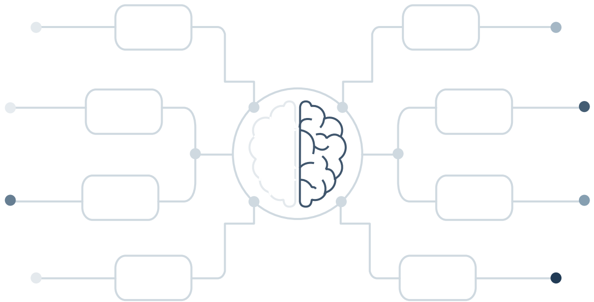
SHA256
HMAC - HKDF



Security & Privacy Information System

Assoc. Prof Ph.D Minh-Triet TRAN
Ph.D Toan-Thinh TRUONG

Hau, NGUYEN PHUC - 20C14003

Sunday, Jan 22, 2022

Agenda

01 SHA-256

02 HMAC-HKDF

03 BUSINESS CASE STUDY

04 REFERENCES

01

SHA-256

To convert plaintext to digest, we need hash algorithms like SHA

SHA History

SHA stands for **Secure Hash Algorithm**:

- **1993 | SHA (SHA-0) (FIPS 180):** Introduced by the National Security Agency
- **1995 | SHA-1 (FIPS 180-1) 160 bits:** Developed by NIST (National Institute of Standards and Technology)
- **2002 | SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512) (FIPS 180-2)**
- 2005: SHA-1 has not been considered secure against well-funded opponents.

SHA-2

Terminologies

1. **Bit:** A binary digit having a value of 0 or 1.
2. **Byte:** A group of 8 bits.
3. **Word:** A group of 32 bits (4 bytes).
4. **Hexadecimal Notation:** Numbering system using a base of 16.
5. **Hex Digit:** Representation of a 4-bit string

Basic terminologies that must be known in order to understand how SHA-256 works (U.S. Department of Commerce 2012)

SHA Properties

Cryptographic hash functions must have the following properties in order to be considered secure (Lantz & Cawrey 2020)

1. **Deterministic:** Produces the same hash for the same input.
2. **Quick to Compute:** Easy to compute the hash value of any given message.
3. **Pre-Image & collision:** infeasible to generate a message from a given hash.
4. **Avalanche Effect:** Impossible to modify a message without changing the hash.
5. **Resistance to Collision:** Impossible to find two different messages with the same hash.

Operations

$SHR^{n}(x) = x \gg n$

$SHR^{1}(1011) = 1011 \gg 1 = 0101$

$SHR^{3}(1011) = 1011 \gg 3 = 0101$

$ROTR^{n}(x)$

$ROTR^{1}(1011) = 1101$

$ROTR^{3}(1011) = 0111$

$z = (x + y) \; mod \; 2^{32}$

$X = 100_{10} = 110\,01100_{2}$

$X = 200_{10} = 1100\,1000_{2}$

SHA Comparison

SHA (Bits) Comparison Tables					
	SHA-1	SHA-224	**SHA-256**	SHA-384	SHA-512
Digest Size (bit)	160	224	**256**	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	**512**	1024	1024
Word Size	32	32	**32**	64	64
Loop	80	64	**64**	80	80

- **Message Size:** Size limit of the input
- **Message Digest Size:** Length of the output

SHA-256

The **Mathematics** behind SHA-256

Message Digest

Alphanumeric: Consists of letters and numbers

Hexadecimal Representation:

- Each character represents 4 bits.
- Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (10 | 01 23 45 67), B (11 | 89 ab cd ef), C (12 | fe dc ba 98), D (13 | 76 54 32 10), E (14 | C3 D2 E1 F0), F (15)

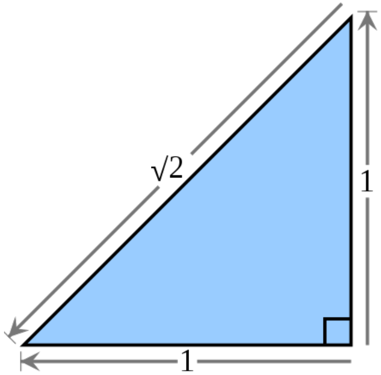
64-Character long

- 256 bits : 4 bytes = 64 characters
  - One character in hex can be represented with 4 bits.

Constant

The first 32 bits of the factional part of square roots of the first 9 primes **2, 3, 5, 7, 11, 13, 17, 19**?

- h0 := 0x6a09e667 **(2)**
- h1 := 0xbb67ae85 **(3)**
- h2 := 0x3c6ef372 **(5)**
- h3 := 0xa54ff53a **(7)**
- h4 := 0x510e527f **(11)**
- h5 := 0x9b05688c **(13)**
- h6 := 0x1f83d9ab **(17)**
- h7 := 0x5be0cd19 **(19)**



Binary: 1.0110 1010 0000 1001 1110…

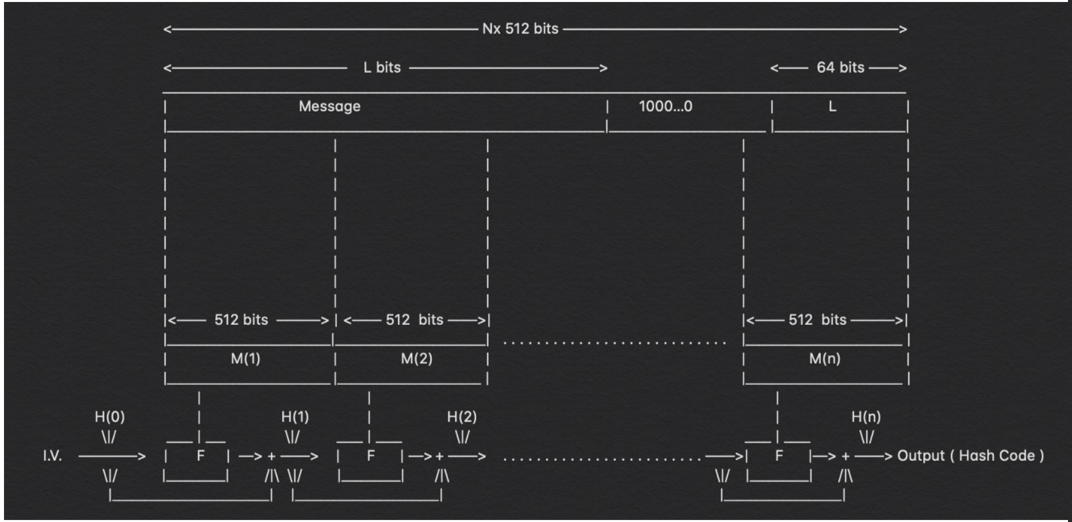
Decimal: 1.41421 35623 73095 0488…

$$1+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\ddots}}}$$

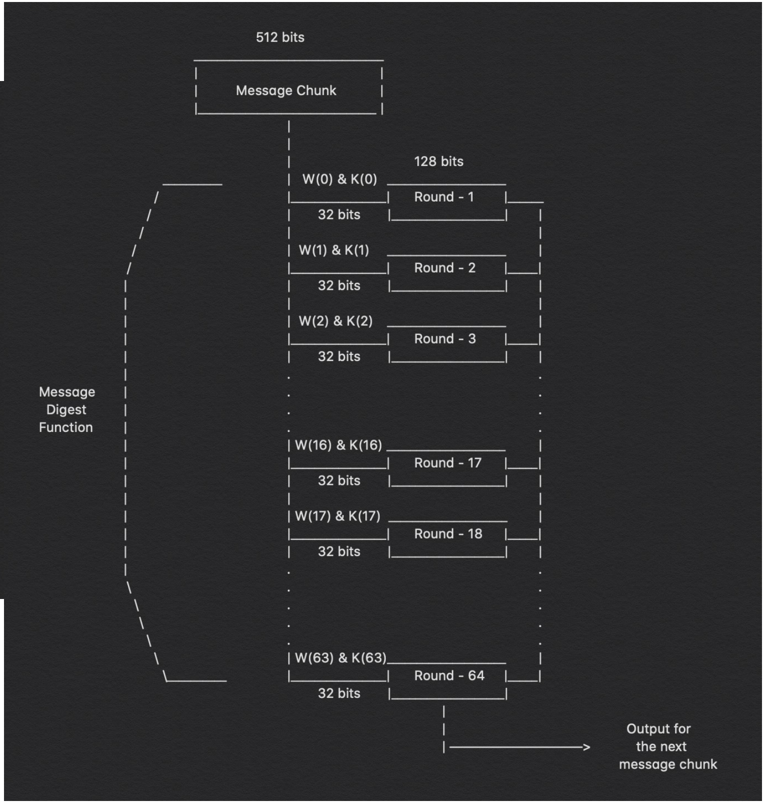
Hexadecimal: 1.6A09 E667 F3BC C908 B2F…

$\sqrt{2}$ = 1.41421 35623 73095 0488 → 0.41421 35623 73095 0488

Round



Nx 512 bits
L bits
64 bits
Message
1000...0
L
512 bits
512 bits
512 bits
M(1)
M(2)
M(n)
H(0)
H(1)
H(2)
H(n)
I.V.
F
F
F
Output (Hash Code)

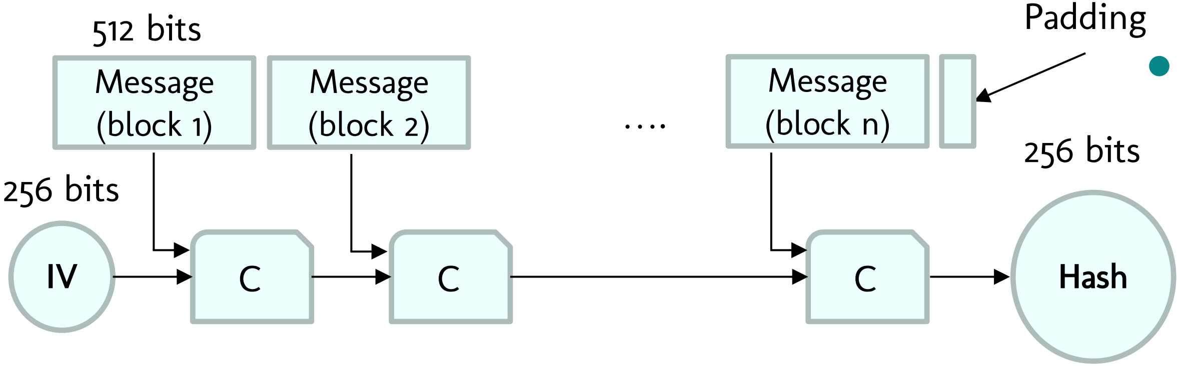


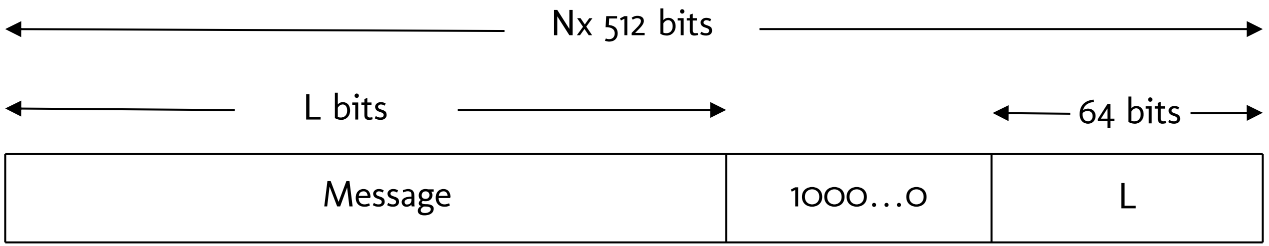
512 bits
Message Chunk
128 bits
W(0) & K(0)
Round - 1
32 bits
W(1) & K(1)
Round - 2
32 bits
W(2) & K(2)
Round - 3
32 bits
Message Digest Function
W(16) & K(16)
Round - 17
32 bits
W(17) & K(17)
Round - 18
32 bits
W(63) & K(63)
Round - 64
32 bits
Output for the next message chunk

Padding bits

M + P + 64 = n x 512

- M = length of original message
- P = padded bits

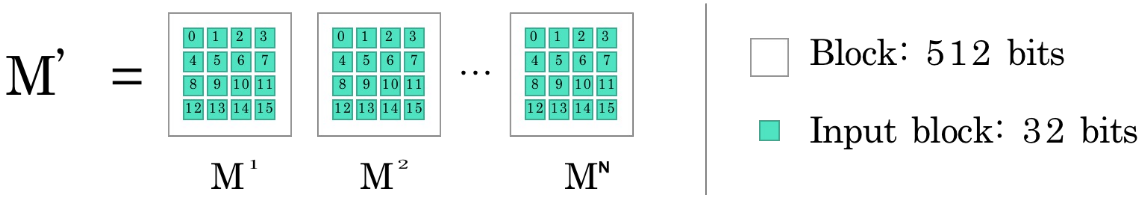




Compress

- Divide the padded binary into 512bit chunks.
- Divide each chucks into 32bit words 16 word per chunk.

SHA-256 will hash these 32bit words 64 times (rounds). Then put them together, become result.

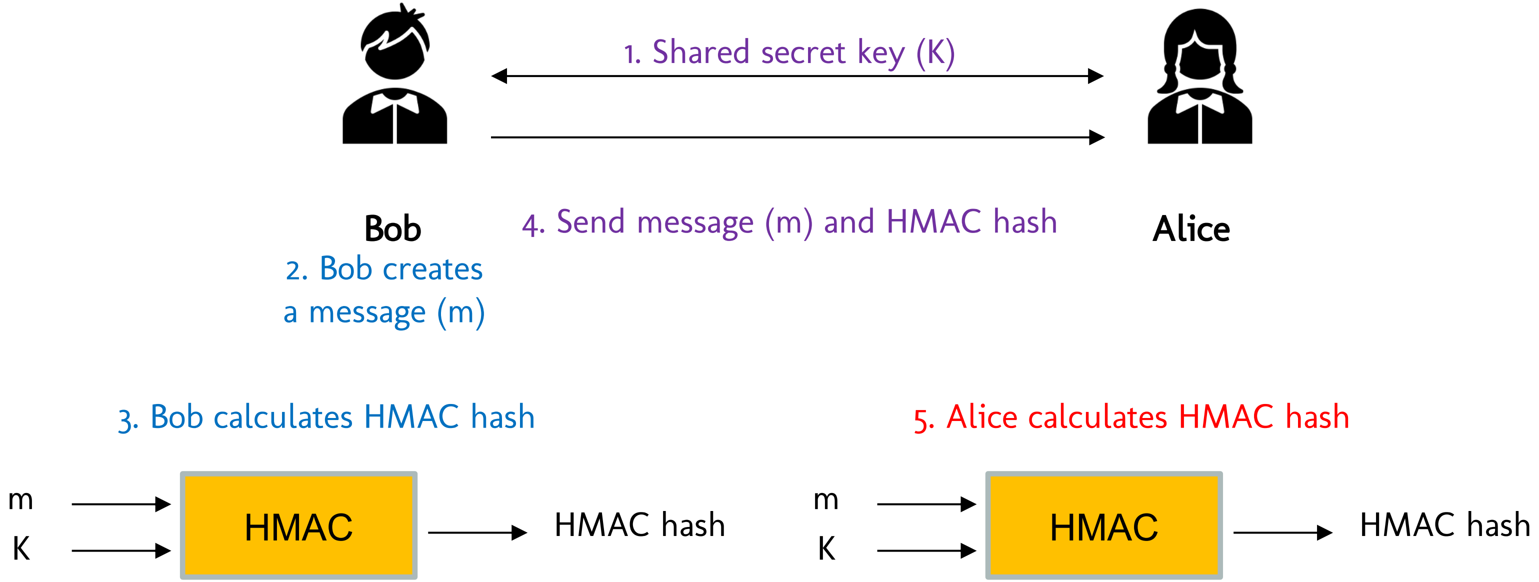


02

HMAC - HKDF

HMAC stands for Hashed-based Message Authentication Code

HMAC



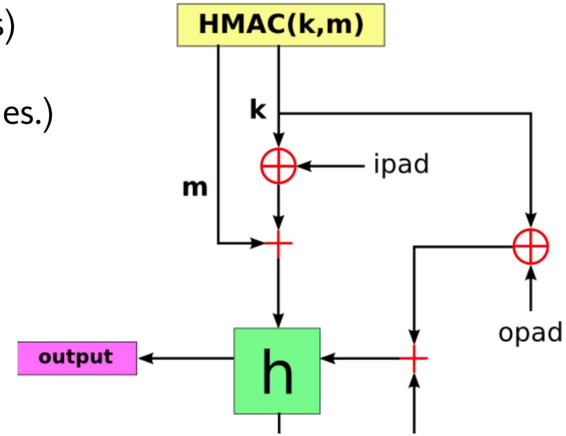
6. Alice verifies the message integrity and authenticity by: received HMAC hash == calculated HMAC hash

HMAC

$$HMAC(K, m) = H((K \oplus opad) \| H((K \oplus ipad) \| m))$$

- Inner_key = $K \oplus$ ipad (ipad is the byte value 0x36 repeated B times)
- Outer_key = $K \oplus$ opad (opad is the byte value 0x5C repeated B times.)
- Inner_hash = H (inner_key $\|$ m)
- HMAC (K, m) = H (outer_key $\|$ inner_hash)

B is the block size in bytes of the underlying hash function



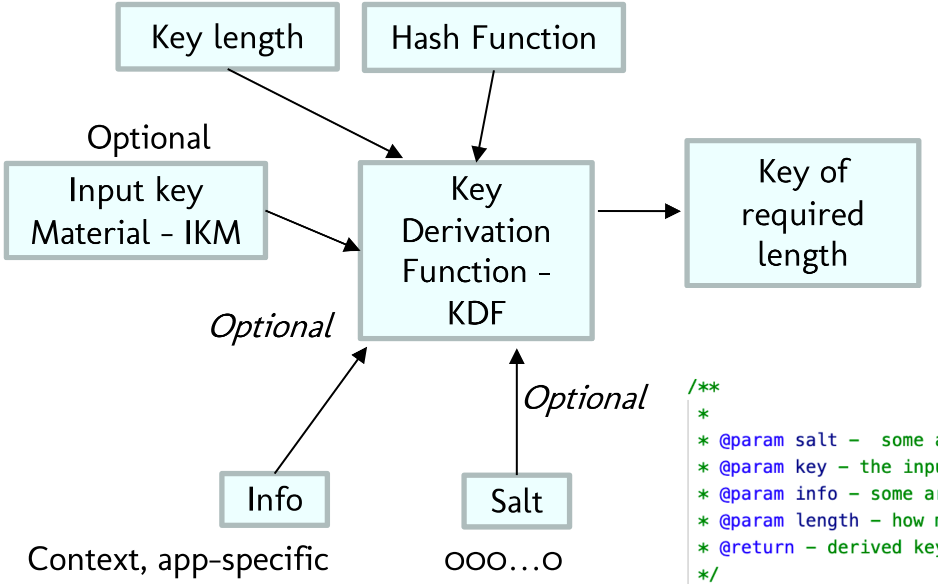
Advantages

- Digital Signatures are larger than HMACs, yet the HMACs provide comparably higher security.
- HMACs are used in administrations where public key systems are prohibited.

Disadvantages

- HMACs uses shared key which lead to non-repudiation.
- If either sender or receiver's key is stolen then it will be easy for attackers to create unauthorized messages.

HKDF (extract-then-expand)




```java
/**
 *
 * @param salt -  some additional randomness (optional)
 * @param key - the input key, from which multiple keys can be derived
 * @param info - some arbitrary string used to bind a derived key to an intended context
 * @param length - how many bytes to derive
 * @return - derived key
 */
public static String hash(String key, int length, String info, String salt) {
    byte[] prk = extractor((salt == null) ? null : Utils.stringToBytes(salt),Utils.stringToBytes(key));
    byte[] okm = expander(prk, Utils.stringToBytes(info), length);
    System.out.println("PRK " + Utils.bytesToHex(prk));
    System.out.println("OKM " + Utils.bytesToHex(okm));
    return Utils.bytesToHex(okm);
}
```

03

BUSINESS CASE STUDY

...

Applications

- Password reset via email (HMAC)
- Account activation Email (HMAC)
- Verify data between client & server
- SSL/TLS Certificate (EdDSA, RSA, HKDF, AES256, CBC,SHA256)
- TLS 1.3 (HKDF)

Company

Bitcoin

PoW using SHA-256

HMAC-SHA256

https://developers.momo.vn/v3/vi/docs/payment/api/other/signature

HMAC-SHA256

https://docs.zalopay.vn/v2/general/overview.html

HKDF-SHA512

Communication between Client and Server

Thanks!

Do you have any questions?

nphausg@gmail.com

nphau.medium.com

REFERENCES

- RFC4634
- RFC5869
- NIST_FIPS_180-4
- NIST_FIPS_198-1
- https://link.springer.com/chapter/10.1007/978-3-540-24654-1_13
- https://www.youtube.com/watch?v=-f4Gbk-U758