Systems' Security | *Segurança de Sistemas*

Message Authentication Codes

Miguel Frade

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Overview

Learning Objectives

After this chapter, you should be able to:

1. List and explain the possible attacks that are relevant to message authentication.
2. Define the term message authentication code.
3. List and explain the requirements for a message authentication code.
4. Explain the concept of authenticated encryption.

Introduction

Message Authentication

- one of the most complex areas of cryptography
- one approach is the usage of **Message Authentication Code** (MAC)
    - can be built from cryptographic hash functions
    - or built using a block cipher mode of operation
    - or a new approach known as authenticated encryption

Types of attacks in the context of communications across a network:

- attacks to the confidentiality
  - **Disclosure** – release of message contents to any person or process not possessing the appropriate authorization (by means of a cryptographic key)
  - **Traffic analysis** – discovery of the pattern of traffic between parties (determine the frequency and duration of connections, the number and length of messages between parties)

Types of attacks in the context of communications across a network:

- attacks to the confidentiality
  - **Disclosure** – release of message contents to any person or process not possessing the appropriate authorization (by means of a cryptographic key)
  - **Traffic analysis** – discovery of the pattern of traffic between parties (determine the frequency and duration of connections, the number and length of messages between parties)
- attacks to the non-repudiation
  - **Source repudiation** – denial of transmission of message by source
  - **Destination repudiation** – Denial of receipt of message by destination

Types of attacks in the context of communications across a network (continuation):

- attacks to the authentication
  - **Masquerade** – insertion of messages into the network from a fraudulent source (also known as impersonification)
  - **Content modification** – changes to the contents of a message, including insertion, deletion, transposition, and modification
  - **Sequence modification** – any modification to a sequence of messages between parties, including insertion, deletion, and reordering
  - **Timing modification** – delay or replay of messages

Types of attacks in the context of communications across a network (continuation):

- attacks to the authentication
  - **Masquerade** – insertion of messages into the network from a fraudulent source (also known as impersonification)
  - **Content modification** – changes to the contents of a message, including insertion, deletion, transposition, and modification
  - **Sequence modification** – any modification to a sequence of messages between parties, including insertion, deletion, and reordering
  - **Timing modification** – delay or replay of messages

Message Authentication

- is a procedure to verify that received messages come from the alleged source and have not been altered and may also verify sequencing and timeliness.
- a digital signature is an authentication technique that also includes measures to counter repudiation by the source.

Message authentication strategies

1. **hash value** – the hash value serves as an authenticator, but must me protected (as stated on the previous chapter)

Message authentication strategies

1. **hash value** – the hash value serves as an authenticator, but must me protected (as stated on the previous chapter)
2. **message encryption** – the ciphertext of the entire message serves as its authenticator

Message authentication strategies

1. **hash value** – the hash value serves as an authenticator, but must me protected (as stated on the previous chapter)
2. **message encryption** – the ciphertext of the entire message serves as its authenticator
3. **message authentication code** (MAC) – a function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Message Encryption as an Authenticator

Symmetric encryption

- $A \rightarrow B : \quad E_k(M)$
- provides confidentiality
- if A and B are the only ones that know the key k, then can we say that the message could only be sent by A?

Symmetric encryption

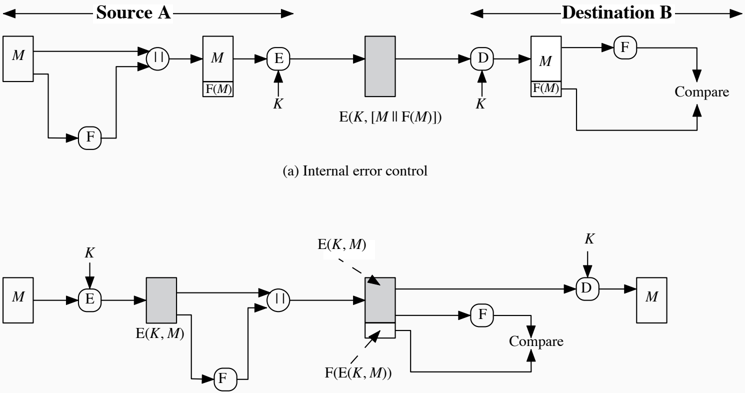
- $A \rightarrow B : \quad E_k(M)$
- provides confidentiality
- if A and B are the only ones that know the key k, then can we say that the message could only be sent by A?
  - **no!**
  - the decryption algorithms are blind, and can “decrypt” any pattern of bits
  - this means that an attacker could perform a DoS by sending random messages that once decrypted don’t have any meaning

Symmetric encryption

- $A \rightarrow B: \quad E_k(M)$
- provides confidentiality
- if A and B are the only ones that know the key k, then can we say that the message could only be sent by A?
    - **no!**
    - the decryption algorithms are blind, and can “decrypt” any pattern of bits
    - this means that an attacker could perform a DoS by sending random messages that once decrypted don’t have any meaning
- solution: add some form of error control

Symmetric encryption + error control

- internal error control provides authentication
- external error control, may, or may not, provide authentication depending on the used function



(a) Internal error control

(b) External error control

Public key encryption

- $A \rightarrow B: \quad E_{PU_B}(M)$
  - provides confidentiality, but not authentication

Public key encryption

- $A \rightarrow B: \quad E_{PU_B}(M)$
    - provides confidentiality, but not authentication
- $A \rightarrow B: \quad E_{PR_A}(M)$
    - provides authentication and non-repudiation, but not confidentiality
    - for the authentication some form of error control must be added

Public key encryption

- $A \rightarrow B : \quad E_{PU_B}(M)$
  - provides confidentiality, but not authentication
- $A \rightarrow B : \quad E_{PR_A}(M)$
  - provides authentication and non-repudiation, but not confidentiality
  - for the authentication some form of error control must be added
- $A \rightarrow B : \quad E_{PU_B}[E_{PR_A}(M)]$
  - provides confidentiality, authentication and non-repudiation
  - this approach is very slow

Message Authentication Code as an Authenticator

Message Authentication Code (MAC)

- use of a secret key to generate a small fixed-size block of data
- known as a cryptographic checksum or MAC
- this value is appended to the message
- does not provide a digital signature, because both sender and receiver share the same key

$$mac = C_k(M)$$

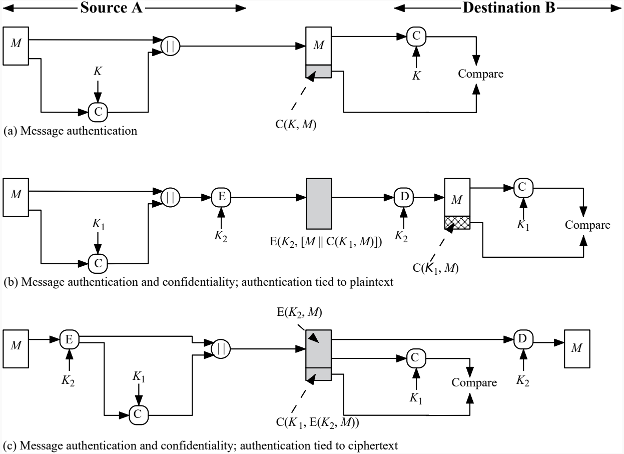
M input message

C MAC function

k shared secret key

mac message authentication code value
also known as *tag*

Message Authentication Code (MAC) – basic uses



Short representations

(a) $A \rightarrow B: \quad M \,\|\, C_k(M)$
without confidentiality

(b) $A \rightarrow B: \quad E_{k_2}[M \,\|\, C_{k_1}(M)]$
internal control

(c) $A \rightarrow B: \quad E_{k_2}(M) \,\|\, C_{k_1}[E_{k_2}(M)]$
external control

Message Authentication Code

Cryptographic requirements

- for any given M, it is computationally infeasible to find $N \neq M$ with $C_k(N) = C_k(M)$
  an attacker should not able to construct a new message to match a given tag without knowing the key

Cryptographic requirements

- for any given M, it is computationally infeasible to find $N \neq M$ with $C_k(N) = C_k(M)$
  an attacker should not able to construct a new message to match a given tag without knowing the key

- it is computationally infeasible, without knowing the key, to find any pair (M, N) with $M \neq N$, such that $C_k(M) = C_k(N)$ – to prevent brute-force attacks based on chosen plaintext

Cryptographic requirements

- for any given M, it is computationally infeasible to find $N \neq M$ with $C_k(N) = C_k(M)$
  an attacker should not able to construct a new message to match a given tag without knowing the key

- it is computationally infeasible, without knowing the key, to find any pair (M, N) with $M \neq N$, such that $C_k(M) = C_k(N)$ – to prevent brute-force attacks based on chosen plaintext

- the authentication algorithm should not be weaker with respect to certain parts of the message than others. If this were not the case, then an opponent who had M and $C_k(M)$ could attempt variations on M at the known “weak spots” with a likelihood of early success at producing a new message that matched the old tags

Security of MACs

- resistance to brute-force attacks
  - the assessment of strength is similar to that for symmetric encryption algorithms
  - the level of effort for brute-force attack on a MAC algorithm can be expressed as $min(2^k, 2^n)$, $k =$ number of bits of the key and $n =$ number of bits of the tag
  - the key length and tag length should be: $min(k, n) \geqslant 128$ bits

Security of MACs

- resistance to brute-force attacks
  - the assessment of strength is similar to that for symmetric encryption algorithms
  - the level of effort for brute-force attack on a MAC algorithm can be expressed as $min(2^k, 2^n)$, $k =$ number of bits of the key and $n =$ number of bits of the tag
  - the key length and tag length should be: $min(k, n) \geqslant 128$ bits
- resistance to cryptanalysis
  - there is much more variety in the structure of MACs, so it is not possible to generalize about the cryptanalysis of MACs
  - there are far less works done on studying this type of attacks when compared whit hash functions

MAC Algorithms

There are two types of MACs:

- **block cipher-based** – can use any symmetric encryption algorithm
  - traditionally been the most common approach to constructing a MAC

There are two types of MACs:

- **block cipher-based** – can use any symmetric encryption algorithm
  - traditionally been the most common approach to constructing a MAC
- **hash based** – can use any hash algorithm (also known as keyed hash)
  - generally execute faster in software than symmetric block cipher-based
  - became more popular after the mandatory use of HMAC in IPsec

Hash based MAC algorithms

- hash function such as SHA was not designed for use as a MAC because it does not rely on a secret key
- there are many ways to incorporate a secret key into an existing hash algorithm
- the approach that has received the most support is HMAC

Hash based MAC algorithms

- hash function such as SHA was not designed for use as a MAC because it does not rely on a secret key
- there are many ways to incorporate a secret key into an existing hash algorithm
- the approach that has received the most support is HMAC

HMAC

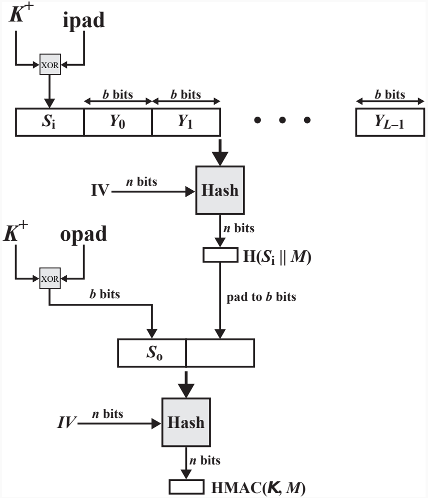
- is the mandatory-to-implement MAC for IP security
- is used in other Internet protocols, such as SSL/TLS
- is publish as RFC 2104 and NIST standard (FIPS 198)

Hash based MAC algorithms

HMAC structure

1. K^+ = append zeros to the left end of k to create a b-bit string (*e.g.*, if K length = 160 bits and $b = 512$, then K will be appended with $(512 - 160) = 352$ bits with zeroes)
2. $S_i = K^+ \oplus ipad$, to produce the b-bit block ($ipad = 00110110$ repeated $b/8$ times)
3. append M to S_i
4. apply hash to the result of step 3: $H(S_i \parallel M)$
5. $S_o = K^+ \oplus opad$, to produce the b-bit block ($opad = 01011100$ repeated $b/8$ times)
6. Append the hash result from step 4 to S_o
7. Apply H to the stream generated in step 6 and output the result

$$HMAC_k(M) = H[K^+ \oplus opad \parallel H(K^+ \oplus ipad) \parallel M]$$



Hash based MAC algorithms

Security of HMAC

- depends on the cryptographic strength of the hash function
- if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one

Hash based MAC algorithms

Security of HMAC

- depends on the cryptographic strength of the hash function
- if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one
- attacking HMAC is harder than attacking a hash function
    - the attacker cannot generate message/code pairs because he does not know k
    - therefore, the attacker must observe a sequence of messages generated by HMAC under the same key
    - for a hash of 128 bits, this requires 2^{64} observed blocks (2^{72} bits) with the same key
    - on a 1 Gbps link, one would need to observe a continuous stream of messages with no change in key for about 150 000 years in order to succeed
    - for that reason the use of MD5 is acceptable inside an HMAC

Block-cipher based MAC algorithms

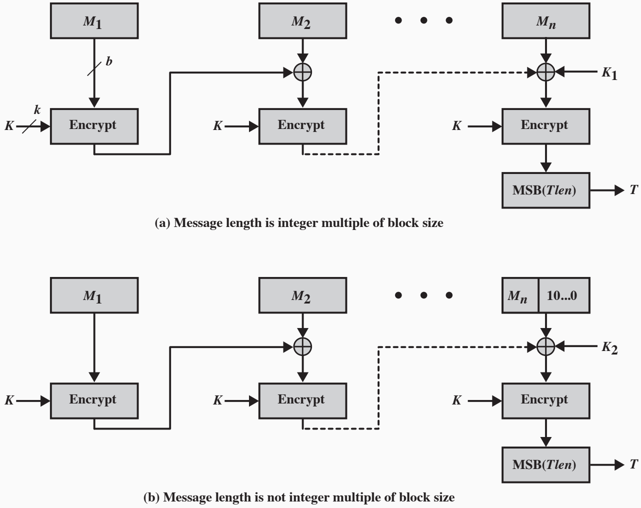
- use a symmetric cipher algorithm as its base
- symmetric cipher algorithms already have a key as input
- used to inside authenticated encryption schemes
- 2 proposals:
  - Data Authentication Algorithm (DAA) – based on DES and no longer safe to use
  - Cipher-based Message Authentication Code (CMAC) – a refinement of DAA, can be used with 3DES and AES

Block-cipher based MAC algorithms

CMAC structure

$$C_1 = E_k(M_1)$$
$$C_2 = E_k(M_2 \oplus C_1)$$
$$C_3 = E_k(M_3 \oplus C_2)$$
...
$$C_n = E_k(M_n \oplus C_{n-1} \oplus k_1)$$
$$T = MSB_{T_{len}}(C_n)$$

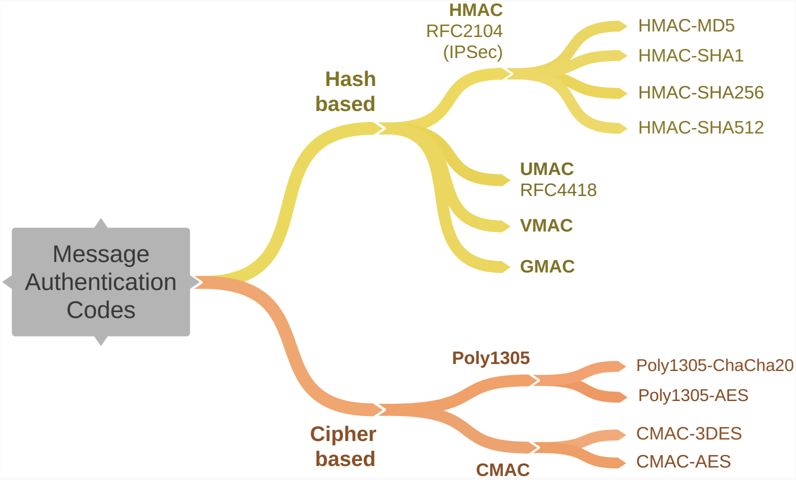
k_1 and k_2 are derivated from k



Block-cipher based MAC algorithms

Poly1305

- created by Daniel J. Bernstein
- the algorithm name is based on the fact that it evaluates the *modulus* of the prime number $2^{130} - 5 = 1\,361\,129\,467\,683\,753\,853\,853\,498\,429\,727\,072\,845\,819$ (40 decimal digits)
- there are several variants:
  - Poly1305-AES
  - Poly1305-Salsa20
  - Poly1305-ChaCha20 – adopted by google for TLS communications (standardized in RFC 7905), can also be used in SSH


Message Authentication Codes
Hash based
HMAC
RFC2104
(IPSec)
HMAC-MD5
HMAC-SHA1
HMAC-SHA256
HMAC-SHA512
UMAC
RFC4418
VMAC
GMAC
Cipher based
Poly1305
Poly1305-ChaCha20
Poly1305-AES
CMAC
CMAC-3DES
CMAC-AES

Authenticated Encryption

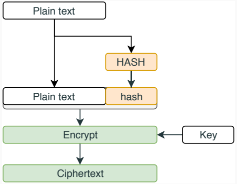
Authenticated encryption (AE)

- is a term used to describe encryption systems that simultaneously protect confidentiality and authenticity (integrity)
- until recently the two services have been designed separately
  - Hashing followed by encryption
  - Independently encrypt and authenticate
  - Authentication followed by encryption
  - Encryption followed by authentication

Traditional approaches

Hash-then-Encrypt

$E_k[M \mid\mid H(M)]$

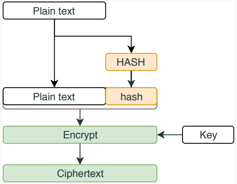


(used on WiFi WEP)

Traditional approaches

Hash-then-Encrypt

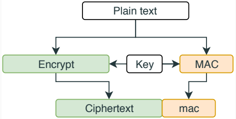
$E_k[M \parallel H(M)]$



(used on WiFi WEP)

Encrypt-and-MAC

$E_k(M) \parallel C_k(M)$

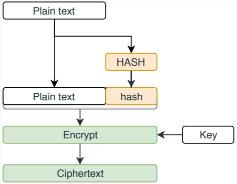


(used on SSH)

Traditional approaches

Hash-then-Encrypt

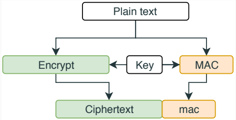
$E_k[M \,||\, H(M)]$



(used on WiFi WEP)

Encrypt-and-MAC

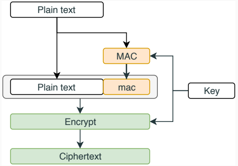
$E_k(M) \,||\, C_k(M)$



(used on SSH)

MAC-then-Encrypt

$E_k[M \,||\, C_k(M)]$

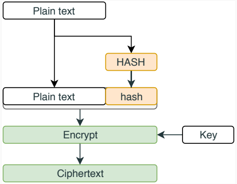


(used on TLS)

Traditional approaches

Hash-then-Encrypt

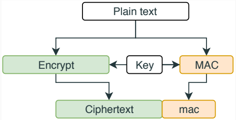
$E_k[M \,||\, H(M)]$



(used on WiFi WEP)

Encrypt-and-MAC

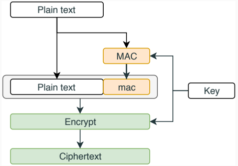
$E_k(M) \,||\, C_k(M)$



(used on SSH)

MAC-then-Encrypt

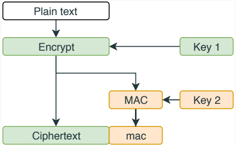
$E_k[M \,||\, C_k(M)]$



(used on TLS)

Encrypt-then-MAC

$E_{k2}(M) \,||\, C_{k1}[E_{k2}(M)]$



(used on IPsec)

Traditional approaches

- these approaches require two passes through the data being protected (time consuming)
- there are security vulnerabilities with all of them
- nevertheless, with proper design, any of these approaches can provide a high level of security

Traditional approaches

- these approaches require two passes through the data being protected (time consuming)
- there are security vulnerabilities with all of them
- nevertheless, with proper design, any of these approaches can provide a high level of security

Authenticated encryption (AE)

- requires only a single pass through the data being protected (more efficient)
- address the security vulnerabilities of the traditional approaches
- it is possible to authenticate data that is not encrypted:
  - **Authenticated Encryption with Associated Data** (AEAD)

Authenticated Encryption with Associated Data (AEAD):

- allows a recipient to check the integrity of both the **encrypted** and **unencrypted** information in a message
- associated data – is a part of the message that does not requires confidentiality, but must be authentic, *e. g.* network protocol headers

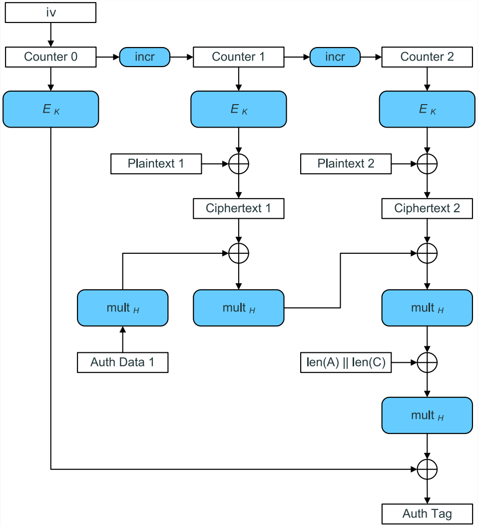
Authenticated Encryption with Associated Data (AEAD):

- allows a recipient to check the integrity of both the **encrypted** and **unencrypted** information in a message
- associated data – is a part of the message that does not requires confidentiality, but must be authentic, *e. g.* network protocol headers

Some Variants:

- **Counter with Cipher Block Chaining-Message** (CCM) – used by IEEE 802.11 WiFi
- **Galois/Counter Mode** (GCM) – mode of operation for symmetric-key cryptographic block ciphers that has been widely adopted because of its performance
  - processes with a single pass over the data
  - supported in the newer versions of IPsec, SSH, TLS 1.2 and TLS 1.3, ...

AUTHENTICATED ENCRYPTION WITH ASSOCIATED DATA – GALOIS/COUNTER MODE (GCM)



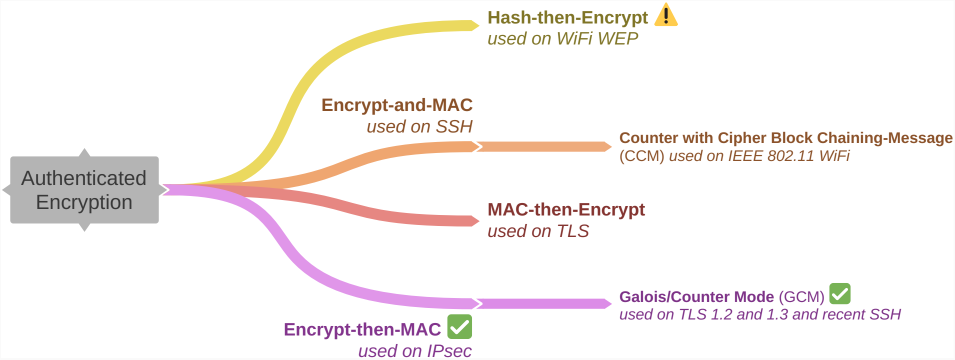
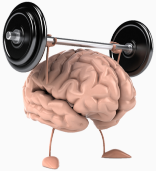

Authenticated Encryption
Hash-then-Encrypt
used on WiFi WEP
Encrypt-and-MAC
used on SSH
Counter with Cipher Block Chaining-Message
(CCM) used on IEEE 802.11 WiFi
MAC-then-Encrypt
used on TLS
Galois/Counter Mode (GCM)
used on TLS 1.2 and 1.3 and recent SSH
Encrypt-then-MAC
used on IPsec

Table 1: Limitations for an untruncated 128-bit authentication tag

Construction	**Max bytes for a single (key,nonce)**	**Max bytes for a single key**
AES256-GCM [1]	64 GB [2]	$\approx$ 350 GB (for $\approx$16 KB long messages)
ChaCha20-Poly1305 [3]	2^{64} bytes (no pratical limit)	Up to 2^{64} messages (no pratical limit)
ChaCha20-Poly1305-IETF [3]	256 GB [2]	Up to 2^{64} messages (no pratical limit)
XChaCha20-Poly1305-IETF [3,4]	2^{64} bytes (no pratical limit)	Up to 2^{64} messages (no pratical limit)

[1] fastest algorithm due to hardware acceleration introduced by Intel in the Westmere processors (in 2010) and newer
[2] it is possible to overcome the limitation by rekeying: using a new (key,nounce) pair
[3] faster than AES in software only implementations (without AES specific hardware acceleration)
[4] is the safest choice (key = 256 bits, block = 512 bits, nounce = 192 bits), but not wide spread usage

Exercises



1. What types of attacks are addressed by message authentication?
2. What are some approaches to producing message authentication?
3. When a combination of symmetric encryption and an error control code is used for message authentication, in what order must the two functions be performed?
4. What is a message authentication code?
5. What is the difference between a message authentication code and a one-way hash function?

Cryptographic Schemes

Use one, or more, cryptographic algorithm, to achieve new goals. There are four types of cryptographic schemes with hash:

1. Message Authentication Code (MAC) – already addressed
2. Authenticated Encryption – already addressed
3. Key Derivation Functions (KDF) – to store passwords in a non-reversible fashion
4. One-Time Password (OTP) – mainly used as second authentication factor

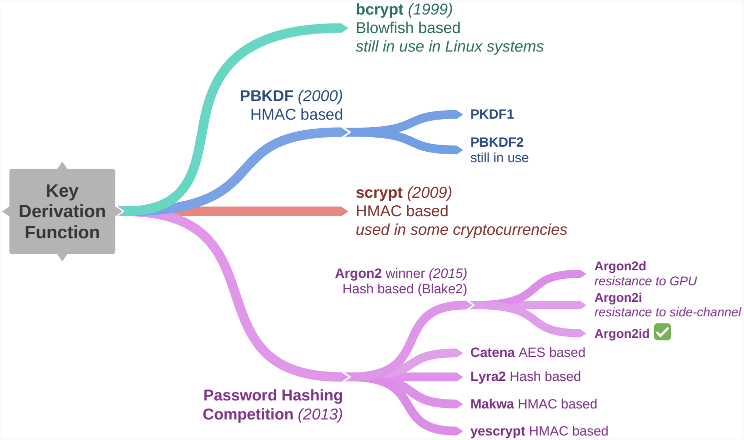
Key Derivation Functions (KDF)

- designed to securely store passwords without encryption
- like an hash function:
  - the output has fixed length (256, 512, ...) bits
  - non-reversible
- slower than hash functions → harder to brute-force
  - can be parameterized to adjust the required computational effort
  - common parameters are: salt, number of iterations, derived key length

Key Derivation Functions (KDF)

- designed to securely store passwords without encryption
- like an hash function:
  - the output has fixed length (256, 512, ...) bits
  - non-reversible
- slower than hash functions $\rightarrow$ harder to brute-force
  - can be parameterized to adjust the required computational effort
  - common parameters are: salt, number of iterations, derived key length
- Disadvantages:
  - it is still needed to keep a secret value on the server side
  - people tend to use weak passwords and reuse them (this is a problem with passwords in general)

Key Derivation Functions



Key Derivation Function

Argon2

- based on Blake2b hash
- three versions
    - Argon2d – is faster, suitable for cryptocurrencies, is resistant to GPU accelaration;
    - Argon2i – is slower, preferred for password hashing, is resistant to side-channel attacks;
    - Argon2id – is a hybrid version;

Key Derivation Function

Argon2

- based on Blake2b hash
- three versions
  - Argon2d – is faster, suitable for cryptocurrencies, is resistant to GPU accelaration;
  - Argon2i – is slower, preferred for password hashing, is resistant to side-channel attacks;
  - Argon2id – is a hybrid version;

Parameters:

- primary:
  - **password** – any length from 0 to $2^{32} - 1$ bytes;
  - **number of iterations** – used to tune the running time independently of the memory size;
  - **memory required** – it is a memory-hard function;
  - **parallelism** – how many independent computational chains can be run;
- secondary (default values maybe used):
  - salt – random value, 128 bits is recommended for password hashing;
  - output length – any integer number of bytes from 4 to $2^{32} - 1$
  - secret value – serves as key if necessary, but by default none is used;
  - associated data – optional arbitrary extra data;
  - algorithm type – *d*, *i*, or *id*;

Key Derivation Function

Argon2 example:

- input values:
    - password = `Correct Horse Battery Staple`;
    - memory = `1 GB`
    - iterations = `4`
    - parallelism = `1`
- output:

  `$argon2id$v=19$m=1048576,t=4,p=1$1k8ic3+vS0kDB+4J7bcjQg$ok3jcq9yW7Rw7GuhZfJi0IdcoixCZYWgCcHxp8bny6M`
    - `$argon2id` – the variant of Argon2 being used
    - `$v=19` – the version of Argon2 being used
    - `$m=1048576,t=4,p=1` – the memory (m), iterations (t) and parallelism (p)
    - `$1k8ic3+vS0kDB+4J7bcjQg` – the base64-encoded randomly selected salt
    - `$ok3jcq9yW7Rw7GuhZfJi0IdcoixCZYWgCcHxp8bny6M` – the base64-encoded derived key

Key Derivation Function

Argon2 example:

- input values:
  - password = `Correct Horse Battery Staple`;
  - memory = `1 GB`
  - iterations = `4`
  - parallelism = `1`
- output:
  `$argon2id$v=19$m=1048576,t=4,p=1$1k8ic3+vS0kDB+4J7bcjQg$ok3jcq9yW7Rw7GuhZfJi0IdcoixCZYWgCcHxp8bny6M`
  - `$argon2id` – the variant of Argon2 being used
  - `$v=19` – the version of Argon2 being used
  - `$m=1048576,t=4,p=1` – the memory (m), iterations (t) and parallelism (p)
  - `$1k8ic3+vS0kDB+4J7bcjQg` – the base64-encoded randomly selected salt
  - `$ok3jcq9yW7Rw7GuhZfJi0IdcoixCZYWgCcHxp8bny6M` – the base64-encoded derived key
- time to calculate on my laptop ≈ 2,5 seconds

Key Derivation Function

Argon2 guidelines for choosing the parameters:

1. set **memory** limit to the amount of memory you want to reserve for password hashing
2. then, set **iterations** to 3 and measure the time it takes to hash a password
3. if it is too long for your application, reduce **memory**, but keep **iterations** set to 3

Key Derivation Function

Argon2 guidelines for choosing the parameters:

1. set **memory** limit to the amount of memory you want to reserve for password hashing
2. then, set **iterations** to 3 and measure the time it takes to hash a password
3. if it is too long for your application, reduce **memory**, but keep **iterations** set to 3

Use cases:

- for online use (*e. g.* login in on a website), a **1 second** computation is likely to be the acceptable maximum
- for interactive use (*e. g.* a desktop application), a **5 second** pause after having entered a password is acceptable if the password doesn't need to be entered more than once per session
- for non-interactive use and infrequent use (*e. g.* restoring an encrypted backup), an even slower computation can be an option

Key Derivation Function

Argon2 guidelines for choosing the parameters:

1. set **memory** limit to the amount of memory you want to reserve for password hashing
2. then, set **iterations** to 3 and measure the time it takes to hash a password
3. if it is too long for your application, reduce **memory**, but keep **iterations** set to 3

Use cases:

- for online use (*e. g.* login in on a website), a **1 second** computation is likely to be the acceptable maximum
- for interactive use (*e. g.* a desktop application), a **5 second** pause after having entered a password is acceptable if the password doesn't need to be entered more than once per session
- for non-interactive use and infrequent use (*e. g.* restoring an encrypted backup), an even slower computation can be an option

Note

But the best defense against brute-force password cracking remains using **strong passwords**

One-Time Password (OTP)

- also known as **one-time pin** or **dynamic password**
- is a password that is valid for **only one** login session or transaction

One-Time Password (OTP)

- also known as **one-time pin** or **dynamic password**
- is a password that is valid for **only one** login session or transaction
- common as a two-factor authentication → requires access to something a person has
    - such as a small device with the OTP calculator built into it, or a smartcard, or specific cellphone, or software application
- main advantage: not vulnerable to **replay attacks**
- main disadvantage: vulnerable to man-in-the-middle attacks

One-Time Password (OTP)

- also known as **one-time pin** or **dynamic password**
- is a password that is valid for **only one** login session or transaction
- common as a two-factor authentication $\rightarrow$ requires access to something a person has
  - such as a small device with the OTP calculator built into it, or a smartcard, or specific cellphone, or software application
- main advantage: not vulnerable to **replay attacks**
- main disadvantage: vulnerable to man-in-the-middle attacks
- there are two types:
  - based only on a seed (or counter)
  - based on time-synchronization

HMAC-based One-time Password (HOTP)

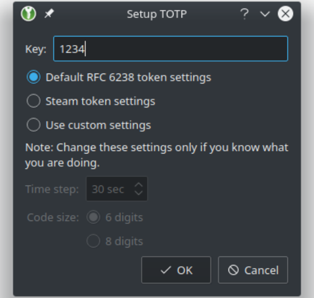
- based on an increasing counter (C) value and a static symmetric key (k)
- specified in RFC4226
  - uses HMAC-SHA-1
  - $HOTP(k, C) = Truncate(HMAC\_SHA1(k, C))$
  - $Truncate()$ returns a number with a configurable amount of digits from 6 to 10
- used on SMS tokens sent by banks
  - vulnerable to SIM swap scam and fake cell phone towers

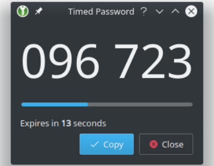
Time-based One-Time Password (TOTP) RFC 6238

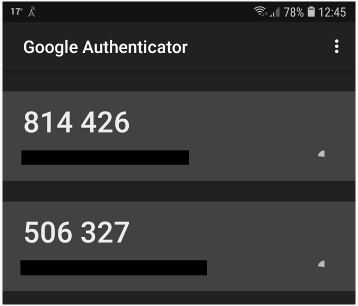
- is an extension of the HMAC-based One-time Password algorithm
- $TOTP = HTOP(k, T)$, where $T = (Current\_Unix\_time - T0)/X$ and X is the time step
- must validate over a range of time (typically 1 minute) due to latency, both network and human, and unsynchronised clocks

Time-based One-Time Password (TOTP) RFC 6238

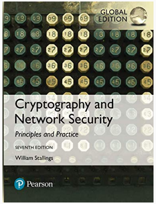
- is an extension of the HMAC-based One-time Password algorithm
- $TOTP = HTOP(k, T)$, where $T = (Current\_Unix\_time - T0)/X$ and X is the time step
- must validate over a range of time (typically 1 minute) due to latency, both network and human, and unsynchronised clocks
- TOPT apps: Keepass, Google Authenticator, LastPass Authenticator, ...







Questions?



Chapters 12 of

William Stallings, Cryptography and Network Security: Principles and Practice, Global Edition, 2016