Message authentication

-- Hash based MAC functions
-- MAC functions based on block ciphers
-- Authenticated encryption

(c) Levente Buttyán (buttyan@crysys.hu)

Secret prefix method

\[ \text{MAC}_k(x) = H(k|x) \]

- insecure!
  - assume an attacker knows the MAC on \( x \): \( M = H(k|x) \)
  - he can produce the MAC on \( x'|y \) as \( M' = f(M,y) \), where \( x' \) is \( x \) with padding and \( f \) is the compression function of \( H \)

Message authentication
**A similar mistake**

$$\text{MAC}_k(x) = H_k(x)$$
where $H_k(.)$ is $H(.)$ with $CV_0 = k$

**Secret suffix method**

$$\text{MAC}_k(x) = H(x|k)$$

- insecure if $H$ is not collision resistant
  - using a birthday attack, the attacker finds two inputs $x$ and $x'$ such that $H(x) = H(x')$ (can be done off-line without the knowledge of $k$)
  - then obtaining the MAC $M$ on one of the inputs, say $x$, allows the attacker to forge a text-MAC pair $(x', M)$
- weaknesses
  - MAC depends only on the last chaining variable
  - key is involved only in the last step
**Envelop method**

$$\text{MAC}_K(x) = H(k|x|k)$$

- a key recovery attack has been discovered on this scheme (requiring $2^{64}$ text-MAC pairs for MD5 with 128-bit key)
- although, not really practical, the attack still represents an architectural flaw

---

**HMAC**

$$\text{HMAC}_k(x) = H( (k^* \oplus \text{opad}) | H( (k^* \oplus \text{ipad}) | x ) )$$

where
- $h$ is a hash function with input block size $b$ and output size $n$
- $k^*$ is $k$ padded with 0s to obtain a length of $b$ bits
- $\text{ipad}$ is 00110110 repeated $b/8$ times
- $\text{opad}$ is 01011100 repeated $b/8$ times

---

Message authentication 5

Message authentication 6
**Encrypted hash**

\[ \text{MAC}_K(x) = E_K(H(x)) \]

- Off-line search for messages with colliding MAC values is possible here without the knowledge of \( k \rightarrow H \) must be collision resistant!
- Collision resistant hash functions usually have larger output size than the block size of the block cipher \( \rightarrow \) which mode to use to encrypt the hash?
- Two messages having the same hash value will have the same MAC value under all keys

**CBC-MAC**

- CBC MAC is secure for messages of a fixed number of blocks
- Forgery is possible if variable length messages are allowed
A known-text forgery

- given two text-MAC pairs \((x, M)\) and \((x', M')\), a third valid text-MAC pair can be computed as follows:
  \[
  (x | 100... | x'_1 \oplus M | x'_2 | ... | x'_L, M')
  \]

A chosen-text forgery

- given a known text-MAC pair \((x_1, M_1)\)
- request MAC for \(M_1\), receive \(M_2 = E_k(M_1 \oplus 0) = E_k(M_1)\)
- \(M_2\) is the MAC of the message \((x_1|0)\)
Another chosen-text forgery

- given two known text-MAC pairs: \((x_1, M_1), (x_2, M_2)\)
- request MAC for message \(x_1|M_1\oplus M_2\oplus z\), where \(z\) is an arbitrary block
- receive \(M_3 = E_k(M_1\oplus M_2\oplus z\oplus M_1) = E_k(M_2\oplus z)\)
- \(M_3\) is also the MAC for message \(x_2|z\)

\[
\begin{align*}
\text{last block of } x_1 & \quad \rightarrow \quad M_4 \oplus M_2 \oplus z \\
k & \quad \rightarrow \quad E \\
M_1 & \quad \rightarrow \quad E \\
M_3 = E_k(M_2 \oplus z) & \\
\text{last block of } x_2 & \quad \rightarrow \quad z \\
k & \quad \rightarrow \quad E \\
M_2 & \quad \rightarrow \quad E \\
E_k(z \oplus M_2) = M_3
\end{align*}
\]

How to use CBC-MAC in practice?

- use the optional final encryption
  - reduces the threat of exhaustive key search (key is \((k, k')\) \(\rightarrow\) key length is doubled)
  - prevents the previously presented existential forgeries
  - has marginal overhead (only last block is encrypted multiple times)

- prepend the message with a block containing the length of the message before the MAC computation

- use \(k\) to encrypt the length and obtain \(k' = E_k(\text{length})\), and use \(k'\) as the MAC key (i.e., use message dependent MAC keys)
CMAC

- proposed to fix problems with CBC-MAC

![CMAC Diagram]

Authenticated encryption schemes

- simultaneously protect the confidentiality and the integrity of a message

  motivations:
  - to prevent chosen-ciphertext attacks (such as the Vaudenay attack)
    - the decryption oracle immediately recognizes improperly constructed ciphertexts and refuses to decrypt them
    - attacker can construct a correct ciphertext only if he already knows the plaintext → decryption oracle becomes useless
  - efficiency (in some cases)
    - needs fewer operations if the message is encrypted and the authentication tag is computed in a single pass

  approaches:
  - specialized schemes (e.g., XCBC, OCB, CCM)
  - combine “regular” encryption and MAC functions (but be careful!)
    - $E_k(x, MAC_{k_2}(x))$ (check for padding oracle attack !)
    - $E_k(x), MAC_{k_2}(x)$ (check for padding oracle attack !)
    - $E_k(x), MAC_{k_2}(E_k(x))$ (considered to be the most secure approach)
CCM mode

- CCM means CTR mode and CBC-MAC
  - (two pass) authenticated encryption mode
  - integrity protection is based on CBC-MAC
  - encryption is based on CTR mode
  - the same block cipher and key is used for both operations

- inputs:
  - K – key
  - N – nonce (should not repeat for a given key K)
  - m – message to be protected
  - a – additional data to be authenticated only (e.g., message header)

- outputs:
  - encrypted message
  - encrypted authentication tag (MAC value)

CCM – computing the authentication tag

- first block $B_0$:

- next blocks containing $a$:

- next blocks containing $m$:
CCM – computing the authentication tag

- given $B_0, B_1, ..., B_n$:
  
  $X_1 \leftarrow E_k(B_0)$
  
  $X_{i+1} \leftarrow E_k(X_i + B_i)$ for $i = 1, 2, ..., n$
  
  $T \leftarrow \text{first-M-bytes}(X_{n+1})$

  output $T$ as the MAC value

---

CCM – encryption

- the key stream blocks are computed as
  
  $S_i \leftarrow E_k(C_i)$ for $i = 0, 1, 2, ...$

  where $C_i$ is formatted as:

  - the first length($m$) octets of $S_1, S_2, ...$ are XORed to $m$ to produce the ciphertext
  - $S_0$ is used to encrypt the authentication tag:

  $U \leftarrow T + \text{first-M-bytes}(S_0)$
CCM – notes

- **security**
  - Level of confidentiality and authenticity is in-line with other proposed authenticated encryption modes, e.g., OCB
  - Encryption of the authentication tag T for avoiding MAC collision attacks (attacker gets no information about the CBC-MAC results)
  - Same key for MAC and encryption? No problem…
    - Evenly never gets the same input (C_i's are very likely different from B_i's)
    - An intermediate value in the CBC-MAC computation may collide with a C_i, but those values cannot be observed, and they affect only T which is encrypted

- **efficiency**
  - Two pass processing, but ...
  - Blocks used by the authentication function match up the blocks used by the encryption function

- **nonce selection**
  - Nonce values should be unique within the scope of a key
  - Nonce can be a sequence number
  - Otherwise, a pre-computation attack would be possible
    - Assume that the key is 128 bits long
    - Choose a particular nonce N_0
    - Choose 2^{14} key, and for each K store (K, S_1)
    - When a genuine message with N_0 is sent, guess the first 16 octets of the plaintext (usually higher layer header fields) and compute S_1' = H((message changes + N_0) K)
    - Look-up S_1' in the table (you will find it with large probability due to the birthday paradox), the corresponding K' value is the key

Summary

- Naïve hash based MAC constructions are usually not secure
- Better to use standard, well-studied constructions, e.g., HMAC
- CBC-MAC is interesting, because it does not need a hash function, but it can use the same block cipher that is used for encryption, anyway
- Existential forgeries against CBC-MAC exist, but there are countermeasures
  - E.g., prepending additional context data such as message length to the message, multiple encryption of the last block, etc.
- Authenticated encryption modes have some advantages
  - Efficiency: the two goals may be achieved in a single pass
  - Security: no information is leaked through a padding oracle