\begin{align*}
\text{PMK} &= \text{PBKDF}(pw, ssid|\text{ssidlength}) \quad \text{with } c = 4096 \\
\text{PTK} &= H(\text{PMK} || \text{ANonce} || \text{SNonce} || \text{AP MAC address} || \text{STA MAC address}) \\
\text{MIC} &= \text{HMAC-MD5}(\text{PTK}, \text{2nd message})
\end{align*}
\]

**Offline(D, SsidList):**

- foreach \(pw^*\) in \(D\) do
  - foreach \(ssid^*\) in \(SsidList\) do
    - \(\text{PMK}^* = F(pw^*, ssid^*)\)
    - \(T[\text{PMK}^*] = pw^*\)
    - Add \(\text{PMK}^*\) to \(P[\text{ssid}^*]\)

**Return P, T**

**Online(P, T, MIC, ANonce, ...):**

- foreach \(\text{PMK}^*\) in \(P[\text{ssid}]\) do
  - \(\text{MIC}^* = G(\text{PMK}^*, \text{data})\)
  - If \(\text{MIC}^* = \text{MIC}\) then
    - Return \(T[\text{PMK}^*]\)

**Time-space trade-off**
Password recap

• Short passwords can be cracked easily
  – See also: JohnTheRipper, aircrack, tools

• Salting and iteration are helpful and needed
  – Salts must be sufficiently large and unpredictable
  – Still possible to crack in some cases

From xkcd.com