05. Hash Functions and Message Authentication Codes

이형태

2019학년도 2학기

Hash Functions

Hash Function and Simple Example

- A hash function is any function that takes any data of arbitrary size as an input and returns an output of fixed size.
- Bitwise XOR

	Bit 1	Bit 2	$\cdots$	Bit n
Block 1	x_{11}	x_{12}	$\cdots$	x_{1n}
Block 2	x_{21}	x_{22}	$\cdots$	x_{2n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Block m	x_{m1}	x_{m2}	$\cdots$	x_{mn}
Hash Output	y_1	y_2	$\cdots$	y_n

Cryptographic Hash Functions

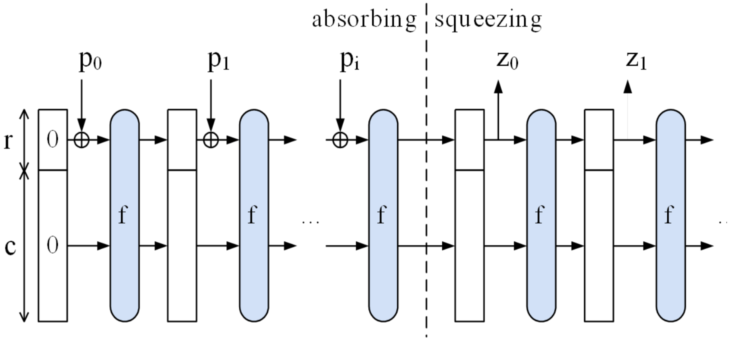
- A cryptographic hash function h is a function that takes a message of arbitrary length as an input and returns a **message digest** of fixed length. It should satisfy the following conditions:
  1. **Efficiently computable**: Given an input m, $h(m)$ should be efficiently computed.
  2. **One-wayness or Pre-image resistant**: Given y, it is computationally infeasible to find an m' such that $h(m') = y$.
  3. **Second pre-image resistant or Weak collision resistant**: Given m, it is computationally infeasible to find an m' such that $h(m') = h(m)$.
  4. **Collision resistant**: It is computationally infeasible to find m and m' such that $h(m) = h(m')$.
- e.g.) MD5, HAVAL-128, SHA-1, SHA-2, SHA-3

History: Secure Hash Algorithm (SHA)

- 1993: Originally developed by NIST
- 1995: Revised as SHA-1: 160-bit output (80-bit security)
- 2002: Included 3 additional versions (SHA-2), SHA-256, SHA-384, SHA-512
  - SHA-2 shares the same structure and mathematical operations as SHA-1
- 2005: Collisions for SHA-1 with 2^{69} operations (Broken!) were found by X. Wang et al.
- 2007: Started SHA-3 competition hosted by NIST
- 2012: Keccak proposed by G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche was finally selected.

Design Rationale of SHA-3

- Based on sponge constructions: Data is absorbed, then the result is squeezed.



- f is a function for block permutation which consists of XOR, AND, and NOT operations.

https://i.stack.imgur.com/JsbvV.png

Instances

Instance	Output Size	Block Size	Capacity	Security		
				Collision	Preimage	2nd Preimage
SHA3-224	224	1152	448	112	224	224
SHA3-256	256	1088	512	128	256	256
SHA3-384	384	832	768	192	384	384
SHA3-512	512	576	1024	256	512	512

Message Authentication Codes

Message Authentication Codes (MAC)

- Motivation: Message integrity and message authentication
  - Alice and Bob want to be assured that any manipulations of a message x in transit are detected.
  - Bob computes the message authentication code (MAC) as a function of the message x and the shared secret key k,

    $$y = \mathrm{MAC}_k(x),$$

    and sends (x, y) to Alice.
  - Alice verifies y using the shared secret key k and the received message x.
- Security requirement: It should be hard to generate a valid output of the function MAC without knowing the secret key k (as digital signatures).

Message Authentication Codes (MAC)

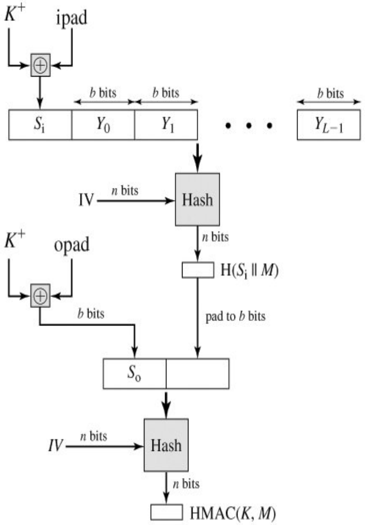
- Properties of MAC
  - Cryptographic checksum
  - Symmetric techniques
  - Arbitrary input length/fixed output length
  - Message integrity
  - Message authentication
  - No nonrepudiation
- Comparison with digital signature
  - Pros: Faster than digital signatures
  - Cons: No nonrepudiation
- Design of MAC
  - Use hash functions
  - Use block ciphers

MACs from Hash Functions: HMAC

- Proposed by M. Bellare, R. Canetti, and H. Krawczyk in 1996
- Exploit cryptographic hash functions as a building block
- Utilized in both IP Security suite, TLS (Transport Layer Security), SET (Secure Electronic Transformation)
- Possible to prove the security of HMAC under certain assumptions

HMAC Construction

1. Append zeros to the left end of K to create b-bit string K^+
2. Compute $S_i = K^+ \oplus \text{ipad}$ where ipad is 0x36 repeated $b/8$ times
3. Append M to S_i
4. Compute $H(S_i \| M)$
5. Compute $S_o = K^+ \oplus \text{ipad}$ where opad is 0x5C repeated $b/8$ times
6. Append $H(S_i \| M)$ to S_o
7. Compute and output $H(S_o \| H(S_i \| M))$



Security of HMAC

- Can provide provable security (i.e., HMAC is secure under certain assumption)
- Generating a valid output of MAC is equivalent to one of the following attacks on the exploited hash function:
  - The attacker can compute an output of the compression function even with an IV that is random, secret, and unknown to the attacker.
  - The attacker can finds collisions in the hash function even when the IV is random and secret.

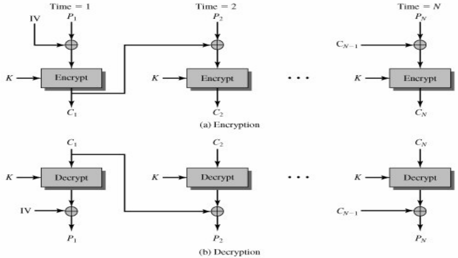
  $\Rightarrow$ Related to finding collisions of the exploited hash function

MACs from Block Ciphers: CBC-MAC

- Combined with a block cipher and modes of operations
- AES with CBC Modes is the most popular in practice

Recall: Cipher Block Chaining (CBC) Mode

- $\mathrm{Enc}(K, P_i) = \begin{cases} \mathrm{Enc}_{\mathsf{Block}}(K, IV \oplus P_1) & \text{for the first block} \\ \mathrm{Enc}_{\mathsf{Block}}(K, C_{i-1} \oplus P_i) & \text{for other blocks} \end{cases}$
- $\mathrm{Dec}(K, C_i) = \begin{cases} \mathrm{Dec}_{\mathsf{Block}}(K, C_1) \oplus IV & \text{for the first block} \\ \mathrm{Dec}_{\mathsf{Block}}(K, C_i) \oplus C_{i-1} & \text{for other blocks} \end{cases}$



Description of CBC-MAC

- CBC-MAC Generation: Given a secret key k, the initial value IV, and a message x divided into n blocks, $x_1, \ldots, x_n$, it computes

  $$y_1 = \mathrm{Enc}_{\mathrm{Block}}(k, IV \oplus x_1)$$

  and

  $$y_i = \mathrm{Enc}_{\mathrm{Block}}(k, x_i \oplus y_{i-1})$$

  for $2 \leq i \leq n$ and returns (x, y_n).

- CBC-MAC Verification: Given (x, y_n), check whether

  $$y_n \stackrel{?}{=} \text{CBC-MAC}_k(x)$$

  using the above CBC-MAC generation algorithm

References

PP10 C. Paar and J. Pelzl, Understanding Cryptography, Springer, 2010

Sta05 W. Stallings, Cryptography and Network Security: Principles and Practice, 4th edition, Pearson Prentice Hall, 2005