ICT Course: Information Security

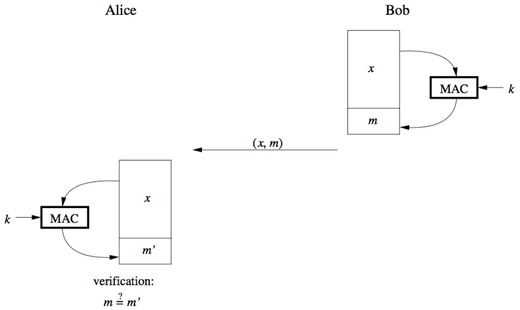
Nguyen Minh Huong

ICT Department, USTH

October 2, 2020

Session 8: Message Authentication Codes and Key Establishment

Message Authentication Codes (MACs) - Overview



Properties:

- Cryptographic checksum or keyed hash function
- Using symmetric-key scheme (much faster than DS)
- Provides:
  - Message integrity
  - Message authentication
  - no non-repudiation

MACs from Hash functions

Overview:

- Use hash function, e.g, SHA-1 as a building block to construct MAC
- Basic idea: key is hashed together with the message, e.g, HMAC
- Two ways of construction:
  - secret prefix MAC: $m = MAC_k(x) = h(k||x)$
  - secret suffix MAC: $m = MAC_k(x) = h(x||k)$

MACs from hash functions- 2 construction ways

Secret Prefix MACs:

Attack Against Secret Prefix MACs

Alice		**Oscar**		**Bob**
				$x = (x_1, \dots, x_n)$ $m = h(k \| \| x_1, \dots, x_n)$
		↯ intercept	$\xleftarrow{(x,m)}$	
		$x_O =$ $(x_1, \dots, x_n, x_{n+1})$ $m_O = h(m \| \| x_{n+1})$		
	$\xleftarrow{(x_O, m_O)}$			
$m' =$ $h(k \| \| x_1, \dots, x_n, x_{n+1})$ since $m' = m_O$ $\Rightarrow$ valid signature!				

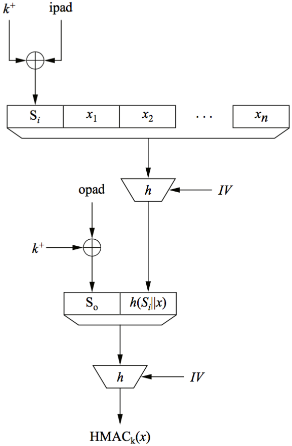
Without knowing the key, attacker can generate a valid MAC by adding an additional block

Secret Suffix MAC

- If attacker can find message x_0 such that $h(x) = h(x_0)$, $m = h(x||k) = h(x_0||k)$ can be found

HMAC

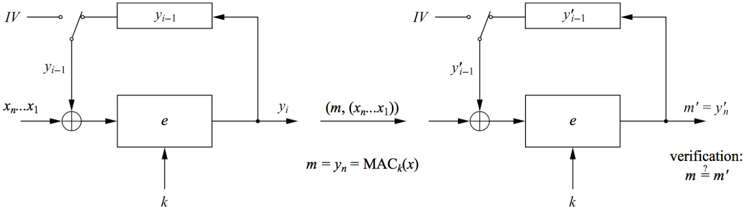
- Proposed by Mihir Bellare, Ran Canetti and Hugo Krawczyk in 1996
- Widely used
- Scheme: inner hash and outer hash



- $k_+ = (k||0...0)$: expended key (b bits)
- ipad = (00110110,00110110, ..., 00110110)
- x_i: message blocks
- opad = (01011100, 01011100, ..., 01011100)
- $HMAC_k(x) = ?$

MACs from Block ciphers

- Using block ciphers to construct MACs
- The most popular approach in practice: a block cipher in CBC mode (CBC-MAC)
- Principle of CBC-MAC:



Key Establishment - Introduction

- Key Establishment: deals with establishing a shared secret between two or more parties.
- Key Establishment methods:
  - Key transport: 1 party generates and distributes a secret key
  - Key Agreement: parties jointly generate a secret key (Ideally, no single party can control what the key value will be)
- Identification of parties is the most important concern

Key Establishment- Introduction

- Key Freshness
- Key Derivation

The n^2 key distribution prolem

Problem:

- Each pair of users needs secured channel
- n users will need: $n(n-1) \approx n^2$ keys $(n.(n-1)/2$ if symmetric keys are used)
- A problem for large networks

$\rightarrow$ Key Distribution?

Symmetric Key Distribution

- Key Distribution Center
- Kerberos Protocol

Key Distribution Center (KDC)

KDC:

- is a server trusted by all users
- shares a secret key, namely Key Encryption Key (KEK) with each user
- KEK is used to securely transmit secret session keys to users

Scheme:

Alice	KDC	Bob
Alice KEK: k_A	**KDC** KEK: k_A, k_B	**Bob** KEK: k_B
$\xrightarrow{\text{RQST}(ID_A, ID_B)}$		
	generate random k_{ses} $y_A = e_{k_A}(k_{ses})$ $y_B = e_{k_B}(k_{ses})$	
$\xleftarrow{y_A}$		$\xrightarrow{y_B}$
$k_{ses} = e_{k_A}^{-1}(y_A)$		$k_{ses} = e_{k_B}^{-1}(y_B)$
$y = e_{k_{ses}}(x)$	$\xrightarrow{y}$	$x = e_{k_{ses}}^{-1}(y)$

KDC - cont

Advantages:

- only n KEKs are maintained long-term
- new user needs to establish only KEK with KDC

Attacks:

- Replay attack
- Key confirmation attack

Kerberos Protocol

Provides:

- user authentication
- Key distribution protocol $\longrightarrow$ key confirmation

Timeliness:

- lifetime of the session key: T
- Time stamp to assure message is recent and not replay attack: T_S

Kerberos Protocol

Alice		**KDC**	**Bob**
KEK: k_A generate nonce r_A		KEK: k_A, k_B	KEK: k_B
	$\xrightarrow{\text{RQST}(ID_A, ID_B, r_A)}$		
		generate random k_{ses} generate lifetime T $y_A = e_{k_A}(k_{ses}, r_A, T, ID_B)$ $y_B = e_{k_B}(k_{ses}, ID_A, T)$	
	$\xleftarrow{y_A, y_B}$		
$k_{ses}, r'_A, T, ID_B = e^{-1}_{k_A}(y_A)$ verify $r'_A = r_A$ verify ID_B verify lifetime T generate time stamp T_S $y_{AB} = e_{k_{ses}}(ID_A, T_S)$			
	$\xrightarrow{y_{AB}, y_B}$		
			$k_{ses}, ID_A, T = e^{-1}_{k_B}(y_B)$ $ID_A', T_S = e^{-1}_{k_{ses}}(y_{AB})$ verify $ID_A' = ID_A$ verify lifetime T verify time stamp T_S
$y = e_{k_{ses}}(x)$	$\xrightarrow{y}$		$x = e^{-1}_{k_{ses}}(y)$

Assymetric Key Distribution

Problem:

- DHKE does not provide authenticated key
- Man-in-the-middle attack against DHKE:

Alice		**Oscar**		**Bob**
choose $a = k_{pr,A}$				choose $b = k_{pr,B}$
$A = k_{pub,A} \equiv \alpha^a \bmod p$				$B = k_{pub,B} \equiv \alpha^b \bmod p$
	$\xrightarrow{A}$	$\lightning$ substitute $\tilde{A} \equiv \alpha^o$	$\xrightarrow{\tilde{A}}$	
	$\xleftarrow{\tilde{B}}$	$\lightning$ substitute $\tilde{B} \equiv \alpha^o$	$\xleftarrow{B}$	
$k_{AO} \equiv (\tilde{B})^a \bmod p$		$k_{AO} \equiv A^o \bmod p$		$k_{BO} \equiv (\tilde{A})^b \bmod p$
		$k_{BO} \equiv B^o \bmod p$		

$\Rightarrow$ Need authentication for the key to assure Alice and Bob to know the key is only from each other $\Rightarrow$ Certificate: $(k_{pub,A}, ID_A)$

Example

We reconsider the Diffie–Hellman key exchange protocol. Assume now that Oscar runs an active man-in-the-middle attack against the key exchange. For the Diffie–Hellman key exchange, use the parameters

$$p = 467, \alpha = 2, and\ a = 228, b = 57$$

for Alice and Bob, respectively.
Oscar uses the value $o = 16$. Compute the key pairs k_{AO} and k_{BO}
(i) the way Oscar computes them, and
(ii) the way Alice and Bob compute them.

Cerificates

- Certificates should bind the identity of a user to their public key
- Applying cryptographic mechanism

$$Cert_A = [(k_{pub,A}, ID_A), sig_{k_{pr}}(k_{pub,A}, ID_A)]$$

- Certificates are provided by trusted third party: Certification Authority (CA):
  - Certificate Generation with user-provided keys: users ask CA to sign
  - Certificate Generation with CA-provided keys: CA generates keys

Certification Generation

Certificate Generation with User-Provided Keys

Alice		**CA**
generate $k_{pr,A}, k_{pub,A}$		
	$\xrightarrow{\text{RQST}(k_{pub,A}, ID_A)}$	
		verify ID_A $s_A = \text{sig}_{k_{pr},CA}(k_{pub,A}, ID_A)$ $\text{Cert}_A = [(k_{pub,A}, ID_A), s_A]$
	$\xleftarrow{\text{Cert}_A}$	

Certificate Generation with CA-Generated Keys

Alice		**CA**
request certificate	$\xrightarrow{\text{RQST}(ID_A)}$	
		verify ID_A generate $k_{pr,A}, k_{pub,A}$ $s_A = \text{sig}_{k_{pr},CA}(k_{pub,A}, ID_A)$ $\text{Cert}_A = [(k_{pub,A}, ID_A), s_A]$
	$\xleftarrow{\text{Cert}_A, k_{pr,A}}$	

Public-Key Infrastructure

- Certificate
- CA, chain of CAs
- Certificate Revocation Lists