Blockchain

Nepal

Recap

Why do we need Hash Functions?

You received an important message

1. How can you know that message is from the person it says?
   (Message Authentication)

2. How can you be sure that nobody (perhaps a hacker) altered/changed the message during transmission?
   (Message Integrity)

Solution

Message Authentication + Integrity Protection

=

Message Authentication Codes (MAC)

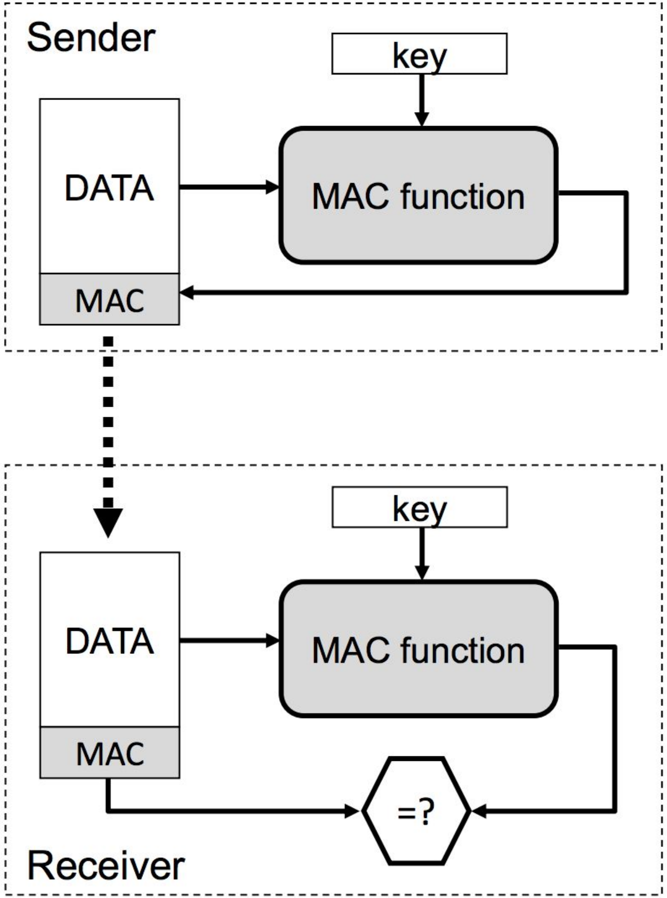
Message Authentication Code

MAC : $\{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^n$

- Can be viewed as hash function with an additional key

- After successful verification of the MAC value, the receiver can be assured that the named sender created the message and it has not be altered

- Examples: HMAC, CBC-MAC etc.



Properties of MAC functions (Desired)

- Key non-recovery
  - Hard to recover key **K**, given one or more message-MAC pairs **(m_i, M_i)**

- Computation Resistance
  - Given one or more message-MAC pairs **(m_i, M_i)**, it is hard to find any message-MAC pairs **(m, M)** such that **$m \neq m_i$**

Attacker model & Attack objectives

Attack Model

- Known message MAC Pairs
- (Adaptive) chosen messages

Attack Objectives

- Forge MAC value
  - Selective forgery
  - Existential forgery
- Recover message key

Key size & MAC value size

Key Size : k bits

- Key complexity : 2^k (brute force attack)

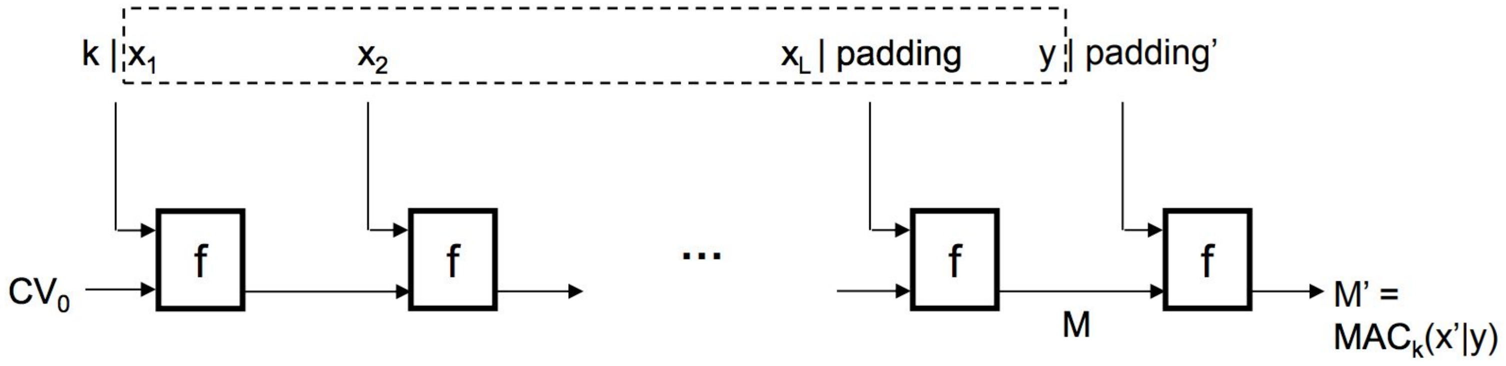
MAC value size : n bits

- MAC value complexity : 2^n

⇒ min(2^k, 2^n) should be sufficiently large

Naive MAC implementations

- Secret prefix method: MACk (x) = H(k|x) **(Insecure - Don't do it)**



- $\mathrm{MAC}_k(x) = H_k(x)$ (where $H_k(.)$ is $H(.)$ with $CV_0 = k$) **(Insecure - Don't do it)**

- Encrypted hash: $\mathrm{MAC}_K(x) = E_K(H(x))$ **(Not recommended)**

Better MACs

- HMAC

- CBC - MAC

HMAC

$\mathrm{HMAC}_k(x) = H(\,(k^+ \oplus \mathrm{opad}) \mid H(\,(k^+ \oplus \mathrm{ipad}) \mid x\,)\,)$

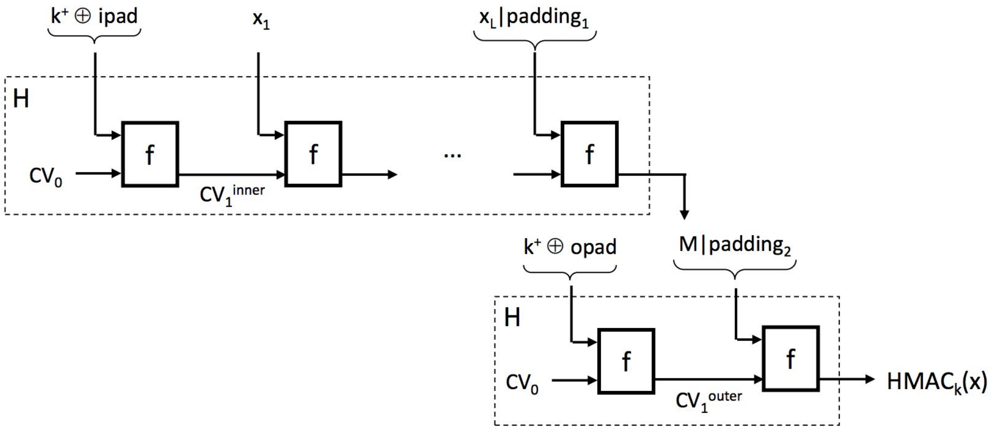
where

– H is a hash function with input block size b and output size n

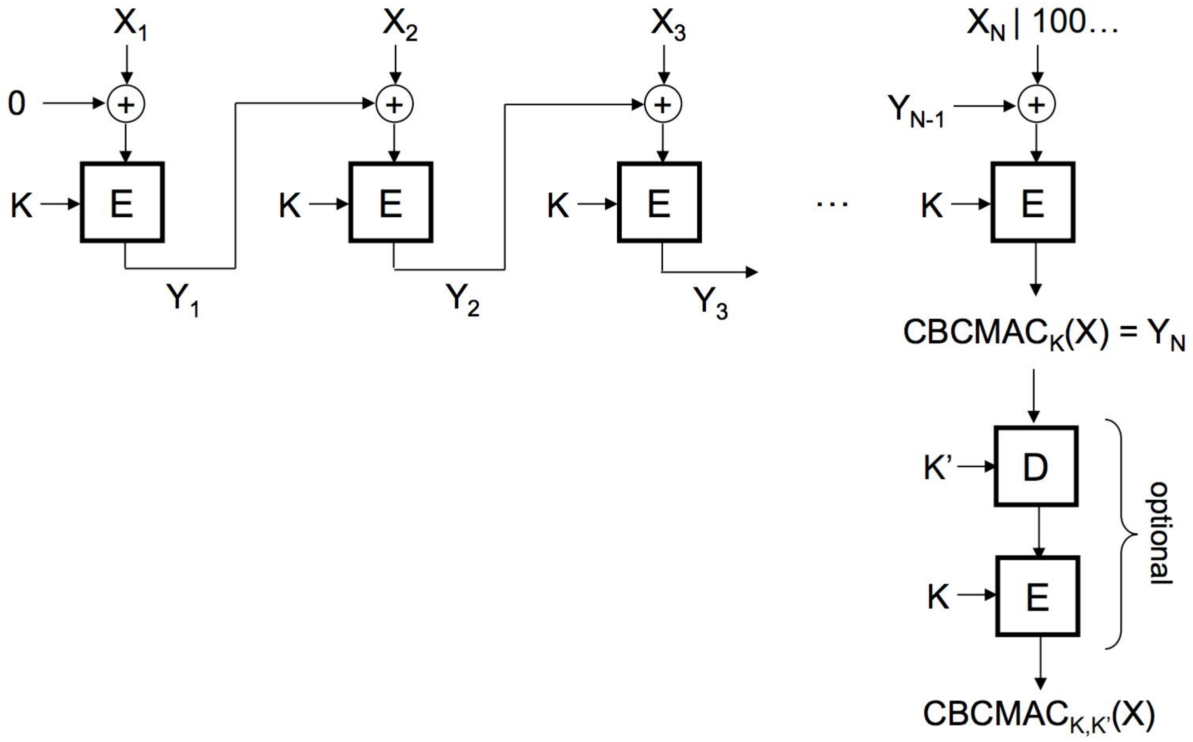
– k^+ is k padded with 0s to obtain a length of b bits

– ipad is 00110110 repeated b/8 times

– opad is 01011100 repeated b/8 times



CBC-MAC


X_1
X_2
X_3
X_N | 100…
0
Y_{N-1}
K
E
…
Y_1
Y_2
Y_3
$\mathrm{CBCMAC}_K(X) = Y_N$
K'
D
optional
$\mathrm{CBCMAC}_{K,K'}(X)$

HMAC in Action

```
// Message to sign
String message = "A quick brown fox jumps over the lazy dog";
// Key used for signing
String key = "Some strong password";
// HMAC algorithm
String HMAC_ALGORITHM = "HmacSha1";

// Generate SecretKey from the user key
SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), HMAC_ALGORITHM);

// Get MAC instance
Mac mac = Mac.getInstance(HMAC_ALGORITHM);
// Initialize the mac with key
mac.init(signingKey);
// Compute MAC
byte[] macBytes = mac.doFinal(message.getBytes());

// show MAC value
String macString = new String(Base64.encode(macBytes));
System.out.println("MAC:" + macString);
```

CBC-MAC in Action

```
// Message to sign
String message = "A quick brown fox jumps over the lazy dog";
byte[] messageBytes = message.getBytes();
// Key used for signing
String key = "passwordpassword";
KeyParameter signingKey = new KeyParameter(key.getBytes());

// Setup MAC with a block cipher
BlockCipher cipher = new AESEngine();
Mac mac = new CBCBlockCipherMac(cipher);

// Initialize the mac with key
mac.init(signingKey);
// Compute MAC
mac.update(messageBytes,0, messageBytes.length);

// get MAC value
byte[] macBytes = new byte[8];
mac.doFinal(macBytes,0);

// show MAC value
String macString = new String(Base64.encode(macBytes));
System.out.println("MAC:" + macString);
```

Exercise

- Write a program to encrypt and authenticate a message. Use two separate keys each for encryption and authentication. You can use AES for encryption and HMAC for MAC.
- Write proper tests to verify encryption and MAC are correct.