

# CS458: Introduction to Information Security

## Notes 7: Message Authentication Codes (MACs)

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

[yelmehdwi@iit.edu](mailto:yelmehdwi@iit.edu)

March 6<sup>th</sup> 2024

Slides: Modified from [Christof Paar and Jan Pelzl](#) , [Steven Gordon at Thammasat University](#), & William Stallings : “Cryptography and Network Security, 8 Edition, 2020

- Message Authentication Functions
- The principle behind MACs
- The security properties that can be achieved with MACs
- How MACs can be realized with hash functions and with block ciphers

# Message Authentication

- Receiver wants to verify:
  1. Contents of the message have not been modified (**data authentication**)
  2. Source of message is who they claim to be (**source authentication**)
- Possible attacks on message authentication:
  - **Content modification**
  - **Sequence modification**
    - Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.
  - **Timing modification**
    - Delay or replay of messages
- In summary
  - **Message authentication** is a procedure to verify that received messages come from the alleged source and have not been altered.
  - **Message authentication** may also verify sequencing and timeliness.

# Message Authentication Functions

In any message authentication or DS mechanism, there're two levels of functionality:

- At the lower level
  - There is a function that produces an authenticator
  - authenticator: a value to be used to authenticate a message.
  - *This function is typically a cryptographic algorithm, such as a hash function or a block cipher, that takes the message and the secret key as inputs and produces an output that can be used to verify the authenticity of the message.*
- At the higher level
  - The authenticator produced by the lower-level function is used as a primitive in a higher-level authentication protocol.
  - This protocol enables a receiver to verify the authenticity of the message by comparing the received authenticator with the one generated by the sender.
  - *This higher-level protocol may involve additional steps, such as key exchange, message exchange, and verification procedures, to ensure that the message is authenticated correctly.*

# Message Authentication Functions

- *Overall, the combination of the lower-level authenticator generation function and the higher-level authentication protocol provides a secure way to authenticate messages and ensure that they have not been tampered with during transmission.*

# Message Authentication Functions

- There are types of functions that may be used to produce an authenticator:
  - Message encryption
    - The ciphertext of the entire message serves as its authenticator
  - Hash function
    - A function that maps a message of arbitrary length into a fixed-length hash value which serves as the authenticator

## Message authentication code (MAC)

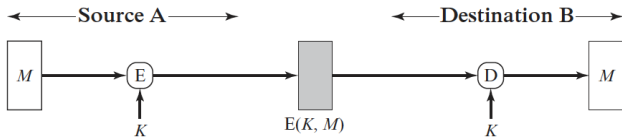
- A function of the message and a secret key that produces a fixed-length value that serves as the authenticator
  - This does not provide a digital signature because both sender and receiver share the same key

# Message Encryption as Authentication

- Message encryption (either symmetric or public-key) by itself can provide a measure of authentication.
- The ciphertext of the entire message serves as its authenticator
  - The message must have come from the sender because the ciphertext can be decrypted using the sender key.
  - None of the bits in the message have been altered because an opponent does not know how to manipulate the bits of the ciphertext to induce meaningful changes to the plaintext.
- The analysis differs for symmetric and public-key encryption schemes.

# Symmetric Key Encryption

- Assumption:
  - Decryption using wrong key or modified ciphertext will produce unintelligible output
- Symmetric key encryption can provide: data authentication and source authentication (as well as confidentiality)

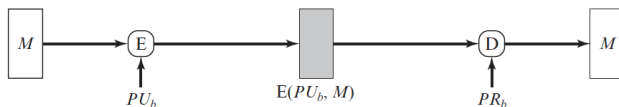


- However, typically authentication is performed separately to encryption for confidentiality
  - Avoid overhead of using encryption when not needed
- Q) How to make sure the message is correct?
  - There must be some internal structure to the plaintext so that the receiver can distinguish between well-formed plaintext and random bits.
- *Although the symmetric key provides message authentication, we would often to use other techniques to provide authentication.*



# Public-Key Encryption

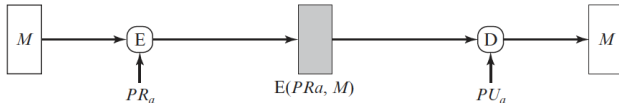
- The straightforward use of public-key encryption provides **confidentiality** but **not authentication**.
- A encrypts M using B's public key, which provides confidentiality



- There must be some internal structure to the plaintext so that the receiver can distinguish between well-formed plaintext and random bits.
- **Q) Do we have message authentication (data integrity/authenticity)?**

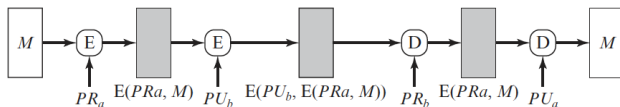
# Public-Key Encryption

- To provide **authentication**, A uses its private key to encrypt the message, and B uses A's public key to decrypt.
  - use the keys of sender not the receiver



# Public-Key Encryption

- To provide both **confidentiality** and **authentication**, A can encrypt  $M$  first using its private key which provides the **digital signature**, and then using B's public key, which provides **confidentiality**



- Disadvantage is that the public-key algorithm must be exercised four times rather than two in each communication
  - too costly, too slow

# More to authentication than simple encryption?

- While encryption can provide some level of authentication by ensuring that only authorized parties can access the encrypted message, there are many situations where alternative authentication schemes are necessary.
  - If a message is broadcast to multiple destinations in a network (such as an alarm signal in a military control center):
    - *it may be more efficient to have a single node responsible for evaluating the authenticity of the message, rather than requiring every recipient to decrypt the message and perform authentication individually.*
  - If one side has a heavy load, it cannot afford to decrypt all messages:
    - *it may be more practical to randomly select messages for authentication rather than decrypting and authenticating every message.*
  - Even if a message is encrypted during transmission, it may be vulnerable to tampering or manipulation once it is decrypted by the recipient.
    - *using a different type of authentication, such as a digital signature, can help protect the message even after it has been decrypted.*
  - Authentication of a computer program in plaintext is an attractive service

# More to authentication than simple encryption?

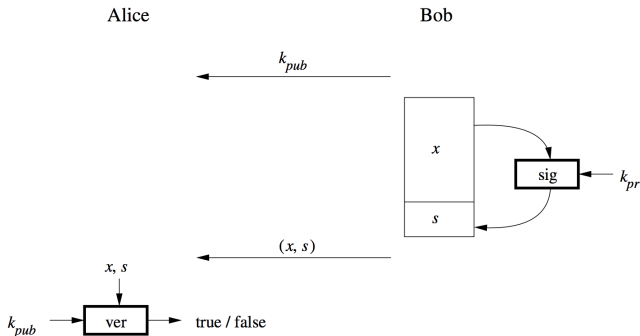
- *Overall, while encryption is an important aspect of secure communication, it is often necessary to use alternative authentication schemes to ensure the authenticity and integrity of a message in different contexts and situations.*

# Introduction to Message Authentication Codes (MACs)

- **Message Authentication Codes** (MACs) are also called “cryptographic checksums”
  - Is small piece of information used to authenticate a message, in other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed.
- Similar to **digital signatures**, **MACs** append an **authentication tag** to a message

# Introduction to Message Authentication Codes (MACs)

- Recall motivation for digital signatures
  - “message authentication”
  - How to do that?



- Let's try the same with symmetric algorithm

# Principle of Message Authentication Codes

- MACs use a symmetric key  $k$  for generation and verification
- When A has a message to send to B, it calculates the MAC as a function of the message and the key:
  - Append small, fixed-size block of data to message: cryptographic checksum or MAC

$$\text{MAC} = F(K, M)$$

where

$M$  = input message

$F$  = MAC function

$K$  = shared secret key

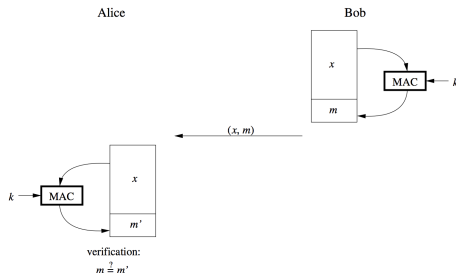
MAC = message authentication code

- The message plus MAC are transmitted to the intended recipient
- The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC.
- The received MAC is compared to the calculated MAC, if the received MAC matches the calculated MAC
  - The receiver is assured that the message has not been altered
  - The receiver is assured that the message is from the alleged sender



# Principle of Message Authentication Codes

- MACs use a symmetric key  $k$  for generation and verification
- MAC function similar to encryption, but does not need to be reversible
  - Easier to design stronger MAC functions than encryption functions
- Computation of a MAC:
  - A MAC is generated by a function  $m = \text{MAC}_k(x)$  that can be computed by anyone knowing the secret key  $k$
- Bob computes  $m = \text{MAC}_k(x)$  and sends  $(x, m)$  to Alice.
- Alice receives  $(x, m')$  and verifies that  $m' = m$ .



# Properties of Message Authentication Codes

1. **Cryptographic checksum**: A MAC generates a cryptographically secure authentication tag for a given message.
2. **Symmetric**: MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.
3. **Arbitrary message size**: MACs accept messages of arbitrary length.
4. **Fixed output length**: MACs generate fixed-size authentication tags.
5. **Message integrity**: MACs provide message integrity: Any manipulations of a message during transit will be detected by the receiver.
6. **Message (data origin) authentication**: The receiving party is assured of the origin of the message.
7. **No nonrepudiation**: Since MACs are based on symmetric principles, they do not provide nonrepudiation

# Requirements for Message Authentication Codes

- MAC is a many-to-one function
  - Messages are arbitrarily long and the MAC has fixed length, thus there will be more than one message with the same MAC
  - But finding these needs to be very difficult
- Computationally easy to compute the MAC
- It is infeasible to find another message with same MAC
  - Knowing  $x$  and  $m = MAC_k(x)$ , it is computationally infeasible to construct another message  $x'$  with  $MAC_k(x) = MAC_k(x')$
- MACs should be uniformly distributed in the sense that for randomly chosen messages,  $x$  and  $x'$ , the probability that is  $MAC_k(x) = MAC_k(x')$  is  $2^{-n}$ 
  - If the attacker chooses a random bit pattern of length  $n$ , the chances of it being the correct signature is  $2^{-n}$
- MAC should depend equally on all bits of the message
  - If  $x'$  is obtained from  $x$  by certain transformations (even switching one bit), then the probability that the two have the same MAC is  $2^{-n}$

# Security of MACs

- Security Requirement
  - Key is secret and difficult to find from pairs of  $(x, \text{MAC})$
  - Given pairs of  $(x, \text{MAC})$ , difficult to find the MAC of another message
- Brute Force Attacks on MACs
- Cryptanalysis

# Brute Force Attacks on MACs

- A brute force attack in MAC is more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs
  - To attack hash code: Given a fixed message  $x$  with  $n$ -bit hash value  $H(x)$ , a brute-force method of finding a collision is to try different inputs until they find an input  $y$  that produces the desired hash value ( $H(y)=H(x)$ ).
- Two lines of attack:
  - Attack the key space
    - If an attacker can determine the MAC key then it is possible to generate a valid MAC value for any input  $x$
    - Effort  $\approx 2^k$
  - Attack the MAC value
    - The attacker can also work on the tag without attempting to recover the key.
    - Objective:
      - to generate a valid tag for a given message or
      - to find a message that matches a given tag.
    - Effort  $\approx 2^n$
- Effort to break MAC:  $\min(2^k, 2^n)$

# Cryptanalysis

- Cryptanalytic attacks differ from brute-force attacks by utilizing weaknesses within the algorithm itself rather than simply trying every possible key.
- Attackers employ various techniques depending on the specific MAC algorithm's structure and underlying principles.
- Unlike hash functions, MAC algorithms exhibit greater structural variety, making it challenging to develop generalized cryptanalysis techniques.
- This diversity stems from the use of different cryptographic primitives like hash functions, block ciphers, or stream ciphers as the foundation for their design.
- Comparing the strength of a MAC algorithm to the effort required for a brute-force attack is a common way to assess its resistance to cryptanalysis.
  - Ideally, a secure MAC should demand a significantly greater effort to break through cryptanalysis than simply trying every possible key combination.

- *Cryptanalytic attacks on MACs can take many different forms. Some attacks may exploit weaknesses in the algorithm's design, such as mathematical properties that allow an attacker to derive the key or tag from a set of known message-tag pairs. Other attacks may exploit weaknesses in the implementation of the algorithm, such as timing or side-channel attacks that allow an attacker to extract information about the key or tag from the physical implementation of the system.*

# MACs Construction

- Two different approaches:
  - We can construct a secure message authentication code for short, fixed length messages. Based on any sort of random function, aka block cipher.
  - We can construct secure message authentication code for arbitrary length messages using hash function.



# MACs Based on Hash Functions: HMAC

- MAC is realized with cryptographic hash functions
- Motivations
  - Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers
  - Library code for cryptographic hash functions is widely available
- Hash values not intended for MAC. They are not protected by secret keys
  - So, additional protection needs to be built on top of the hash value
- **Basic idea:** Key is hashed together with the message

$$m = MAC_k(x) = h(k, x)$$

- **Q:** How exactly do we mix  $k$  and  $x$ ?
- Two possible constructions:
  - **secret prefix MAC:**  $m = MAC_k(x) = h(k || x)$
  - **secret suffix MAC:**  $m = MAC_k(x) = h(x || k)$
- There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm.
- The approach that has received the most support is HMAC

# Length Extension Attack

- Exploits a vulnerability in certain hash functions to generate a valid MAC tag for a message  $(x||y)$  even without knowing the original message  $(x)$ .
- The attacker only needs the original message's hash value  $(H(x))$  and its length.
- This enables forging messages as the attacker can create a new message  $(y)$  and append it to the original message while maintaining a valid MAC tag.
- *Type of attack where an attacker can use  $H(x)$  and the length of  $x$  to calculate  $H(x||y)$  for an attacker-controlled  $y$ , without needing to know the content of  $x$ .*

# Length Extension Attack

- Algorithms like MD5, SHA-1 and most variants of SHA-2 that are based on the Merkle-Damgård construction<sup>1</sup> are susceptible to this kind of attack.
- Merkle-Damgård construction
  - This construction iteratively processes a message in fixed-size blocks using a compression function.
  - An initialization vector (IV) is used to start the process, and the output of each compression function becomes the input for the next iteration.
  - To ensure security, messages are padded with information about the original message's length, a technique known as length padding or Merkle-Damgård strengthening.
- However, despite the padding, the vulnerability arises because the attacker can manipulate the padding and the message to forge a valid MAC tag.

---

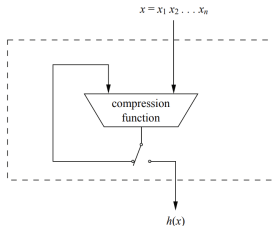
<sup>1</sup> Merkle-Damgård hash function/ construction is a method of building collision-resistant cryptographic hash functions from collision-resistant one-way compression functions.

# Attack Against Secret Prefix MACs

- The attacker obtains the MAC tag ( $m$ ) for an original message ( $x$ ).
- By exploiting the weaknesses in the padding scheme used for length extension, the attacker can append additional data ( $x'$ ) to the original message while calculating a new MAC tag ( $m'$ ).
- This manipulation allows them to forge a new message ( $x||x'$ ) with a valid MAC tag ( $m'$ ), even though they don't possess the secret key.

# Attack Against Secret Prefix MACs

- Assume  $x = (x_1, x_2, \dots, x_n)$



- $m = MAC_k(x) = h(k || x_1 || x_2 || \dots || x_n)$  is computed using Merkle-Damgård hash function/construction.
  - This iterated approach is used in the majority of today's hash functions
- Attack MAC for the message  $x = (x_1, x_2, \dots, x_n, x_{n+1})$ , where  $x_{n+1}$  is an arbitrary additional block, can be constructed from  $m$  without knowing the secret key.

# 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!

- Oscar intercepts  $x = (x_1, x_2, \dots, x_n)$  and  $m$
- Adds  $x_{n+1}$ , and calculates  $m_o = h(m || x_{n+1})$
- Sends  $x = (x_1, x_2, \dots, x_n, x_{n+1})$  and  $m_o$

# Attack Against Secret Suffix MACs

- Assume  $x = (x_1, x_2, \dots, x_n)$
- $m = MAC_k(x) = h(x || k) = h(x_1 || x_2 || \dots || x_n || k)$
- **Attack:**
  - Assume Oscar can find collisions,  $x$  and  $x_o$  such that  $h(x) = h(x_o)$ , then  $m = h(x || k) = h(x_o || k)$
  - Can replace  $x$  with  $x_o$
- **Q:** Is this a problem, i.e., does Oscar gain anything?

It depends on the parameters used in the construction

⇒ Compare brute-force effort with collision-finding effort:

- Example:  $h() \rightarrow$  SHA-1 (160 bit output)  
 $|K| = 128\text{-bit} \rightarrow$  we expect attacker complexity of  $2^{128}$
- but collision search takes  $\approx \sqrt{2^{160}} = 2^{80}$  steps (birthday paradox)
- ⇒ cryptographically, make MAC attackable by birthday attack.  
i.e., the **secret suffix** method also does not provide the security one would like to have from a MAC construction

# HMAC Design Objectives

- There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm.
- The one approach that gained wide support is HMAC (RFC 2104) included in IP security and SSL
- RFC 2104<sup>1</sup> lists the following objectives for HMAC:
  - To use, without modifications, available hash functions
  - To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
  - To preserve the original performance of the hash function without incurring a significant degradation
  - To use and handle keys in a simple way
  - To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function
    - Main advantage of of HMAC over other proposed hash-based scheme:
      - *HMAC can be proven secure provided that the embedded hash function meets certain cryptographic strength criteria..*

---

<sup>1</sup> HMAC: Keyed-Hashing for Message Authentication



# HMAC Construction

- Hash-based message authentication code
- Proposed by Bellare, Canetti and Krawczyk in 1996
- Widely used in practice, e.g., SSL/TLS
- **idea:**
  - Append a secret key to the message and compute the hash value
    - To avoid a brute-force attack, apply the hash twice to mangle thoroughly the bits of the key with those of the message
    - i.e., use two nested secret prefix MACs
  - Roughly,  $h(k || h(k || x))$
  - HMAC uses two passes of hash computation (scheme consists of an inner and outer hash)
  - The secret key is first used to derive two keys (inner and outer).

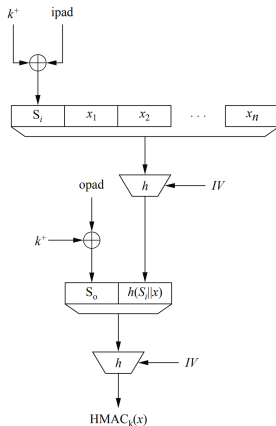
# HMAC Construction

- In reality:  $HMAC_k(x) = h[(k^+ \oplus opad) || h((k^+ \oplus ipad) || x)]$ 
  - The key  $k$  is padded with zeros to the block size of the hash function, and  $ipad$  and  $opad$  are constants of that block size.
  - $k^+$  is expanded key  $k$ 
    - If the length of the key is greater than hash block size, then it will be given as input to the hash function.
    - If the original length of key is smaller than hash block size, zero extended on the left to match hash block size ( $k^+ = 00...00 || k$ ).
  - Expanded key  $k^+$  is XORed with inner and outer pads
  - Padding<sup>1</sup>:
    - Let  $B$  be the block length of hash in bytes.
    - $0x36$  repeated  $B$  times for  $ipad$
    - $0x5c$  repeated  $B$  times for  $opad$   
e.g.,  $B = 64$  for MD5 and SHA-1
- The first pass of the algorithm produces an internal hash derived from the message and the inner key.
- The second pass produces the final HMAC code derived from the inner hash result and the outer key.

---

<sup>1</sup> The values of  $ipad$  and  $opad$  are not critical to the security of the algorithm, but were defined in such a way to have a large Hamming distance from each other and so the inner and outer keys will have fewer bits in common.

# HMAC Construction



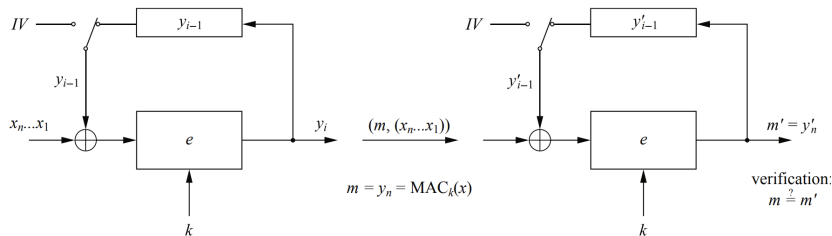
- Note: Message is only processed in inner hash!.
- HMAC is provably secure which means (informally speaking): The MAC can only be broken if a collision for the hash function can be found.

# HMAC and Strong Collisions

- Birthday attacks don't make sense in HMAC scenario
  - Attacker would need to know  $k$  to generate candidate message/hash pairs
- HMAC provides better immunity against **length extension attacks**.
  - *HMAC's design, which involves double hashing and key-based processing, makes it inherently resistant to length extension attacks. The key plays a vital role in ensuring that the hash output is dependent on both the content and the length of the message, which significantly enhances the security of the authentication code.*

# MACs from Block Ciphers

- We can construct a secure message authentication code for short fixed length messages. Based on any sort of random function.
- An alternative method is to construct MACs from block ciphers (e.g., AES)
- Popular: Use AES in CBC (cipher block chaining) mode
- Cipher block chaining message authentication code (CBC-MAC)



- MAC Generation

- Divide the message  $x$  into blocks  $x_i$
- Compute first iteration  $y_1 = E_k(x_1 \oplus IV)$
- Compute  $y_i = E_k(x_i \oplus y_{i-1})$  for the next blocks
- Final block is the MAC value:  $m = MAC_k(x) = y_n$

- MAC Verification

- Repeat MAC computation ( $m'$ )
- Compare results:
  - If  $m' = m$ , the message is verified as correct
  - If  $m' \neq m$ , the message and/or the MAC value  $m$  have been altered during transmission

# Summary: Protecting messages

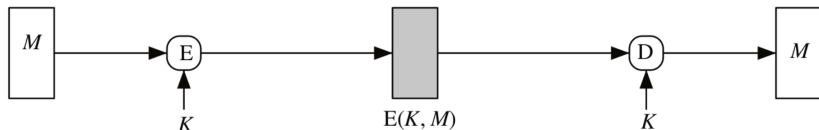
- Encryption protects message **confidentiality**.
  - prevents unauthorized disclosure
- We also wish to protect message **integrity** and **authenticity**.
  - **Integrity** means that the message has not been altered.
    - detect unauthorized writing (i.e., modification of data)
  - **Authenticity (Source Authentication)** means that the message is genuine.
- Encryption alone does **not** provide integrity.
  - *a tampered message can (usually) still decrypt, but the result will usually be garbage*
- **Q:** When data integrity is more important than confidentiality?
  - Example: Inter-bank fund transfers  
Confidentiality may be nice, integrity is **critical**

# Summary: Protecting integrity and authenticity

- Integrity and authenticity are protected using symmetric or asymmetric methods.
- A **digital signature** or a **message authentication code** (MAC) is a string  $s$  that binds an individual or other entity **Alice** with a message  $x$ .
  - The recipient of the message verifies that  $s$  is a valid signature of **Alice** for message  $x$ .
  - It should be hard for **Oscar** to create a valid signature  $s'$  for a message  $x'$  without knowledge of **Alice**'s secret information.



# Summary: Symmetric Encryption: Confidentiality and Authentication



- Confidentiality: only Bob (and Alice) can recover plaintext
- Source Authentication: Alice is only other user with key; must have come from Alice
- Data Authentication (integrity): successfully decrypted; data has not been modified
- Assumption: decryptor can recognize correct plaintext

# Recognizing Correct Plaintext

- Example 1:

- Bob receives ciphertext (supposedly from Alice, using shared secret key K)

CIAEVGDCYXHABCNDSEBENRCYBIDER

- Bob decrypts with key K to obtain plaintext:

SECURITYANDCRYPTOGRAPHYAREFUN

- Was the plaintext encrypted with key K (and hence sent by Alice)?
    - Is the ciphertext received the same as the ciphertext sent by Alice?
  - *The typical answer for above is yes, the plaintext was sent by Alice and nothing has been modified. This is because the plaintext “makes sense”. Our knowledge of most ciphers (using the English language) is that if the wrong key is used or the ciphertext has been modified, then decrypting will produce an output that does not make sense (not a combination of English words).*

# Recognizing Correct Plaintext

- Example 2:

- Bob receives ciphertext (supposedly from Alice, using shared secret key K)

QEFPPQEBTOLKDJBPXDBPLOOVX

- Bob decrypts with key K to obtain plaintext:

FTUEUEFTQIDAZSYQEEMSQEADDKM

- Was the plaintext encrypted with key K (and hence sent by Alice)?
    - Is the ciphertext received the same as the ciphertext sent by Alice?
  - *Based on the previous argument, the answer is no. Or more precise, either the plaintext was not sent by Alice, or the ciphertext was modified along the way. This is because the plaintext makes no sense, and we were expected it to do so.*

# Recognizing Correct Plaintext

- Example 3:

- Bob receives ciphertext (supposedly from Alice, using shared secret key K)

0110100110101101010110111000010

- Bob decrypts with key K to obtain plaintext:

0101110100001101001010100101110

- Was the plaintext encrypted with key K (and hence sent by Alice)?
  - Is the ciphertext received the same as the ciphertext sent by Alice?
- *This is harder. We cannot make a decision without further understanding of the expected structure of the plaintext. What are the plaintext bits supposed to represent? A field in a packet header? A portion of a binary file? A random key? Without further information, the receiver does not know if the plaintext is correct or not. Therefore does not know if the ciphertext was sent by Alice and has not been modified.*

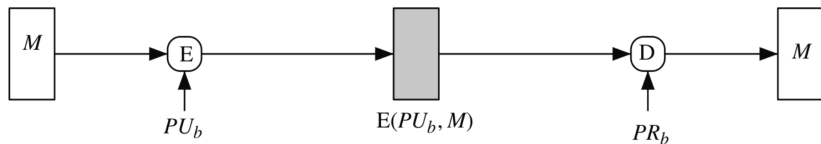
# Recognizing Correct Plaintext

- Example 1:
  - Assume the message is English
  - Plaintext had expected structure; assume the plaintext is correct
    - Sent by **Alice** and has not been modified
- Example 2:
  - Assume the message is English
  - Plaintext had no structure in expected language; assume plaintext is incorrect
    - Either not sent by **Alice** or modified
- Example 3:
  - Binary data, e.g. image, compressed file
  - Cannot know whether correct or incorrect

# Recognizing Correct Plaintext

- Valid plaintexts should be small subset of all possible messages
  - E.g.  $26^n$  possible messages of length  $n$ ; only small subset are valid English phrases
- BUT automatically detecting structure can be be challenging, especially for short messages or those lacking clear grammatical rules.
- Authentication should be possible without decryptor having to know the context of the message being transferred. Traditional decryption often requires some understanding of the data format.
- Solutions
  - Add structured information: Adding redundancy or error-detecting codes (like checksums or CRCs) to the message can help identify corrupted data during transmission. This provides a basic level of integrity verification.
  - Using a MAC algorithm like HMAC (Hash-based Message Authentication Code) offers a stronger solution.
- Plaintext messages have structure
  - *Recognizing correct plaintext is possible if there is some structure in the original plaintext.*

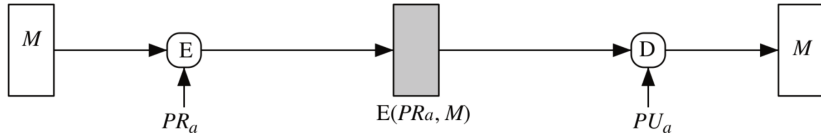
# Summary: Public-Key Encryption: Confidentiality



- Only provides confidentiality
- Same assumption as before: plaintext must have structure so can be recognized as correct or incorrect

# Summary: Public-Key Encryption: Authentication and Signature

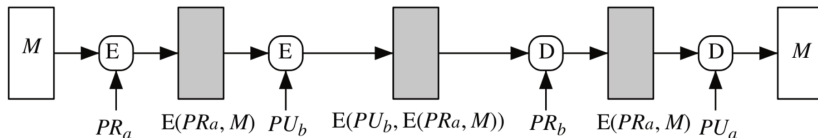
- **Data authentication:** (data has not been modified)
- **Source authentication:** proof that message came from Alice





# Summary: Public-Key Encryption for Authentication

- Confidentiality, authentication and signature



# Summary: Non-non-repudiation

- Alice orders 100 shares of stock from Bob.
- Alice computes MAC using symmetric key.
- Stock drops, Alice claims she did not order.
- Q: Can Bob prove that Alice placed the order?
  - No! Bob also knows the symmetric key, so he could have forged the MAC.
- Problem: Bob knows Alice placed the order, but he can't prove it.

# Summary: Non-repudiation

- **Can an asymmetric scheme help?**
- Alice orders *100* shares of stock from Bob.
- Alice **signs** order with her private key.
- Stock drops, Alice claims she did not order.
- **Q:** Can Bob prove that Alice placed the order?
  - **Yes!** Alice's private key used to sign the order - only Alice knows her private key.
  - Of course, this assumes Alice's private key has not been lost/stolen.

# Summary: Non-repudiation

- **Non-repudiation** refers to a state where the author of a statement will not be able to successfully challenge the authorship of the statement.
- What is the relationship between **authenticity** and **non-repudiation**?
  - **Authenticity**: **Alice** interacts with **Bob** and convinces him that a message  $x$  truly came from her.
  - **Non-repudiation**: Same as above but now **Bob** can convince **Charlie** too.

# Summary

- Message authentication requirements
- MACs provide two security services, message and source authentication using symmetric ciphers.
- MACs are widely used in protocols.
- Both of these services are also provided by digital signatures, but MACs are much faster
- MACs do not provide non-repudiation.
- In practice, MACs are either based on block ciphers or on hash functions.
- HMAC is a popular MAC used in many practical protocols such as Transport Layer Security (TLS) - indicated by a small lock in the browser.

- Understanding Cryptography: A Textbook for Students and Practitioners *available online*
  - Chapter 12: Message Authentication Codes (MACs)