Integrity and Authentication

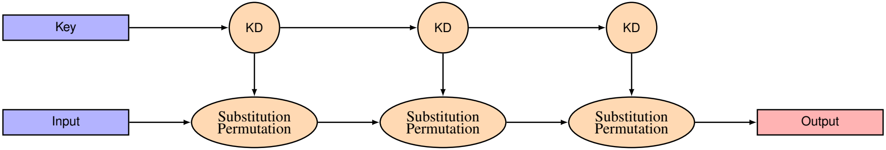
Vincent Migliore

vincent.migliore@insa-toulouse.fr

INSA-TOULOUSE / LAAS-CNRS

Summary of previous lesson

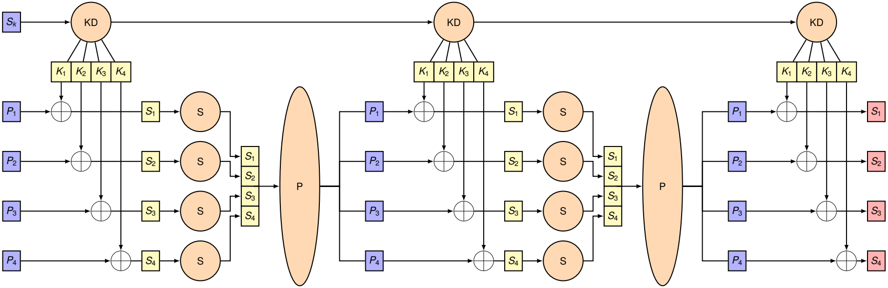
SP-Network



Construction of pseudo-random permutation

- Execution of several rounds parametrized by key.
- In practice, key is pseudo-random and permutation is fixed.
- The more round are executed (with a sufficently large key), the more output is uniform and decorrelated from message.

SP-Network - in details



S-BOX:

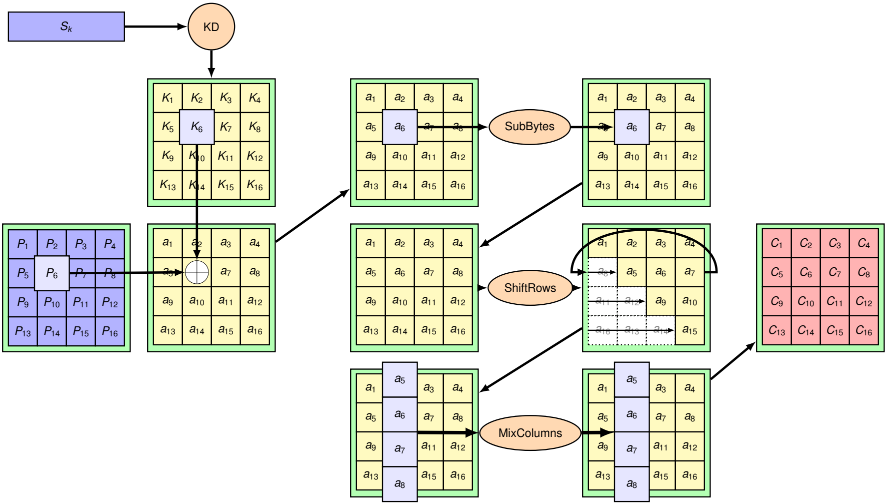
- → Substitutes symbol to another.
- → Non-linear.
- → Provides confusion.
- → Complexify differential cryptanalysis.
- → Does not prevent frequency analysis.

P-BOX (or D-BOX):

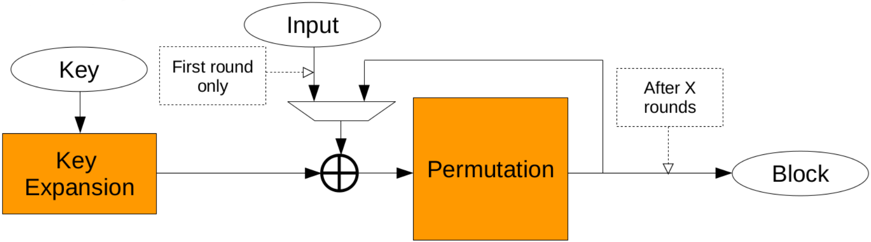
- → Mix symbols of the entire state.
- → Linear.
- → Provides diffusion.
- → Complexify frequency analysis.

Symmetric encryption - Round of AES

Description of 1 round of AES:



Symmetric encryption - case of AES (Rijndael - 2000)



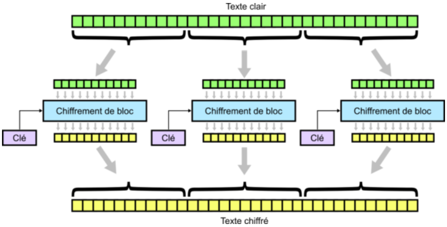
Security

- AES is considered as a good PRP if implemented properly.
- Security depends on the number of rounds executed:

Name	Key length (bits)	Security	rounds
AES-128	128	128	10
AES-196	196	192	12
AES-256	256	256	14

Encryption of larger messages - Mode of operation

Electronic Code Book (ECB)



Construction

The message is split into blocks matching the size of Block-Cipher's block length. Each block is encrypted with the same key.
Pros:

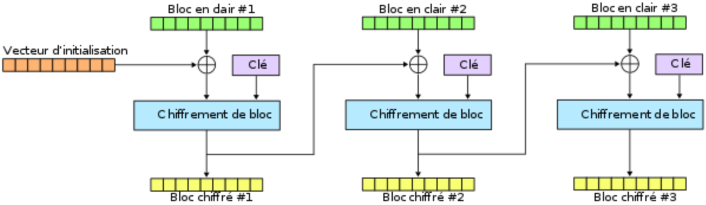
- Simplest construction.
- Destination can decrypt a specific block without extra computations.

Cons:

- Obviously insecure.

Encryption of larger messages - Mode of operation

Cipher Block Chaining (CBC)



Construction

Initialization Vector (IV = nonce) is XORed with input massage block. Then encrypted block is XORed with next input message block.
Pros:

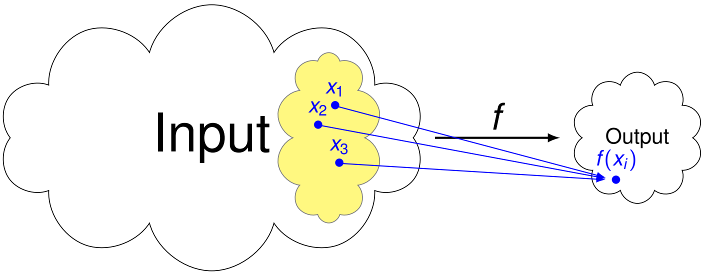
- IND-CPA Secure if IV is random, uniform and unpredictible.

Cons:

- No paralellization.
- decryption of block N requires decryption of all previous blocks.

Properties of integrity check code

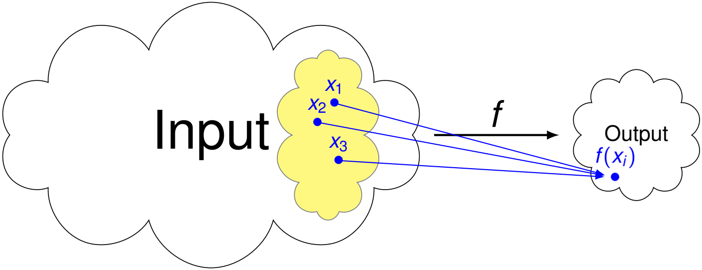
Integrity check code



Desirable properties

- **Small code:** Integrity check code must be very small compared to message;
- **Robustness against bitflips:** A small change on the message greatly change the code (avalanche effect);
- **Impossible forgeability:** Impossible to find pre-image from a given code, impossible to find another messsage with same code, ...

Integrity check code → Hash function



Usual properties of cryptographic Hash functions

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

Among all properties above, which one leads to most devastating attacks?

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question
Among all properties above, which one leads to most devastating attacks?

Answer
First pre-image, since we can exploit any integrity check code.

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

We found an efficient algorithm A that find first pre-image. Does this mean that finding second pre-image is simple?

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

We found an efficient algorithm A that find first pre-image. Does this mean that finding second pre-image is simple?

Answer

Yes:

1. Compute $H(m_1)$.
2. Run algorithm A to find pre-image m_2.
3. Done.

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

We found an efficient algorithm A_2 that find second pre-image. Does this mean that finding a collision is simple?

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

We found an efficient algorithm A_2 that find second pre-image. Does this mean that finding a collision is simple?

Answer

Yes:

1. Choose m_1.
2. Run algorithm A_2 to find m_2.
3. Done.

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Conclusion

First pre-image attack $\implies$ Second pre-image attack $\implies$ Collision attack.
The opposite is not true in general.

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

You are communicating with a server that uses Hash function with first pre-image resistance but not second pre-image resistance. Do you trust the server?

Integrity check code → Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

You are communicating with a server that uses Hash function with first pre-image resistance but not second pre-image resistance. Do you trust the server?

Answer

Obviously not. You have no evidence that message downloaded is the good one since server can find another file with same hash.

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

Same scenario at except that server is now trusty. Do you trust the file?

Integrity check code $\rightarrow$ Hash function

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

Same scenario at except that server is now trusty. Do you trust the file?

Answer

No once again. Adversary in the middle can find another message m_2 with same hash and switch messages.

Hash function security:
worst case attack = exhaustive search?

Hash functions security

Birthday Attack

Consider a teacher with a class of 30 students asks for everybody's birth day. What is the probability that at least one student has the same birth day than another student?

Answers

- $1 - \left(\frac{364}{365}\right)^{30} = 7.9\%$
- $1 - \frac{365!}{(365-30)! \cdot 365^{30}} = 70\%$

Hash functions security

Birthday Attack

Consider a teacher with a class of 30 students asks for everybody's birth day. What is the probability that at least one student has the same birth day than another student?

Answers

- $1 - \left(\frac{364}{365}\right)^{30} = 7.9\%$
  Probability that at least one student has a given birthday
- $1 - \frac{365!}{(365-30)! \cdot 365^{30}} = 70\%$
  Probability that at least two students has the same birthday

Hash functions security

Why?

- P(at least two people have the same birth day)
  = 1 - P(no one shares the same birth day).
- First student: $365/365$
- Second student: $365/365 - 1/365$ (i.e. we remove the birth day of the first student).
- third student: $365/365 - 2/365$.
- $\ldots$
- 30^{th} student: $(365 - 29)/365$.

Hash functions security

Why?

$$
\begin{aligned}
\frac{365-0}{365} \times \frac{365-1}{365} \cdots \frac{365-n-1}{365} &= \prod \frac{365-i}{365} \\
&= \frac{1 \times 2 \cdots (365-n)}{1 \times 2 \cdots (365-n)} \prod \frac{(365-i)}{365} \\
&= \frac{1 \times 2 \cdots (365-n)}{1 \times 2 \cdots (365-n)} \frac{(365-n-i) \cdots 365}{365^n} \\
&= \frac{365!}{(365-n)!.365^n}
\end{aligned}
$$

Hash functions security

Previous answer

$P = 1 - \frac{365!}{(365-n)!.365^n}$

Question

We consider a hash function $f : \mathbb{Z}_M \to \mathbb{Z}_H$.
How many tries t an attacker should test to expect 50% chance of finding a collision?

Hash functions security

Previous answer

$P = 1 - \frac{365!}{(365-n)!.365^n}$

Question

We consider a hash function $f : \mathbb{Z}_M \rightarrow \mathbb{Z}_H$.
How many tries t an attacker should test to expect 50% chance of finding a collision?

Answer

$0.5 = 1 - \frac{H!}{(H-t)!.H^t}$

Hash functions security - Some standard relations for birthday attack

Notation

Let $f : \mathbb{Z}_M \to \mathbb{Z}_H$ be a hash function with H possible outputs. We note:

- $p(n; H)$ the probability to find at least one collision after n tries;
- $n(p; H)$ the number of tries before finding a collision with probability p.

Estimation of $p(n; H)$

$p(n; H) = \frac{365!}{(365-n)!.365^n} \approx 1 - e^{-n^2/(2H)}$.
(Birthday attack exact formula + application of stirling formula $\left(n! \sim \sqrt{2\pi n}(\frac{n}{e})^n\right)$ + application of taylor expansion at order 2).

Estimation of $n(p; H)$

$n(p; H) = \sqrt{2H \ln \frac{1}{1-p}}$

Hash functions security - Some standard relations for birthday attack

Notation

Let $f : \mathbb{Z}_M \to \mathbb{Z}_H$ be a hash function with H possible outputs. We note:

- $p(n; H)$ the probability to find at least one collision after n tries;
- $n(p; H)$ the number of tries before finding a collision with probability p.

Estimation of $p(n; H)$

$p(n; H) = \frac{365!}{(365-n)!.365^n} \approx 1 - e^{-n^2/(2H)}$.
(Birthday attack exact formula + application of stirling formula $\left(n! \sim \sqrt{2\pi n}(\frac{n}{e})^n\right)$ + application of taylor expansion at order 2).

Estimation of $n(p; H)$

$n(p; H) = \sqrt{2H \ln \frac{1}{1-p}}$

Hash functions security - Some standard relations for birthday attack

Notation

Let $f : \mathbb{Z}_M \to \mathbb{Z}_H$ be a hash function with H possible outputs. We note:

- $p(n; H)$ the probability to find at least one collision after n tries;
- $n(p; H)$ the number of tries before finding a collision with probability p.

Estimation of $p(n; H)$

$p(n; H) = \frac{365!}{(365-n)!.365^n} \approx 1 - e^{-n^2/(2H)}$.
(Birthday attack exact formula + application of stirling formula $\left(n! \sim \sqrt{2\pi n}(\frac{n}{e})^n\right)$ + application of taylor expansion at order 2).

Estimation of $n(p; H)$

$n(p; H) = \sqrt{2H \ln \frac{1}{1-p}}$

Hash functions security - Some standard relations for birthday attack

Question

Simplify equation $n(p; H) = \sqrt{2H \ln \frac{1}{1-p}}$ considering $p = 0.5$ and $H = 2^L$

Hash functions security - Some standard relations for birthday attack

Question

Simplify equation $n(p; H) = \sqrt{2H \ln \frac{1}{1-p}}$ considering $p = 0.5$ and $H = 2^L$

Answer

$n(0.5; 2^L) = 2^{L/2} \times 1.1774$

Hash functions security - Numerical application

size of (H)	H	$n(p; H) = 50\%$
16 bits	65 536	300
32 bits	4.3×10^{9}	77 000
64 bits	1.8×10^{19}	5.1×10^{9}
128 bits	3.4×10^{38}	2.2×10^{19}
256 bits	1.2×10^{77}	4.0×10^{38}
512 bits	1.3×10^{154}	8.0×10^{76}

Remark

The birthday attack is the worst case attack. It can be combined with another algorithm to reduce complexity to make a collision.

Hash functions security

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

Question

The birthday attack can be applyied to:

First pre-image attack.

Second pre-image attack.

Collision attack.

Hash functions security

- First Pre-image resistant:
  Knowing $h_{m_1} = H(m_1)$, finding m_2 such as $H(m_2) = h_{m_1}$ is hard
- Second Pre-image resistant (Weak collision resistance):
  For a given m_1, finding m_2 such as $H(m_1) = H(m_2)$ is hard
- Collision-resistant (Strong collision resistance):
  finding m_1 and m_2 such as $H(m_1) = H(m_2)$ is hard

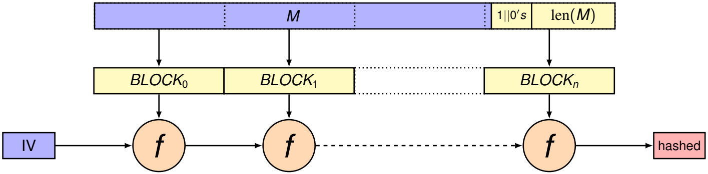
Question

The birthday attack can be applyied to:

- ✗ First pre-image attack.
- ✗ Second pre-image attack.
- ✓ Collision attack.

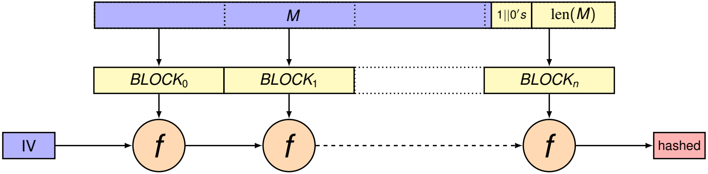
Construction of Hash function

Construction of a hash function - Merkle-Damgård (MD5, SHA-1,SHA-2,...)



- *f* is a compression function: produces an output strictly smaller than input (input and output have fixed size);
- Input message is padded: making length of padded message be a multiple of f input length;
- Merkle-Damgård *strenghtening*: Size of message is appended at the end of padded message. It makes collision security of hash function only relying on collision security of f.

Construction of a hash function - Merkle-Damgård, case of MD5



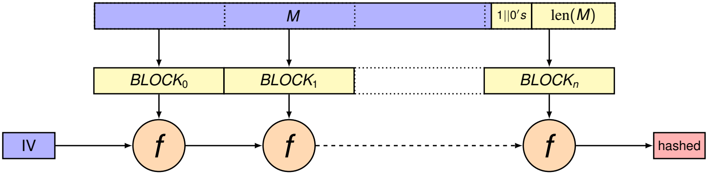
Configuration

Message is split into blocks of 64 bytes. f produces 128 bits IVs.

Limitations

- Birthday attack: $2^{64} < 2^{80} \implies$ not considered secured for modern cryptography.
- Vulnerability to Chosen prefix collision attack (Steven's et al. 2009): $\forall(m_1, m_2)$, at most 2^{39} calls are required to find (s_1, s_2) such as $\mathrm{MD5}(m_1||s_1) = \mathrm{MD5}(m_2||s_2)$. Has been successfully used to forge a fake server certificate from legal authority.

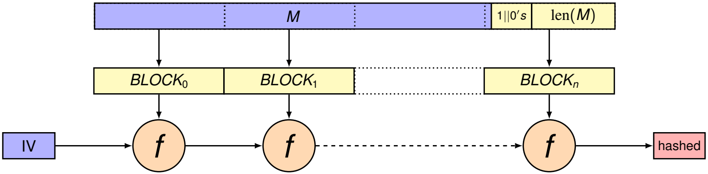
Construction of a hash function - Merkle-Damgård, case of MD5



Other limitation

- MD5 computation is fast:
  A GPU can compute about 150 million hashes per second (Yanjun et al., 2014).
- Chosen Prefix Attack:
  150 millions $\sim 2^{27} \implies 2^{39}/2^{27} = 2^{12}\, sec = 1h\,8m$.

Construction of a hash function - Merkle-Damgård, case of SHA



Secure Hash Algorithm (SHA) familly

- SHA-1:
  Collision in 2^{60} calls (slightly better than MD5), but not secure from modern cryptography point of view. (160 bit output)
- SHA-2:
  - Different output sizes (SHA-224, SHA-256, SHA-384, SHA-512,...);
  - No known vulnerability, just avoid implementations with 31/64 rounds.
- SHA-3:
  Alternative to SHA-2 (not a replacement). More flexibility (can be used to cover several cryptographic algorithms).

Hash function alone: secure?

Integrity with Authentication - AEAD (Authenticated Encryption with Associated Data)

Limitation of Hash functions in practice

Consider a user which downloads a program from a legitimate server. What an attacker can do if it intercepts communication?

Answer

It can replace program to malicious one, computes its hashes, and send malicious program+hashes to user.

Counter-measure?

If user and server share a secret (unknown to attacker), they can use construction called MACs (similar to hash functions) to authenticate message.

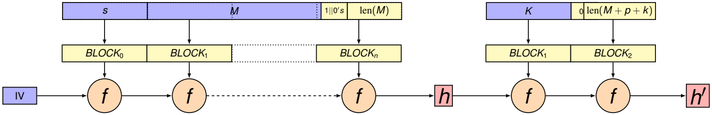
First tentative - *secret-prefix*

Definition

For message m and secret value s, $\mathrm{MAC} = \mathrm{H}(s||m)$.

Why it is bad?

Because Merkle-Damgård based hash functions are vulnerable to extension attack.



Principle

We note p the padding block of message $s||m$.
With pair $(m,\mathrm{MAC} = \mathrm{H}(s||m))$, an attacker can forge (m',h'), where $m' = M||p||K$, and h' obtained by hashing K with $IV = h$.

Second tentative - *secret-suffix*

Definition

For message m and secret value s, $\mathrm{MAC} = \mathrm{H}(m||s)$.

Some architectural weaknesses remain

- Vulnerability on offline second-preimage attack (strong collision):
  (i.e. For given m_1, finding m_2 such as $\mathrm{H}(m_1) = \mathrm{H}(m_2)$). An attacker can search second pre-image offline (i.e. without information on the secret s) and find m_2. Then, attacker can substitute m_1 by m_2.
- Vulnerability on offline collision attack (weak collision):
  (i.e. Find m_1 and m_2 such as $\mathrm{H}(m_1) = \mathrm{H}(m_2)$). If an attacker can ask an authority to compute a MAC, then he asks a MAC for m_1 and an substitute this for m_2.

More robust construction - HMAC

Definition

$\mathrm{HMAC}(S_k, m) = \mathrm{H}((S_k \oplus opad) || \mathrm{H}((S_k \oplus ipad) || m))$

Property - Relaxing strenghtening against collisions

Fundamental property of HMAC is that compression function may not be collision resistant (only PRF is required) if used as intended $\implies$ MD5 and SHA-1 can be used for HMACs.

More robust construction - HMAC

Definition

$\mathrm{HMAC}(S_k, m) = \mathrm{H}((S_k \oplus opad)||\mathrm{H}((S_k \oplus ipad)||m))$

Property - Relaxing strenghtening against collisions

Fundamental property of HMAC is that compression function may not be collision resistant (only PRF is required) if used as intended $\implies$ MD5 and SHA-1 can be used for HMACs.

More robust construction - HMAC

Definition

$\mathrm{HMAC}(S_k, m) = \mathrm{H}((S_k \oplus opad)||\mathrm{H}((S_k \oplus ipad)||m))$

Property - Relaxing strenghtening against collisions

Fundamental property of HMAC is that compression function may not be collision resistant (only PRF is required) if used as intended $\Longrightarrow$ MD5 and SHA-1 can be used for HMACs.

Weakness in case of malicious server - case of MD5

A server has computed a prefix p such as $p||m_1$ and $p||m_2$ collides (i.e. $\mathrm{MD5}(p||m_1) = \mathrm{MD5}(p||m_2)$). What happens if $S_k = p \oplus ipad$?

More robust construction - HMAC

Definition

$\mathrm{HMAC}(S_k, m) = \mathrm{H}((S_k \oplus opad)||\mathrm{H}((S_k \oplus ipad)||m))$

Property - Relaxing strenghtening against collisions

Fundamental property of HMAC is that compression function may not be collision resistant (only PRF is required) if used as intended $\Longrightarrow$ MD5 and SHA-1 can be used for HMACs.

Weakness in case of malicious server - case of MD5

A server has computed a prefix p such as $p||m_1$ and $p||m_2$ collides (i.e. $\mathrm{MD5}(p||m_1) = \mathrm{MD5}(p||m_2)$). What happens if $S_k = p \oplus ipad$?

We note $h_0 = \mathrm{MD5}(p||m_1) = \mathrm{MD5}(p||m_2)$.

$$\begin{aligned}\mathrm{HMAC}(S_k, m_1) &= \mathrm{MD5}((S_k \oplus opad)||\mathrm{MD5}(p||m_1)) \\ &= \mathrm{MD5}((S_k \oplus opad)||h_0)\end{aligned}$$

$$\begin{aligned}\mathrm{HMAC}(S_k, m_2) &= \mathrm{MD5}((S_k \oplus opad)||\mathrm{MD5}(p||m_2)) \\ &= \mathrm{MD5}((S_k \oplus opad)||h_0)\end{aligned}$$

$\longrightarrow$ collision!

Families of MAC algorithms

Block Cipher-based MACs (CMACs)

CMAC is built with a bloc cipher that operates in CBC mode.
NIST SP800-38B.
It's an improvement of CBC-MAC that had vulnerabilities when messages have variable length. A variant, XCBC-MAC was proposed in 2003(RFC3566, https://tools.ietf.org/html/rfc3566)

HASH function based MACs (HMACs)

HMAC (also called Keyed-hash message authentication code) is built with hash function.

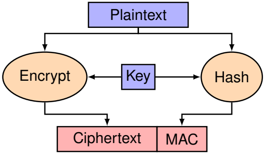
Integrity + Authenticity + Confidentiality

GMC and GMAC mode of operations of bloc ciphers.

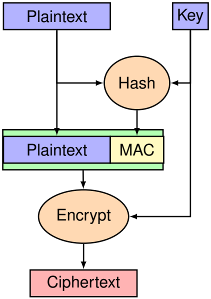
Example of MACs implemented in OpenSSL

CMAC, GMAC, HMAC, KMAC, SipHASH, Poly1305 (Bernstein, selected by google to replace RC4 in TLS/SSL).

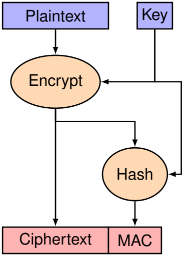
MAC + Encryption: How to?



Encrypt-and-MAC
SSH

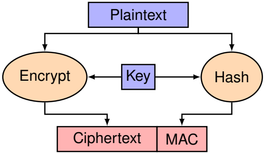


MAC-then-Encrypt
SSL-TLS



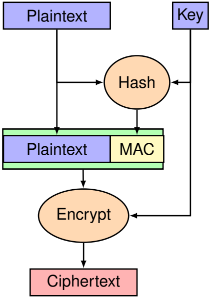
Encrypt-then-MAC
IPSEC

MAC + Encryption: How to?



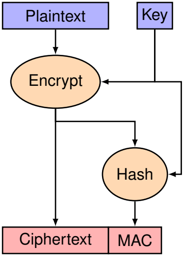
Encrypt-and-MAC

SSH



MAC-then-Encrypt

SSL-TLS



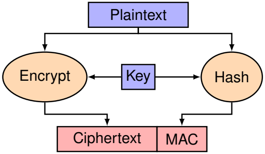
Encrypt-then-MAC

IPSEC

Question

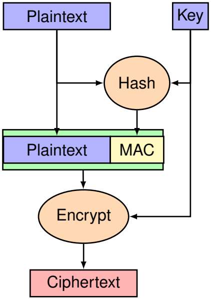
Which of these constructions provides fast output generation?

MAC + Encryption: How to?



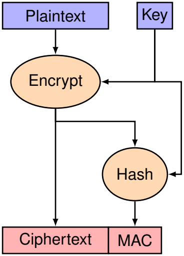
Encrypt-and-MAC

SSH



MAC-then-Encrypt

SSL-TLS



Encrypt-then-MAC

IPSEC

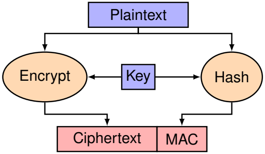
Question

Which of these constructions provides fast output generation?

Answer

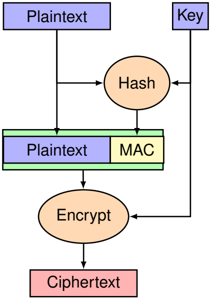
Only Encrypt-and-MAC because Encryption and MAC computation can be parallelized.

MAC + Encryption: How to?



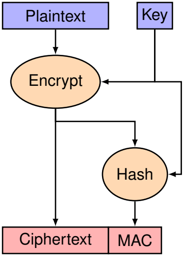
Encrypt-and-MAC

SSH



MAC-then-Encrypt

SSL-TLS



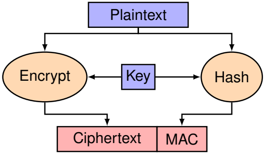
Encrypt-then-MAC

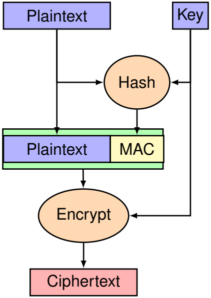
IPSEC

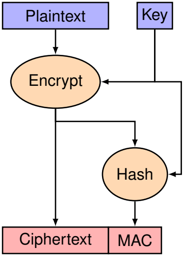
Question

Which of these constructions provides fast verification?

MAC + Encryption: How to?







Encrypt-and-MAC
SSH

MAC-then-Encrypt
SSL-TLS

Encrypt-then-MAC
IPSEC

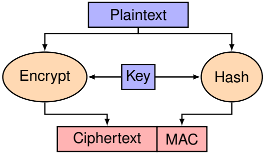
Question

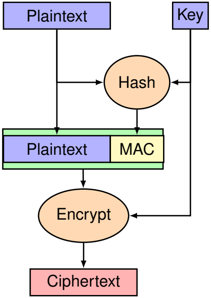
Which of these constructions provides fast verification?

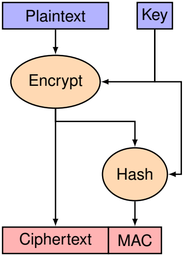
Answer

Only Encrypt-then-MAC because integrity can be verified on the ciphertext. Encrypt-and-MAC and MAC-then-Encrypt needs decryption first.

MAC + Encryption: How to?







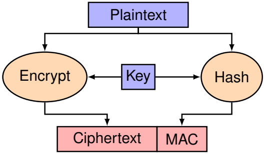
Encrypt-and-MAC
SSH

MAC-then-Encrypt
SSL-TLS

Encrypt-then-MAC
IPSEC

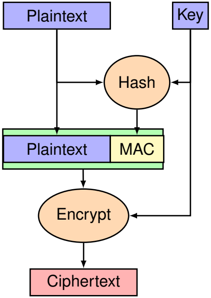
Question
Does one of these constructions is vulnerable to Chosen-Plaintext Attack?

MAC + Encryption: How to?



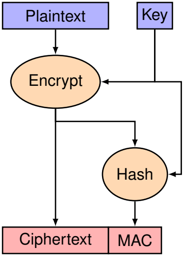
Encrypt-and-MAC

SSH



MAC-then-Encrypt

SSL-TLS



Encrypt-then-MAC

IPSEC

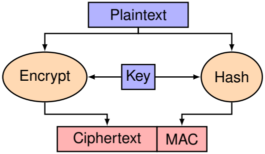
Question

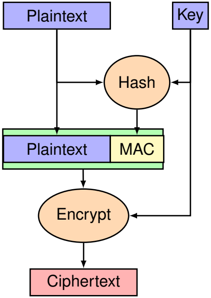
Does one of these constructions is vulnerable to Chosen-Plaintext Attack?

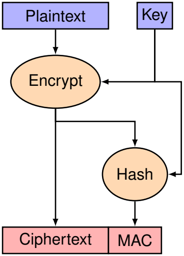
Answer

Encrypt-and-MAC, because MAC only depends on the Plaintext. So, even if Encryption is CPA-secure, Encrypt-and-MAC is not.

MAC + Encryption: How to?







Encrypt-and-MAC — SSH

MAC-then-Encrypt — SSL-TLS

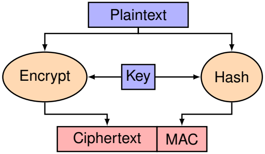
Encrypt-then-MAC — IPSEC

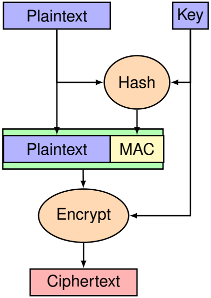
MAC-then-Encrypt security

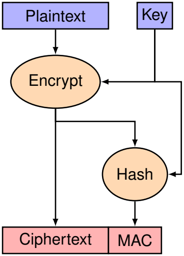
MAC-then-Encrypt is IND-CPA secure, IND-CCA insecure $\Longrightarrow$ Vulnerable for “dynamic” adversary, and protocol specific (BEAST, LUCKY 13, ...).[1]

[1] Bellare and Namprempre, *Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm*, Journal of Cryptology, 2000

MAC + Encryption: How to?







Encrypt-and-MAC
SSH

MAC-then-Encrypt
SSL-TLS

Encrypt-then-MAC
IPSEC

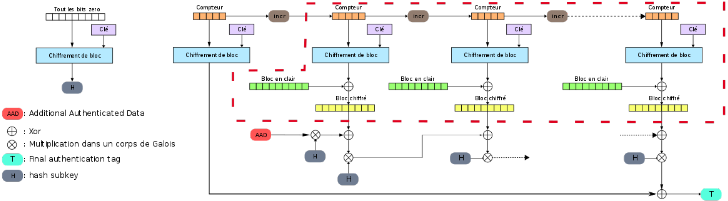
Encrypt-then-MAC security

Encrypt-then-MAC is IND-CPA, IND-CCA, NM-CPA, INT-PTXT, INT-CTXT secure, if Encryption is IND-CPA and MAC strongly unforgeable (i.e. adversary not able de forge a valid MAC on a previously authenticated message).[1]

[1] Bellare and Namprempre, *Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm*, Journal of Cryptology, 2000

Hybrid constructions (implemented in TLS-v1.3)

AES-GCM



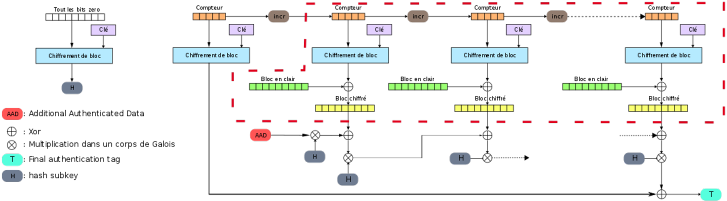
- Encryption is impleted with AES in counter mode to generate a bitstream that is XORed with plaintext.
- MAC is generated by so called "Universal Hashing" using polynomial hashing in a Galois field.
- Efficient: Can be parallelized, pipelinable and support also support variable-length messages.

For more info, see [2]

[2] David A. McGrew and John Viega, *The Security and Performance of the Galois/Counter Mode (GCM) of Operation*, Indocrypt 2004

Hybrid constructions (implemented in TLS-v1.3)

AES-GCM

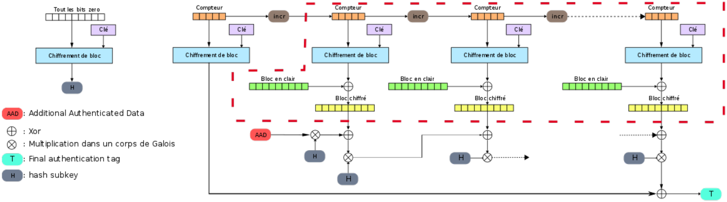


Question

Does AES-GCM follows Encrypt-and-MAC, MAC-then-Encrypt, Encrypt-then-MAC or non of them construction?

Hybrid constructions (implemented in TLS-v1.3)

AES-GCM



Question

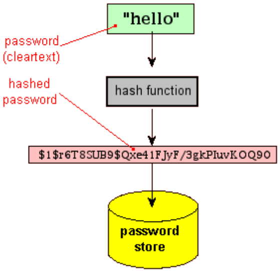
Does AES-GCM follows Encrypt-and-MAC, MAC-then-Encrypt, Encrypt-then-MAC or non of them construction?

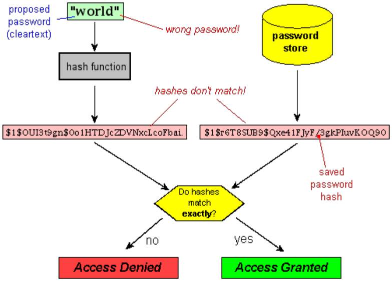
Answer

Encrypt-then-MAC.

Another use of Hash functions: Password checking

Password checking





Password hashed and stored in database

- Because people use in general the same password for several websites, it is critical that password must not be stored in plain.
- Even if stored hashed, if another website uses the same hash function, it can be used "as is" to authenticate to this website.
- In general, passwords are composed in majority a small number of alphanumerical values (so very low entropy), thus finding pre-image generally leads to find the right password.

Password checking - time / space complexity

Use case - SHA1

Hashes has 20 bytes, 8 bytes alphanumerical password (36 values, no uppercase).

No storage

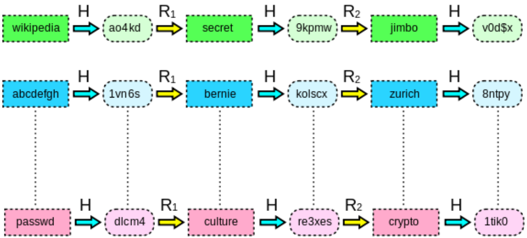
- Number of combinations is $36^8 = 2^{40} \implies$ brute force requires 2^{40} calls to SHA-1 before expecting finding a password.
- Modern 4GHz CPU: 3.5 MHash/sec $= 2^{21}$ Hash/sec $\implies 2^{19}s = 40$ days.

Full storage

Dictionnary of all possible hashes possible:

- Number of combinations: $36^8 = 2^{40}$
- Time complexity: 2^{40} dictionnary entry checking in the worst case, for 4 GHz processor $= 2^{32}$ op/sec. $\implies 2^8$ sec = 4 min.
- Space complexity: 2^{43} bytes of storage (without password storage) $\implies$ 8 Terabytes of data!

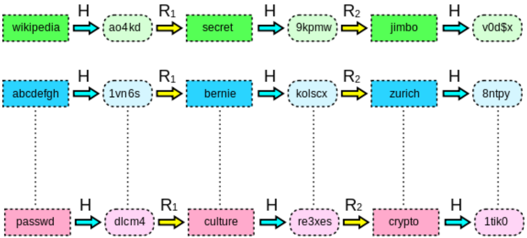
Password checking - Rainbow table



source: wikipedia page

- H: hash function;
- R: reduce function (transform hashes to alphanumerical value);
- We only store left-most and right-most strings.

Password checking - Rainbow table



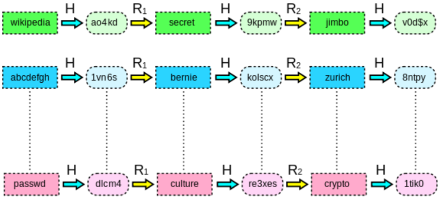
source: wikipedia page

attack

We note h the hashes obtained by attacker after a successfull attack.

- step 1: Check if h is in database. If it is, take the corresponding password p and compute $R_2(H(R_1(H(p)))) \implies$ done.
- step 2: Check if $H(R_2(h))$ is in database. If it is, take the corresponding password p and computes $H(R_1(H(p))) \implies$ done.
- step 3: Check if $H(R_1(H(R_2(h))))$ is in database. If it is, take the corresponding password $p \implies$ done.
- step 4: Fail.

Password checking - Strenghtening



source: wikipedia page

Strenghtening using random salt

- Before storing hashed password p, generate a large random number r and store $\mathrm{H}(r||p)$ and r.
- Rainbow tables are penalized since they are construct with usual characters. Moreover, even if attack succeeds, attack still needs to remove salt.

Strenghtening using slow hash functions

Since attacker must execute hash function many times and legitimate server only one, slowing hash function drastically penalize attacker.

Other constructions

Proof of Work - Hashcash

To avoid spam or denial of service, we force the hashes of sender's message having, say 20 leading bits set to zero (using a customizable header). Also used in bitcoins.

Key derivation

Since hash functions have uniform output, it can be used to make biaised secret to uniform secret.

Next lesson:
Stream cipher