CAN304 W3

Message authentication code

Message integrity

我们一直关注确保通信的保密性。

Integrity：确保接收到的消息来自预期方，并且未被修改，即使攻击者控制该通道。

保密 (secrecy) 和完整性 (integrity) 是正交的问题，可以有一个而没有另一个。

加密可以帮助发现信息是否被更改：如果解密的消息毫无意义，不可读，则被更改了；但加密的目的不是。

Message authentication code (MAC)

消息身份验证代码由三种算法（Gen、Mac、Vrfy）定义：

- Gen：生成一个随机密钥 k
- Mac：将密钥 k 和 message $m \in \{0,1\}^*$ 作为输入，输出 tag t

$$t := Mac_k(m)$$

- Vrfy：将密钥 k、message m 和 tag t 作为输入；输出 1（“accept”）或 0（“reject”）(如果是 Gan 生成的密钥，则接受；如果不是，则拒绝)

$$For\ all\ m\ and\ all\ k\ output\ by\ Genknow$$
$$Vrfy_k(m, Mac_k(m)) = 1$$

最后验证主要靠 message m 和 tag t。

Fixed-length MAC

Construction

- Let F be a block cipher

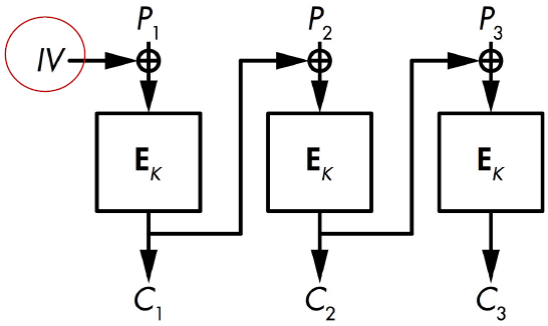
- Construct the following MAC Π:
  - Gen: choose a uniform key k for F
  - $\mathrm{Mac_k}(m)$: output $\mathrm{t} \leftarrow \mathrm{F_k}(m)$
  - $\mathrm{Vrfy_k}(\mathrm{m}, \mathrm{t})$: output 1 iff $\mathrm{F_k}(m) = t$

在实践中，block ciphers 具有短的，固定长度块大小。因此，之前的构造仅限于对短的、固定长度的消息进行身份验证。

CBC-MAC

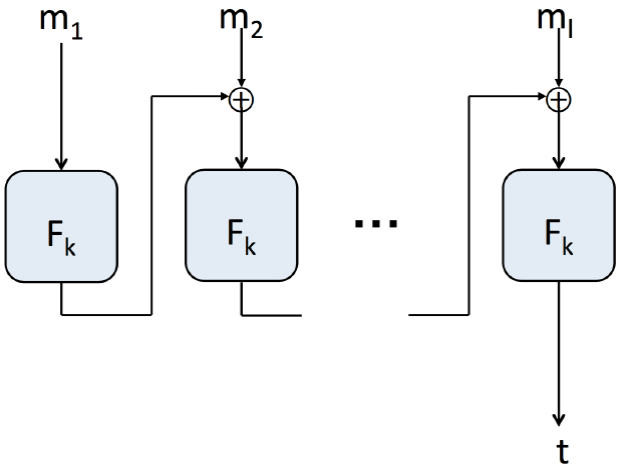
Recall CBC mode

CBC 使用先前块的密文来加密当前块：将先前块的密文和当前块的明文进行异或操作，再加密其结果。这样每个块的加密取决于所有先前块的内容。



CBC-MAC

CBC-MAC 使用 CBC mode，$m = m_1 m_2 \dots m_l$。



CBC-MAC 主要生成 tag t，用来验证。

CBC-MAC vs. CBC-mode encryption

- CBC-MAC 是确定性的 (deterministic，没有 IV，initialization vectors)，对于相同的输入总是有相同的结果
- 在 CBC-MAC 中，仅输出最终值 t，通过重新计算结果进行验证

- 可以从其 MAC 的 tag 中恢复原始 message 吗?
  - 不行!

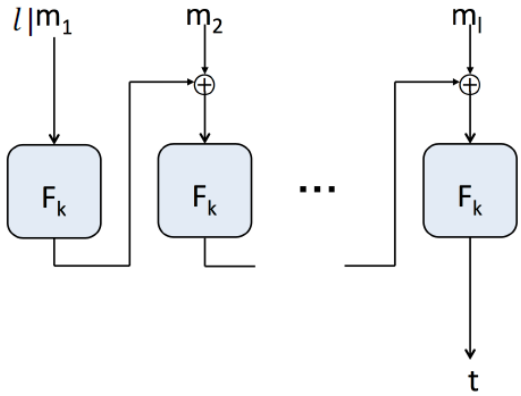
CBC-MAC extensions

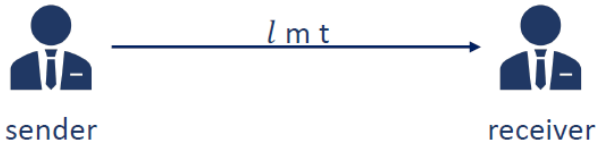
处理可变长度消息的几种方法

- 其长度不是块长度的倍数

最简单的方法之一：在应用 CBC-MAC 之前预置消息长度

- $m=m_1m_2...m_l$
- Suppose length(m_1) < block size

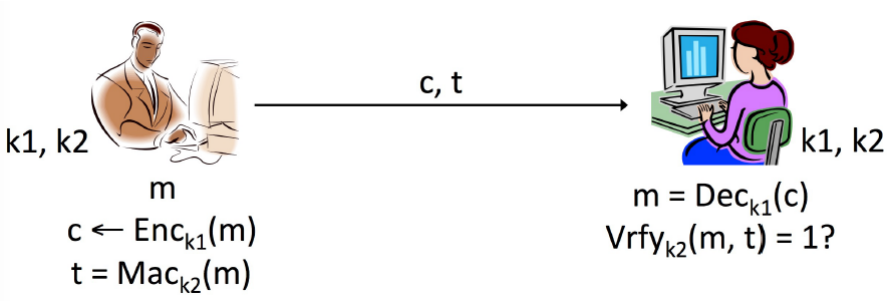




假如块的长度为128 bit，我们把消息分成长度为 128 bit 的块，然后到 m_1 的时候，发现 m_1 的长度不足 128 bit。我们使用 l 把 m_1 的长度补成 128 bit，再进行 CBC-MAC。之后发送方要把 l 也告诉接收方，接收方以相同的操作来进行验证。

Secure communications

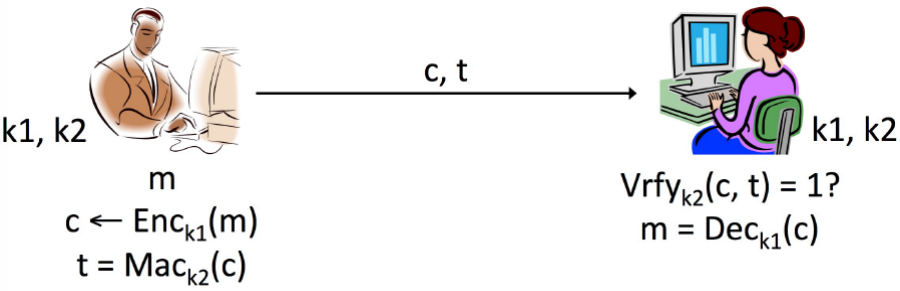
在消息传递中，我们希望同时达到 Secrecy 和 integrity。我们用加密来达到 Secrecy，用 MAC 来达到 integrity。



上图中，发送者用 k1 加密消息，用 k2 生成 MAC tag，然后把加密消息和 tag 发给接收方；接收方用 k1 解密消息，用 k2 来验证消息是否被修改过。这样就达成了加密通讯。

不过这里存在一个问题：假如我们发送两次相同的信息，密文会不同，但 tag 会相同 (MAC is deterministic)，这会泄露信息。

不过这个问题很好解决：我们不生成明文的 MAC，而是生成密文的 MAC，这样每次密文不同，tag 也不同。而接收方收到消息后，需要先用密文验证，再解密。



Attacks in Secure sessions

Replay attack：sender 给 receiver 发送两条消息 (c1, t1) 和 (c2, c2)；attacker 截留第二条消息 (c2, c2)，并伪装成 sender 再发一遍 (c1, t1)。

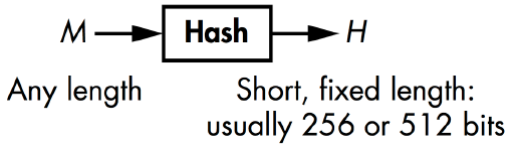
Re-ordering attack：sender 给 receiver 发送两条消息 (c1, t1) 和 (c2, c2)；attacker 截留两条消息，伪装成 sender 更改顺序后发送两条消息。

Solution：使用 counters 和 identities 来防止这些攻击 (以及其他攻击)。即发送消息时带上自己的标识和消息的数量，如 ("Bob"|m_1|1) 或 ("Alice"|m_5|5)。

Hash functions and applications

Hash functions

(Cryptographic) hash function：(加密哈希函数) 将任意长度的输入映射到固定长度的，简短的摘要 (digest)。



可以定义 keyed or unkeyed hash functions (即，使用密钥生成，和不使用密钥生成)。

Properties of cryptographic hash functions

Let H: {0,1}* → {0,1}^{n} be a hash function (任意长度映射到固定长度)。Cryptographic hash functions 有三个性质：

- Preimage resistance

  我们可以把任意的信息 M 映射为 hash value h，M 便是 h 的preimage。我们只能通过 M 计算出 h，无法通过 h 计算出 M (One-way function)

- Second preimage resistance

  给定一个 x，我们无法找到一个 x' ($x' \neq x$) 来使得 $H(x) = H(x')$，这个也叫 weak collision resistant。

- Collision resistance

  collision 是指给定两个不同的输入 x 和 x'，它们的输出 $H(x) = H(x')$。如果我们无法找到 H 的 collision，那么 H 是 collision-resistance 的，这也叫 strong collision resistance。

weak collision resistant 和 strong collision resistance 的区别：weak collision resistant 是给定一个 x，找到 x'；strong collision resistance 是要找到一对 x 和 x'。

Generic hash-function attacks

再 hash function 中，输入是任意长度的，但输出是固定长度的 - {0, 1}^{n}，即 2^n。因此，如果我们对 2^n+1 个信息进行 hash function，则 100% 保证至少有一个 collision。

不过，如果我们希望有 50% 的概率找到一个 collision，需要有多少个信息呢？— $2^{n/2}$ 个。

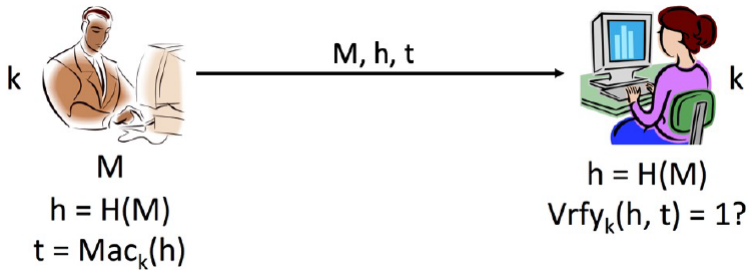
Hash functions in practice

- MD5
  - 128-bit output length
  - should no longer be used
- SHA-1
  - 160-bit output length
  - 理论分析表明它存在一些弱点
  - 很常见，但有被 SHA-2 代替的趋势
- SHA-2
  - 224-bit, 256-bit, 384-bit or 512-bit output lengths
  - 没有已知的重大弱点
- SHA-3/Keccak
  - 与 SHA family 截然不同的设计
  - Supports 224, 256, 384, and 512-bit outputs

HMAC

现在我们使用 hash function 来进行验证：sender 计算 message m 的 hash value h，然后把 m 和 h 发给 receiver；receiver 再计算 m 的 hash value，并与 h 作比较，如果一致，则消息没有被篡改。

我们可以把 Hash function 和 block cipher-based MAC 这两个 crypto primitives 结合起来：



HMAC：

Construction

$$\text{HMAC: } S(k,m) = H(k \oplus opad || H(k \oplus ipad || m))$$

- ipad: 00110110
- opad: 01011100

k 是密钥，m 是 message，H 是 hash function，II 代表 cancat。

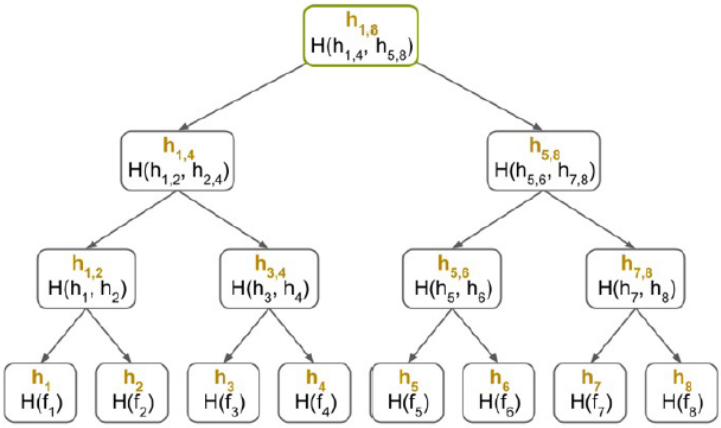
Merkle tree

Hash function 的另一项应用是 Merkle tree。

$h = MHT(x_1, x_2, \ldots, x_n)$

Construction

Suppose we have 8 files $(f_1, \ldots, f_8)$, H is collision resistant hash function



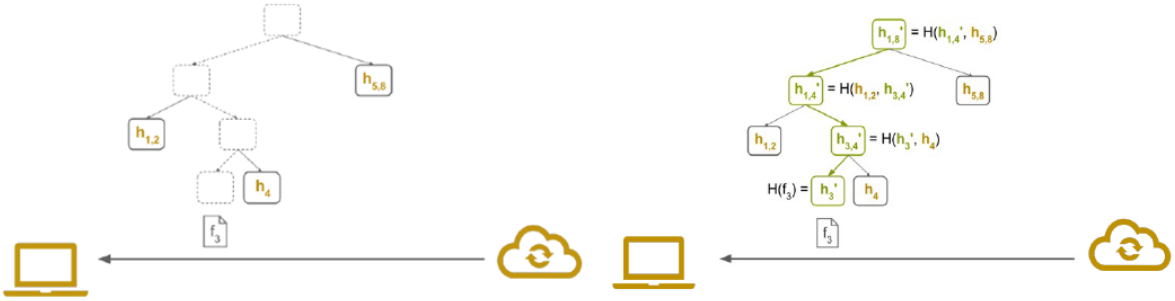
$$h_{1,8} = MHT(f_1, \ldots, f_8)$$

假如我们有 8 个文件，我们将文件两两合并求 hash value，将结果再两两合并求 hash value……最后的结果就是 Merkle tree 的输出 (leaf 节点是 文件，root 节点是输出)。

Merkle proof

我们把 8 个文件和 Merkle tree 传到网盘里，现在我们想验证 file 3 有没有被更改。

我们只需要下载 file 3 f_3，h_4，$h_{1,2}$，$h_{5,8}$，就可以验证 integrity。



我们根据 f_3 和 h_4 算出 $h_{3,4}$，然后根据 $h_{3,4}$ 和 $h_{1,2}$ 算出 $h_{1,4}$，最后根据 $h_{1,4}$ 和 $h_{5,8}$ 算出 $h_{1,8}$。如果 $h_{1,8}$ 和之前的一样，那么 file 3 就没有被更改。

Bitcoin

比特币是 hash function 的另一个应用，它是第一个也是最广泛认可的加密货币。