Network Security (NetSec)

IN2101 – WS 17/18

Prof. Dr.-Ing. Georg Carle

Dr. Heiko Niedermayer
Dr. Miguel Pardal
Quirin Scheitle
Acknowledgements: Dr. Cornelius Diekmann

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

Chapter 8: Cryptographic Hash Functions and MACs

Chapter 8: Cryptographic Hash Functions and MACs

Motivation

- Common practice in data communications: *error detection code*, to identify random errors introduced during transmission
  - Examples: Parity, Bit-Interleaved Parity, Cyclic Redundancy Check (CRC)

Motivation

- Common practice in data communications: *error detection code*, to identify random errors introduced during transmission
    - Examples: Parity, Bit-Interleaved Parity, Cyclic Redundancy Check (CRC)
- Underlying idea of these codes: add redundancy to a message for being able to *detect*, or even *correct* transmission errors

Motivation

- Common practice in data communications: *error detection code*, to identify random errors introduced during transmission
  - Examples: Parity, Bit-Interleaved Parity, Cyclic Redundancy Check (CRC)
- Underlying idea of these codes: add redundancy to a message for being able to *detect*, or even *correct* transmission errors
- The error detection/correction code of choice and its parameters is a trade-off between:
  - Computational overhead
  - Increase of message length
  - Probability/characteristics of errors on the transmission medium

Motivation

- Essential security goal: *Data integrity*
  - We received message m. Has m been modified by an attacker?

Motivation

- Essential security goal: *Data integrity*
  - We received message m. Has m been modified by an attacker?
- It is a different (and much harder!) problem to determine if m has been *modified on purpose!*

Motivation

- Essential security goal: *Data integrity*
  - We received message m. Has m been modified by an attacker?
- It is a different (and much harder!) problem to determine if m has been *modified on purpose!*
- Consequently, we need to add a code that fulfills some additional properties which should make it *computationally infeasible* for an attacker to tamper with messages

Motivation

- Essential security goal: *Data integrity*
  - We received message m. Has m been modified by an attacker?
- It is a different (and much harder!) problem to determine if m has been *modified on purpose!*
- Consequently, we need to add a code that fulfills some additional properties which should make it *computationally infeasible* for an attacker to tamper with messages
- Outline:
  1. Cryptographic Hash Functions
  2. Message Authentication Codes

Definition

- A function h is called a ***hash function*** if:

Definition

- A function *h* is called a ***hash function*** if:
  - *Compression*: *h* maps an input *x* of arbitrary length to an output *h(x)* of fixed length *n*:
    h: $\{0,1\}^* \rightarrow \{0,1\}^n$
  - *Ease of computation*: Given *h* and *x* it is easy to compute *h(x)*

Definition

- A function *h* is called a ***hash function*** if:
  - *Compression*: *h* maps an input *x* of arbitrary length to an output *h(x)* of fixed length *n*:
    h: $\{0,1\}^* \rightarrow \{0,1\}^n$
  - *Ease of computation*: Given *h* and *x* it is easy to compute *h(x)*
- A function *h* is called a ***one-way function*** if
  - *h* is a *hash function*
  - For all pre-specified outputs y, it is *computationally infeasible* to find an *x* with *h(x) = y*

Definition

- A function h is called a ***hash function*** if:
  - *Compression*: h maps an input x of arbitrary length to an output $h(x)$ of fixed length n:
    h: $\{0,1\}^* \to \{0,1\}^n$
  - *Ease of computation*: Given h and x it is easy to compute $h(x)$
- A function h is called a ***one-way function*** if
  - h is a *hash function*
  - For all pre-specified outputs y, it is *computationally infeasible* to find an x with $h(x) = y$
- Example: given a large prime number p and a primitive root g in Z_p^*
  Let $h(x) = g^x \bmod p$
  Then h is a one-way function

Definition

- A function H is called a ***cryptographic hash function*** if:

Definition

- A function H is called a ***cryptographic hash function*** if:
  1. H is a one-way function (1^{st} *pre-image resistance*):
     For all pre-specified outputs y, it is computationally infeasible to find an x with $H(x) = y$

Definition

- A function H is called a ***cryptographic hash function*** if:
  1. H is a one-way function (1^{st} *pre-image resistance*):
     For all pre-specified outputs y, it is computationally infeasible to find an x with $H(x) = y$
  2. 2^{nd} *pre-image resistance*:
     Given x it is computationally infeasible to find any second input x' with $x \neq x'$ such that $H(x) = H(x')$
     Note: This property is very important for digital signatures.

Definition

- A function H is called a ***cryptographic hash function*** if:
  1. H is a one-way function (1^{st} *pre-image resistance*):
     For all pre-specified outputs y, it is computationally infeasible to find an x with $H(x) = y$
  2. 2^{nd} *pre-image resistance*:
     Given x it is computationally infeasible to find any second input x' with $x \neq x'$ such that $H(x) = H(x')$
     Note: This property is very important for digital signatures.
  3. *Collision resistance*:
     It is computationally infeasible to find any pair (x, x') with $x \neq x'$ such that $H(x) = H(x')$

Comparison to CRC:

- In networking there are codes for error detection.
- Common example: Cyclic redundancy checks (CRC)
  - Based on binary polynomial division with Input / CRC divisor.
  - The remainder of the division is the resulting error detection code.
  - CRC is a fast compression function.

Definition

Comparison to CRC:

- In networking there are codes for error detection.
- Common example: Cyclic redundancy checks (CRC)
  - Based on binary polynomial division with Input / CRC divisor.
  - The remainder of the division is the resulting error detection code.
  - CRC is a fast compression function.
- Why not use CRC?

Definition
Comparison to CRC:

- In networking there are codes for error detection.
- Common example: Cyclic redundancy checks (CRC)
  - Based on binary polynomial division with Input / CRC divisor.
  - The remainder of the division is the resulting error detection code.
  - CRC is a fast compression function.
- Why not use CRC?
  - CRC is <u>not</u> a cryptographic hash function
  - CRC does not provide 2^{nd} pre-image resistance and collision resistance
  - CRC is additive
    - If $x' = x \oplus \triangle$, then $CRC(x') = CRC(x) \oplus CRC(\triangle)$
  - CRC is useful for protecting against noisy channels
  - But not against intentional manipulation

Can Hashing ensure Integrity?

Case:
No attacker

Alice (A)

Bob (B)

$m, H(m)$

ok

Case:
With attacker

Alice (A)

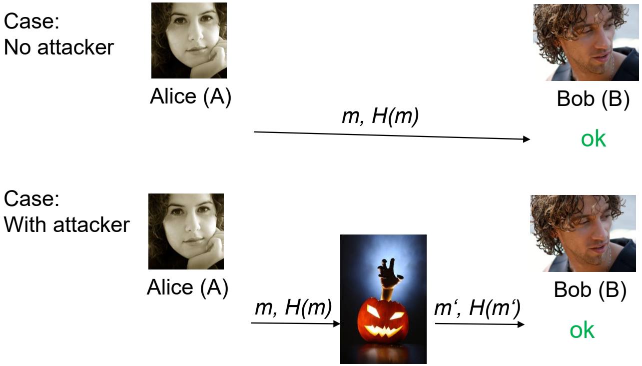
$m, H(m)$

$m', H(m')$

Bob (B)

ok

Can Hashing ensure Integrity?



- Applying a hash function is not sufficient to secure a message.
- *H(m)* needs to be protected.

Can Hashing ensure Integrity?

Case:
No attacker

Alice (A)

share symmetric key K

Bob (B)

m, $MAC_K(m)$

ok

Case:
With attacker

Alice (A)

m, $MAC_K(m)$

m', $MAC_K(m)$

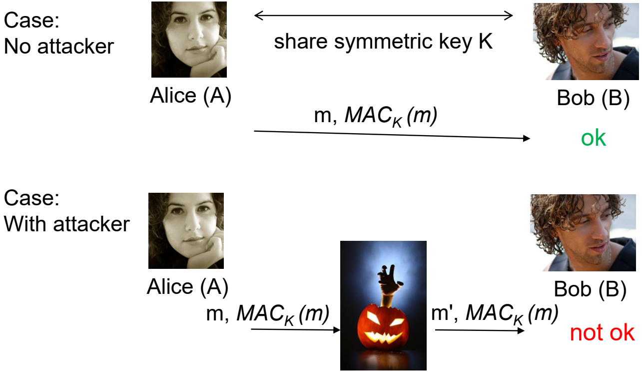
Bob (B)

not ok

- Simply hashing a message and appending the hash is not secure against intentional manipulation (compare with CRC)!

Applications

Can Hashing ensure Integrity?



- Simply hashing a message and appending the hash is not secure against intentional manipulation (compare with CRC)!
- Solution:
  - Include a secret in the hash.
  - Since the secret key k is unknown to the attacker, the attacker cannot compute $MAC_K(m')$ (see next section).

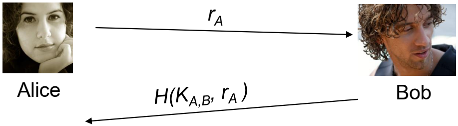
Other applications of cryptographic hash functions which require some caution:

- Pseudo-random number generation
  - The output of a cryptographic hash function is assumed to be uniformly distributed
  - Although this property has not been proven in a mathematical sense for common cryptographic hash functions, such as MD5, SHA-1, it is often used
  - Start with random seed, then hash
    - $b_0 = seed$
    - $b_{i+1} = H(b_i | seed)$

Other applications of cryptographic hash functions which require some caution:

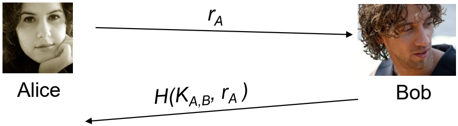
- Pseudo-random number generation
  - The output of a cryptographic hash function is assumed to be uniformly distributed
  - Although this property has not been proven in a mathematical sense for common cryptographic hash functions, such as MD5, SHA-1, it is often used
  - Start with random seed, then hash
    - $b_0 = seed$
    - $b_{i+1} = H(b_i | seed)$
- Encryption
  - Remember: Output Feedback Mode (OFB) - encryption by generating a pseudo random stream, and performing XOR with plain text
  - Generate a key stream as follow:
  - $k_0 = H(K_{A,B} | IV)$
  - $k_{i+1} = H(K_{A,B} | k_i)$
  - The plain text is XORed with the key stream to obtain the cipher text.

- Authentication with a *challenge-response* mechanism



Applications

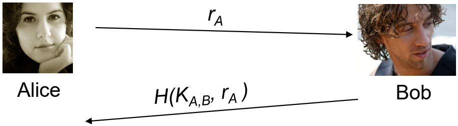
- Authentication with a *challenge-response* mechanism



- Given only Alice and Bob know the shared secret $K_{A,B}$, Alice knows that an attacker is not able to compute $H(K_{A,B}, r_A)$. Therefore the response must be from Bob.
- Mutual authentication can be achieved by a 2nd exchange in opposite direction

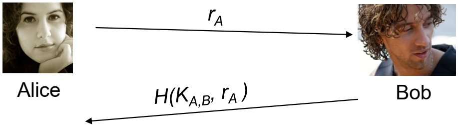
Applications

- Authentication with a *challenge-response* mechanism



- Given only Alice and Bob know the shared secret $K_{A,B}$, Alice knows that an attacker is not able to compute $H(K_{A,B}, r_A)$. Therefore the response must be from Bob.
- Mutual authentication can be achieved by a 2nd exchange in opposite direction
- This type of authentication is based on a authentication method called *challenge-response* and used, for example, by HTTP digest authentication
  - It avoids transmitting the transport of the shared key (e.g. password) in clear text

- Authentication with a *challenge-response* mechanism



- Given only Alice and Bob know the shared secret $K_{A,B}$, Alice knows that an attacker is not able to compute $H(K_{A,B}, r_A)$. Therefore the response must be from Bob.
- Mutual authentication can be achieved by a 2nd exchange in opposite direction
- This type of authentication is based on a authentication method called *challenge-response* and used, for example, by HTTP digest authentication
  - It avoids transmitting the transport of the shared key (e.g. password) in clear text
- Another type of a challenge-response would be, for example, if Bob signs the challenge "r_A" with his private key
- Note that this kind of authentication does not include negotiation of a session key.
- Protocols for key negotiation will be discussed in subsequent chapters.

Common Cryptographic Hash Functions

- Cryptographic Hash Functions:

Common Cryptographic Hash Functions

- Cryptographic Hash Functions:
  - Message Digest 5 (MD5): Considered broken.
    - Invented by R. Rivest, Successor to MD4. Considered broken.

Common Cryptographic Hash Functions

- Cryptographic Hash Functions:
  - Message Digest 5 (MD5): Considered broken.
    - Invented by R. Rivest, Successor to MD4. Considered broken.
  - Secure Hash Algorithm 1 (SHA-1): Considered broken.
    - Old NIST standard.
    - Invented by the National Security Agency (NSA). Inspired by MD4.

Common Cryptographic Hash Functions

- Cryptographic Hash Functions:
  - Message Digest 5 (MD5): Considered broken.
    - Invented by R. Rivest, Successor to MD4. Considered broken.
  - Secure Hash Algorithm 1 (SHA-1): Considered broken.
    - Old NIST standard.
    - Invented by the National Security Agency (NSA). Inspired by MD4.
  - Secure Hash Algorithm 2(SHA-2)
    - Also a NIST standard and invented by the National Security Agency (NSA).
    - The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits
    - SHA-1 attacks have not been successfully extended to SHA-2

Common Cryptographic Hash Functions

- Cryptographic Hash Functions:
  - Message Digest 5 (MD5): Considered broken.
    - Invented by R. Rivest, Successor to MD4. Considered broken.
  - Secure Hash Algorithm 1 (SHA-1): Considered broken.
    - Old NIST standard.
    - Invented by the National Security Agency (NSA). Inspired by MD4.
  - Secure Hash Algorithm 2(SHA-2)
    - Also a NIST standard and invented by the National Security Agency (NSA).
    - The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits
    - SHA-1 attacks have not been successfully extended to SHA-2
  - Secure Hash Algorithm 3 (SHA-3):
    - Current NIST standard (since October 2012).
    - Keccak algorithm by G. Bertoni, J. Daemen, M. Peeters und G. Van Assche.

Definition

- (Cryptographic) hashes alone do not protect against tampering!
- MACs include a secret key K in addition to the message m they aim to protect.
    - Only the persons with knowledge of K can (re-)compute the MAC.

Definition

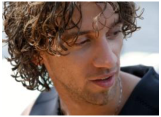
- (Cryptographic) hashes alone do not protect against tampering!
- MACs include a secret key K in addition to the message m they aim to protect.
  - Only the persons with knowledge of K can (re-)compute the MAC.
- Procedure:
  - Sender s computes $MAC_K(m)$.
  - $<m, MAC_K(m)>$ is sent to the receiver r.
  - r receives $<m', MAC_K(m)>$.
    - r can compute $MAC_K(m')$ based on his knowledge of K and m'.
    - If $MAC_K(m') = MAC_K(m)$, he knows that $m = m'$, since nobody else had knowledge of K.

- (Cryptographic) hashes alone do not protect against tampering!
- MACs include a secret key K in addition to the message m they aim to protect.
  - Only the persons with knowledge of K can (re-)compute the MAC.
- Procedure:
  - Sender s computes $MAC_K(m)$.
  - $<m, MAC_K(m)>$ is sent to the receiver r.
  - r receives $<m', MAC_K(m)>$.
    - r can compute $MAC_K(m')$ based on his knowledge of K and m'.
    - If $MAC_K(m') = MAC_K(m)$, he knows that $m = m'$, since nobody else had knowledge of K.
- MACs:
  - Prove message authenticity $\leftrightarrow$ integrity.
  - Do detect tampering.
  - Cannot be forged.
  - Can be replayed.

Application

Alice (A)

share symmetric key K



Bob (B)

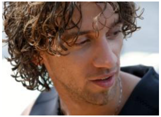
m, $MAC_K(m)$

- Alice protects/authenticates her message m with a MAC function
- Alice has to send m and the MAC value to Bob.

Application

Alice (A)

share symmetric key K



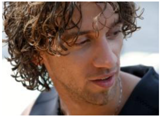
Bob (B)

m, $MAC_K(m)$

- Alice protects/authenticates her message *m* with a MAC function
- Alice has to send *m* and the MAC value to Bob.
- Examples for potential MAC constructions:
  - HMAC
  - CBC-MAC / CMAC
  - Enc_K (h(m)) $\rightarrow$ NO!!

Alice (A)

share symmetric key K



Bob (B)

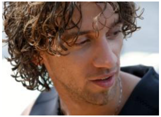
m, $MAC_K(m)$

- Bob can verify the MAC code by using the shared key:
  - He reads Alice's $MAC_K(m)$
  - He can check if his $MAC_K(m')$ matches the one sent by Alice.
  - Only Alice and Bob who know K can do this.

Application

Alice (A)

share symmetric key K



Bob (B)

m, $MAC_K(m)$

- Bob can verify the MAC code by using the shared key:
  - He reads Alice's $MAC_K(m)$
  - He can check if his $MAC_K(m')$ matches the one sent by Alice.
  - Only Alice and Bob who know K can do this.
- Take home message: for authenticity checks the receiver needs to know m and a secure modification check value that it can compare.
  - Think about it: Why is $Enc_K(m)$ usually not sufficient?

Application

- Reasons for constructing MACs from cryptographic hash functions:
  - Cryptographic hash functions generally execute faster than symmetric block ciphers (Note: with AES this is not much of a problem today)
  - There are no export restrictions to cryptographic hash functions

Application

- Reasons for constructing MACs from cryptographic hash functions:
  - Cryptographic hash functions generally execute faster than symmetric block ciphers (Note: with AES this is not much of a problem today)
  - There are no export restrictions to cryptographic hash functions
- Basic idea: “mix” a secret key K with the input and compute a hash value.
- The assumption that an attacker needs to know K to produce a valid MAC nevertheless raises some cryptographic concern:
  - The construction $H(K \parallel m)$ is not secure
  - The construction $H(m \parallel K)$ is not secure
  - The construction $H(K \parallel p \parallel m \parallel K)$ with p denoting an additional padding field does not offer sufficient security

Attack Against an Insecure MAC

- For illustrative purposes, consider the following MAC definition:
  - Input: message $m = (x_1, x_2, ..., x_n)$ with x_i being 128-bit values, and key K
  - Compute $\triangle(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting XOR
  - Output: $MAC_K(m) := Enc_K(\triangle(m))$ with $Enc_K(x)$ denoting AES encryption

Attack Against an Insecure MAC

- For illustrative purposes, consider the following MAC definition:
  - Input: message $m = (x_1, x_2, ..., x_n)$ with x_i being 128-bit values, and key K
  - Compute $\triangle(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting XOR
  - Output: $MAC_K(m) := Enc_K(\triangle(m))$ with $Enc_K(x)$ denoting AES encryption
- The key and the MAC length are both 128 bit, so we would expect an effort of about 2^{127} operations to break the MAC (being able to forge messages).

Attack Against an Insecure MAC

- For illustrative purposes, consider the following MAC definition:
  - Input: message $m = (x_1, x_2, ..., x_n)$ with x_i being 128-bit values, and key K
  - Compute $\triangle(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting XOR
  - Output: $MAC_K(m) := Enc_K(\triangle(m))$ with $Enc_K(x)$ denoting AES encryption
- The key and the MAC length are both 128 bit, so we would expect an effort of about 2^{127} operations to break the MAC (being able to forge messages).
- Unfortunately the MAC definition is insecure:
  - Attacker Eve wants to forge messages. Eve does not know K.
  - Alice and Bob exchange a message (m, $MAC_K(m)$), Eve eavesdrops it.
  - Eve can construct a message m' that yields the same MAC:

Attack Against an Insecure MAC

- For illustrative purposes, consider the following MAC definition:
  - Input: message $m = (x_1, x_2, ..., x_n)$ with x_i being 128-bit values, and key K
  - Compute $\triangle(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting XOR
  - Output: $MAC_K(m) := Enc_K(\triangle(m))$ with $Enc_K(x)$ denoting AES encryption
- The key and the MAC length are both 128 bit, so we would expect an effort of about 2^{127} operations to break the MAC (being able to forge messages).
- Unfortunately the MAC definition is insecure:
  - Attacker Eve wants to forge messages. Eve does not know K.
  - Alice and Bob exchange a message (m, $MAC_K(m)$), Eve eavesdrops it.
  - Eve can construct a message m' that yields the same MAC:
    - Let $y_1, y_2, ..., y_{n-1}$ be arbitrary 128-bit values
    - Define $\boldsymbol{y_n} := y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \triangle(m)$
    - This $\boldsymbol{y_n}$ allows to construct the new message $m' := (y_1, y_2, ..., \boldsymbol{y_n})$

Attack Against an Insecure MAC

- For illustrative purposes, consider the following MAC definition:
  - Input: message $m = (x_1, x_2, ..., x_n)$ with x_i being 128-bit values, and key K
  - Compute $\triangle(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting XOR
  - Output: $MAC_K(m) := Enc_K(\triangle(m))$ with $Enc_K(x)$ denoting AES encryption
- The key and the MAC length are both 128 bit, so we would expect an effort of about 2^{127} operations to break the MAC (being able to forge messages).
- Unfortunately the MAC definition is insecure:
  - Attacker Eve wants to forge messages. Eve does not know K.
  - Alice and Bob exchange a message (m, $MAC_K(m)$), Eve eavesdrops it.
  - Eve can construct a message m' that yields the same MAC:
    - Let $y_1, y_2, ..., y_{n-1}$ be arbitrary 128-bit values
    - Define $\boldsymbol{y_n} := y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \triangle(m)$
    - This $\boldsymbol{y_n}$ allows to construct the new message $m' := (y_1, y_2, ..., \boldsymbol{y_n})$
    - Therefore, $MAC_K(m') = Enc(\triangle(m'))$

$$\begin{aligned} &= Enc_K(y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \boldsymbol{y_n})) \\ &= Enc_K(y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \triangle(m))) \\ &= Enc_K(\triangle(m))) = MAC_K(m) \end{aligned}$$

Attack Against an Insecure MAC

- For illustrative purposes, consider the following MAC definition:
  - Input: message $m = (x_1, x_2, ..., x_n)$ with x_i being 128-bit values, and key K
  - Compute $\triangle(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting XOR
  - Output: $MAC_K(m) := Enc_K(\triangle(m))$ with $Enc_K(x)$ denoting AES encryption
- The key and the MAC length are both 128 bit, so we would expect an effort of about 2^{127} operations to break the MAC (being able to forge messages).
- Unfortunately the MAC definition is insecure:
  - Attacker Eve wants to forge messages. Eve does not know K.
  - Alice and Bob exchange a message (m, $MAC_K(m)$), Eve eavesdrops it.
  - Eve can construct a message m' that yields the same MAC:
    - Let $y_1, y_2, ..., y_{n-1}$ be arbitrary 128-bit values
    - Define $\boldsymbol{y_n} := y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \triangle(m)$
    - This $\boldsymbol{y_n}$ allows to construct the new message $m' := (y_1, y_2, ..., \boldsymbol{y_n})$
    - Therefore, $MAC_K(m') = Enc(\triangle(m'))$

$$\begin{aligned} &= Enc_K(y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \boldsymbol{y_n})) \\ &= Enc_K(y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \triangle(m))) \\ &= Enc_K(\triangle(m))) = MAC_K(m) \end{aligned}$$

  - Therefore, $MAC_k(m)$ is a valid MAC for m', since $\triangle m = \triangle m'$
  - When Bob receives (m', $MAC_K(m)$) from Eve, he will accept it as being originated from Alice.

Common MAC Functions

- MAC Functions:
  - Hash MAC (**HMAC**):
    - Standardized in RFC 2104.
    - Used in conjunction with cryptographic hash functions (e.g. SHA-3)
    - See following slides.

Common MAC Functions

- MAC Functions:
  - Hash MAC (**HMAC**):
    - Standardized in RFC 2104.
    - Used in conjunction with cryptographic hash functions (e.g. SHA-3)
    - See following slides.
  - Cipher Block Chaining MAC (**CBC-MAC**):
    - Recommended by NIST.
    - Based on CBC mode encryption (e.g. with AES).
    - See following slides.

Common MAC Functions

- MAC Functions:
  - Hash MAC (**HMAC**):
    - Standardized in RFC 2104.
    - Used in conjunction with cryptographic hash functions (e.g. SHA-3)
    - See following slides.
  - Cipher Block Chaining MAC (**CBC-MAC**):
    - Recommended by NIST.
    - Based on CBC mode encryption (e.g. with AES).
    - See following slides.
  - Cipher based MAC (**CMAC**):
    - AES-CMAC is standardized by IETF as RFC 4493 and its truncated form in RFC 4494.
    - See following slides.

Common MAC Functions

- MAC Functions:
  - Hash MAC (**HMAC**):
    - Standardized in RFC 2104.
    - Used in conjunction with cryptographic hash functions (e.g. SHA-3)
    - See following slides.
  - Cipher Block Chaining MAC (**CBC-MAC**):
    - Recommended by NIST.
    - Based on CBC mode encryption (e.g. with AES).
    - See following slides.
  - Cipher based MAC (**CMAC**):
    - AES-CMAC is standardized by IETF as RFC 4493 and its truncated form in RFC 4494.
    - See following slides.
  - Poly1305:
    - Standardized in RFC 7539.

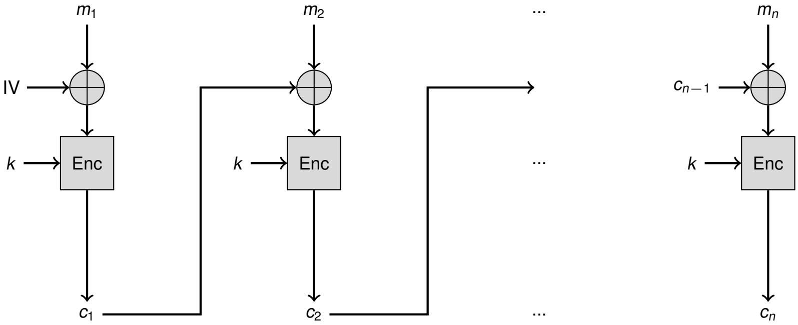
Common MAC Functions: Hash MACs (HMAC)

- The construction *H(K | m | K)*, called prefix-suffix mode, has been used for a while.
  - See for example RFC 1828
  - It has been also used in earlier implementations of the Secure Socket Layer (SSL) protocol (until SSL 3.0)
  - However, it is now considered vulnerable to attack by the cryptographic community.

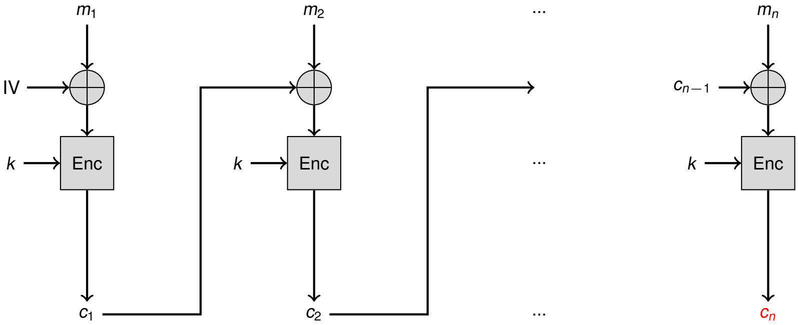
Common MAC Functions: Hash MACs (HMAC)

- The construction *H(K | m | K)*, called prefix-suffix mode, has been used for a while.
  - See for example RFC 1828
  - It has been also used in earlier implementations of the Secure Socket Layer (SSL) protocol (until SSL 3.0)
  - However, it is now considered vulnerable to attack by the cryptographic community.
- The most used construction is **HMAC**: $H\ (\ K \oplus opad\ |\ H\ (\ K \oplus ipad\ |\ m\ ))$
  - The length of the key K is first extended to the block length required for the input of the hash function H by appending zero bytes.
  - Then it is xor'ed respectively with two constants opad and ipad
  - The hash function is applied twice in a nested way.
  - Currently no attacks have been discovered on this MAC function.

- A CBC-MAC is computed by encrypting a message in CBC Mode and taking the last ciphertext block or a part of it as the MAC:



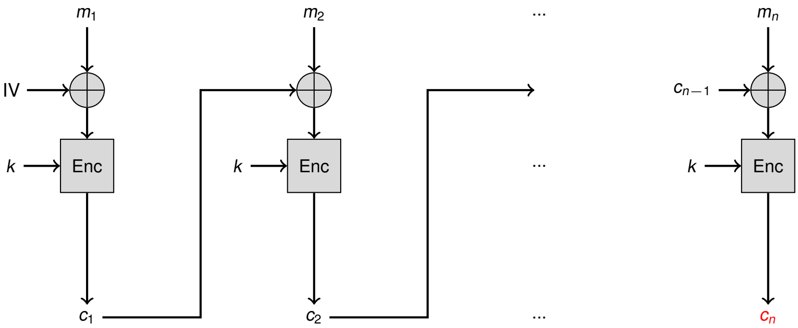
- A CBC-MAC is computed by encrypting a message in CBC Mode and taking the last ciphertext block or a part of it as the MAC:



- $\mathrm{MAC}_k(m) = c_n$ for some publicly known, fixed, IV.

Common MAC Functions: Cipher Block Chaining MACs (CBC-MAC)

- A CBC-MAC is computed by encrypting a message in CBC Mode and taking the last ciphertext block or a part of it as the MAC:



- $\text{MAC}_k(m) = c_n$ for some publicly known, fixed, IV.
- This MAC needs not to be mixed with a secret any further, as it has already been produced using a shared secret K.
- This scheme works with any block cipher (AES, Twofish, 3DES, ...)
- It is used, e.g., for IEEE 802.11 (WLAN) WPA2, many modes in SSL / IPSec use some CBC-MAC construction.

Common MAC Functions: Cipher Block Chaining MACs (CBC-MAC)

- CBC-MAC security
  - CBC-MAC must NOT be used with the same key as for the encryption
  - In particular, if CBC mode is used for encryption, and CBC-MAC for authenticity with the same key, the MAC will be equal to the last cipher text block
  - If the length of a message is unknown or no other protection exists, CBC-MAC can be prone to length extension attacks. CMAC resolves the issue.

Common MAC Functions: Cipher Block Chaining MACs (CBC-MAC)

- CBC-MAC security
  - CBC-MAC must NOT be used with the same key as for the encryption
  - In particular, if CBC mode is used for encryption, and CBC-MAC for authenticity with the same key, the MAC will be equal to the last cipher text block
  - If the length of a message is unknown or no other protection exists, CBC-MAC can be prone to length extension attacks. CMAC resolves the issue.
- CBC-MAC performance
  - Older symmetric block ciphers (such as DES) require more computing effort than dedicated cryptographic hash functions, e.g. MD5, SHA-1 therefore, these schemes are considered to be slower.
  - However, newer symmetric block ciphers (AES) is faster than conventional cryptographic hash functions.
  - Therefore, AES-CBC-MAC is becoming popular.

Common MAC Functions: Cipher-based MACs (CMAC)

- CMAC is a modification of CBC-MAC
  - Compute keys k_1 and k_2 from shared key k.
  - Within the CBC processing
    - XOR complete blocks before encryption with k_1
    - XOR incomplete blocks before encryption with k_2
    - k is used for the block encryption
  - Output is the last encrypted block or the l most significant bits of the last block.
- XCBC-MAC (e.g. found in TLS) is a predecessor of CMAC where k_1 and k_2 are input to algorithm and not derived from k.

Chapter 8: Cryptographic Hash Functions and MACs

Literature

(Beyond the scope of examination)

- B. Coskun, N. Memon, *Confusion/Diffusion Capabilities of Some Robust Hash Functions*, CISS 2006: Conference on Information Sciences and Systems
- H. Krawczyk, M. Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, Internet RFC 2104, February 1997.
- R. Merkle, *One Way Hash Functions and DES*, Proceedings of Crypto ‘89, Springer, 1989.
- Niels Ferguson, Bruce Schneier, *Practical Cryptography*, John Wiley & Sons, 2003
- Peter Selinger, *http://www.mscs.dal.ca/ selinger/md5collision/*
- P. Metzger, *IP Authentication using Keyed MD5*, IETF RFC 1828, August 1995
- R. L. Rivest. *The MD5 Message Digest Algorithm*, Internet RFC 1321, April 1992.
- M. Robshaw. *On Recent Results for MD2, MD4 and MD5*, RSA Laboratories’ Bulletin, No. 4, November 1996.
- Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, *Collision Search Attacks on SHA1*, February 2005
- G. Yuval. *How to Swindle Rabin*, Cryptologia, July 1979.

Literature

- Niels Ferguson, Stefan Lucks, Bruce Schneier, et. al., *Skein Specification v1.1*
- *http://www.skein-hash.info*
- NIST (National Institute for Standards and Technology (USA)), *CRYPTOGRAPHIC HASH ALGORITHM COMPETITION*, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html
- G. Bertoni, J. Daemen, M. Peeters und G. Van Assche, *Cryptographic Sponge Functions* http://sponge.noeke 0.1.pdf
- G. Bertoni, J. Daemen, M. Peeters und G. Van Assche, *Keccak Reference (version 3.0)*, http://keccak.noekeo reference-3.0.pdf
- G. Bertoni, J. Daemen, M. Peeters und G. Van Assche, *Keccak sponge function family main document*, http://keccak.noekeon.org/Keccak-main-2.1.pdf