Network Security

Cryptographic Hash Functions Add-on

Benjamin’s slides are authoritative

Motivation (1)

- Common practice in data communications: *error detection code*, to identify random errors introduced during transmission
  - Examples: Parity, Bit-Interleaved Parity, Cyclic Redundancy Check (CRC)
- ➔ Underlying idea of these codes: add redundancy to a message for being able to *detect*, or even *correct* transmission errors
- ➔ The error detection/correction code of choice and its parameters: trade-off between
  - Computational overhead
  - Increase of message length
  - Probability/characteristics of errors on the transmission medium

Motivation (2)

- Essential security goal: *Data integrity*

  ➔ We received message *m*. Has *m* been modified by an attacker?

- It is a different (and much harder!) problem to determine if *m* has been *modified on purpose!*
- Consequently, we need to add a code that fulfills some additional properties which should make it *computationally infeasible* for an attacker to tamper with messages

- Outline
  1. Cryptographic Hash Functions
  2. Message Integrity and Authenticity based on Crypto. Hash Functions

Overview

- ❑ **Cryptographic Hash Function**

Cryptographic Hash Functions: Definition

- Definition: A function *h* is called a ***hash function*** if
  - *Compression: h* maps an input *x* of arbitrary finite bit length to an output *h(x)* of fixed bit length *n*:

    $h: \{0,1\}^* \rightarrow \{0,1\}^n$
  - *Ease of computation:* Given *h* and *x* it is *easy* to compute *h(x)*

- Definition: A function *h* is called a **one-way function** if
  - *h* is a *hash function*
  - for essentially all pre-specified outputs *y*, it is *computationally infeasible*
    to find an *x* such that $h(x) = y$

- Example: given a large prime number *p* and a primitive root *g* in Z^*_p

  Let $h(x) = g^x \bmod p$

  Then *h* is a one-way function

Cryptographic Hash Functions: Definition

- Definition: A function H is called a **cryptographic hash function** if
  1. H is a *one-way function*
     Also called *1st pre-image resistance:*
     For essentially all pre-specified outputs y, it is *computationally infeasible* to find an x such that $H(x) = y$
  2. *2nd pre-image resistance:*
     Given x it is *computationally infeasible* to find any second input x' with $x \neq x'$ such that $H(x) = H(x')$
     Note: This property is very important for digital signatures.
  3. *Collision resistance:*
     It is *computationally infeasible* to find any pair (x, x') with $x \neq x'$ such that $H(x) = H(x')$
  4. *Sometimes: Random oracle property:*
     It is computationally infeasible to distinguish $H(m)$ from random n-bit value

General Remarks (2)

- In networking there are codes for error detection.
- Cyclic redundancy checks (CRC)
  - CRC is commonly used in networking environments
  - CRC is based on binary polynomial division with Input / CRC divisor (divisor depends on CRC variant).
  - The remainder of the division is the resulting error detection code.
  - CRC is a fast compression function.
- Why not use CRC?
  - CRC is <u>not</u> a cryptographic hash function
  - CRC does not provide 2nd pre-image resistance and collision resistance
  - CRC is additive
    - If $x' = x \oplus \Delta$, then $CRC(x') = CRC(x) \oplus CRC(\Delta)$
  - CRC is useful for protecting against noisy channels
  - But not against intentional manipulation

Overview

- MAC and other applications

Case:
No attacker

Alice (A)

m, H(m) →

Bob (B)

ok

Case:
With attacker

Alice (A)

m, H(m) →

m‘, H(m‘) →

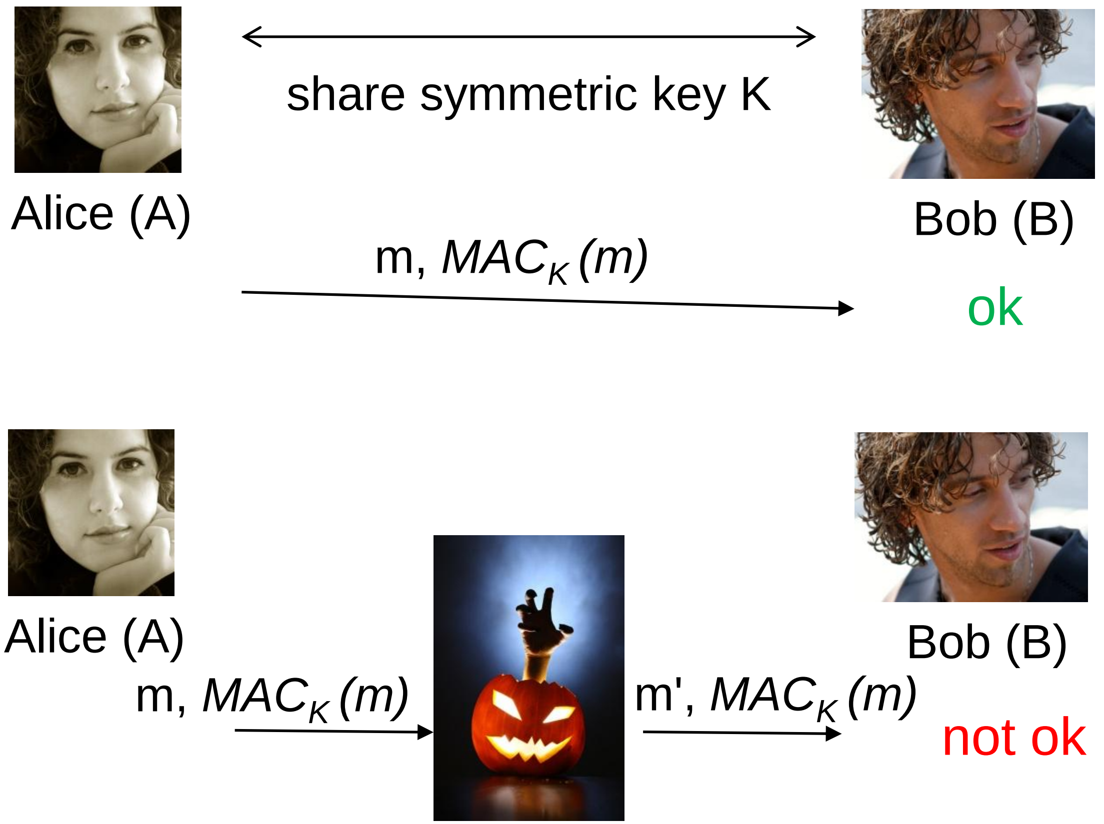
Bob (B)

ok

- Applying a hash function is not sufficient to secure a message.
- *H(m)* needs to be protected.

Case:
No attacker

Case:
With attacker



- Since the secret key k is unknown to the attacker, the attacker cannot compute $MAC_K(m')$

A Simple Attack Against an Insecure MAC

- For illustrative purposes, consider the following MAC definition:
  - Input: message $m = (x_1, x_2, ..., x_n)$ with x_i being 128-bit values, and key K
  - Compute $\Delta(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting XOR
  - Output: $MAC_K(m) := Enc_K(\Delta(m))$ with $Enc_K(x)$ denoting AES encryption
- The key length is 128 bit and the MAC length is 128 bit, so we would expect an effort of about 2^{127} operations to break the MAC (being able to forge messages).
- Unfortunately the MAC definition is insecure:
  - Attacker Eve wants to forge messages. Eve does not know K
  - Alice and Bob exchange a message $(m, MAC_K(m))$, Eve eavesdrops it
  - Eve can construct a message m' that yields the same MAC:
    - Let $y_1, y_2, ..., y_{n-1}$ be arbitrary 128-bit values
    - Define $\boldsymbol{y_n} := y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \Delta(m)$
    - This $\boldsymbol{y_n}$ allows to construct the new message $m' := (y_1, y_2, ..., \boldsymbol{y_n})$
    - Therefore,
$$\begin{aligned} MAC_K(m') = Enc(\Delta(m')) &= Enc_k(y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \boldsymbol{y_n})) \\ &= Enc_k(y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \Delta(m))) \\ &= Enc_k(\Delta(m))) \\ &= MAC_k(m) \end{aligned}$$
  - Therefore, $MAC_k(m)$ is a valid MAC for m'
  - When Bob receives $(m', MAC_K(m))$ from Eve, he will accept it as being originated

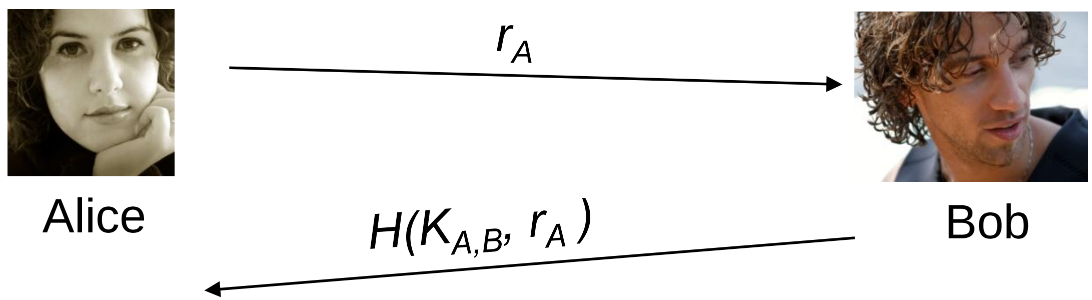
Other Applications which require some Caution

- Pseudo-random number generation
  - The output of a cryptographic hash function is assumed to be uniformly distributed
  - Although this property has not been proven in a mathematical sense for common cryptographic hash functions, such as MD5, SHA-1, it is often used
  - Start with random seed, then hash
    - b_0 = seed
    - $b_{i+1} = H(b_i \mid seed)$

- Encryption
  - Remember: Output Feedback Mode (OFB) – encryption performed by generating a pseudo random stream, and performing XOR with plain text
  - Generate a key stream as follow:
    - $k_0 = H(K_{A,B} \mid IV)$
    - $k_{i+1} = H(K_{A,B} \mid k_i)$
  - The plain text is XORed with the key stream to obtain the cipher text.

Other Applications of Cryptographic Hash Functions

- Authentication with a *challenge-response* mechanism



Other Applications of Cryptographic Hash Function

- Authentication with a *challenge-response* mechanism
  - Alice → Bob: random number “r_A”
  - Bob → Alice: “$H(K_{A,B}, r_A)$”
  - Based on the assumption that only Alice and Bob know the shared secret $K_{\mathrm{A,B}}$, Alice can conclude that an attacker would not be able to compute $H(K_{A,B}, r_A)$, and therefore that the response is actually from Bob
  - Mutual authentication can be achieved by a 2nd exchange in opposite direction
  - This authentication is based on a well-established authentication method called „*challenge-response*“
  - This type of authentication is used, e.g., by HTTP digest authentication
    - It avoids transmitting the transport of the shared key (e.g. password) in clear text
  - Another type of a challenge-response would be, e.g., if Bob signs the challenge “r_A” with his private key
  - Note that this kind of authentication does not include negotiation of a session key.
  - Protocols for key negotiation will be discussed in subsequent chapters.

Other Applications of Cryptographic Hash Functions

- Cryptographic hash values can also be used for error detection, but they are generally computationally more expensive than simple error detection codes such as CRC

Overview

- Common Structures of Hash Functions
  - Merkle-Damgård construction
  - SHA-1
  - SHA-3 and Skein

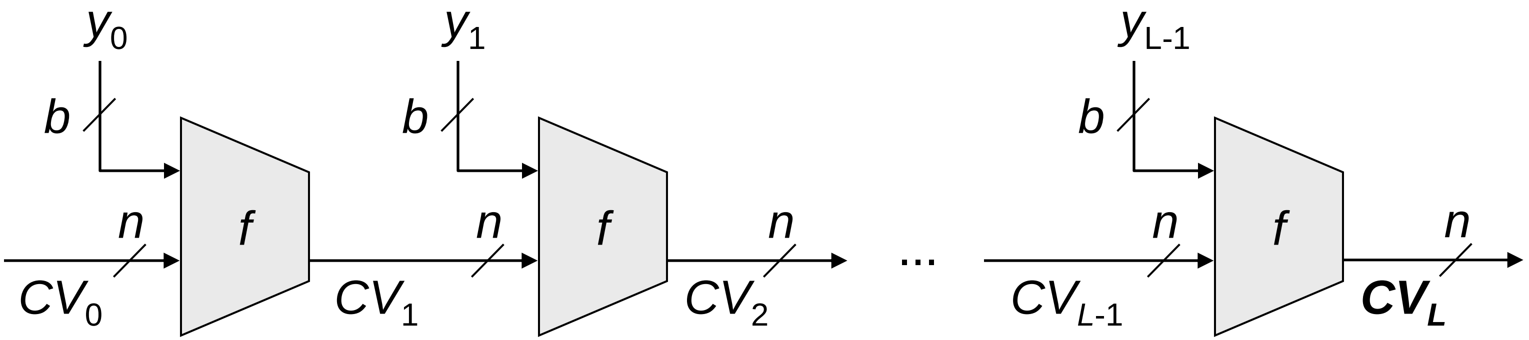
Overview of Commonly Used Cryptographic Hash Functions and Message Authentication Codes

- Cryptographic Hash Functions:
  - Message Digest 5 (MD5): **Considered broken**.
    - Invented by R. Rivest, Successor to MD4. **Considered broken**.
  - Secure Hash Algorithm 1 (SHA-1): **Considered broken**.
    - Old NIST standard.
    - Invented by the National Security Agency (NSA). Inspired by MD4.
  - Secure Hash Algorithm 3 (SHA-3):
    - Current NIST standard (since October 2012).
    - Keccak algorithm by G. Bertoni, J. Daemen, M. Peeters und G. Van Assche.

- Message Authentication Codes:
  - MACs constructed from cryptographic hash functions:
    - Example HMAC, RFC 2104, details later
  - CBC-MAC, CMAC
    - Uses blockcipher in Cipher Block Chaining mode
      (Encryption: XOR plain text with cipher text of previous block, then encrypt)
    - CMAC better than pure CBC-MAC, details later

Merkle-Damgård construction (1)

- Like many of today's block ciphers follow the general structure of a Feistel network, cryptographic hash functions such as SHA-1 follow the **Merkle-Damgård construction**:
  - Let *y* be an arbitrary message. Usually, the length of the message is appended to the message and padded to a multiple of some block size *b*. Let $(y_0, y_1, ..., y_{L-1})$ denote the resulting message consisting of *L* blocks of size *b*
  - The general structure is as depicted below:



  - *CV* is a *chaining value*, with CV_0 := *IV* and H(*y*) := CV_L
  - *f* is a specific compression function which compresses $(n + b)$ bit to *n* bit

Merkle-Damgård construction (2)

- The hash function *H* according to Merkle-Damgård construction can be summarized as follows:

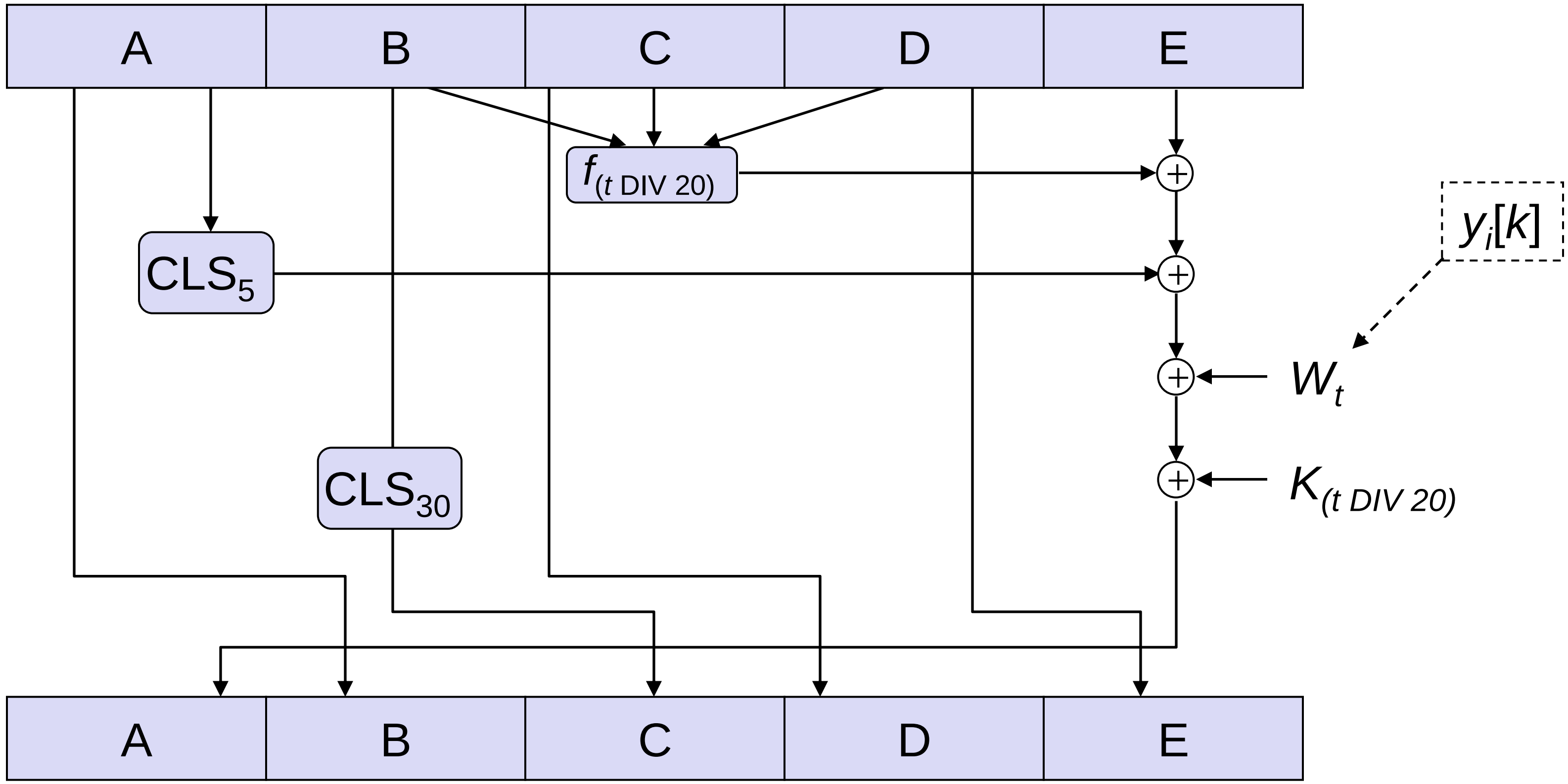
$$CV_0 = IV \quad = \text{initial n-bit value}$$
$$CV_i = f(CV_{i-1}, y_{i-1}) \quad 1 \leq i \leq L$$
$$H(y) = CV_L$$

- Security proofs by the authors [Mer89a] have shown shown that if the compression function *f* is collision resistant, then the resulting iterated hash function *H* is also collision resistant.
- However, the construction has undesirable properties like length extension attacks. The Merkle-Damgård construction can be strengthened:
  - by adding a block with the length of the message (length padding).
  - by using a wide pipe construction where the hash output has less bits than the intermediate chaining values CV_i with $i < L$.
    - Hash shorter than state good as less info leaked to attacker (e.g. against length extension). However, less search space for other attacks like brute force.

The Secure Hash Algorithm SHA-1 (1)

- Also SHA-1 follows the common structure as described above:
  - SHA-1 works on 512-bit blocks and produces a 160-bit hash value
  - Initialization
    - The data is padded, a length field is added and the resulting message is processed as blocks of length 512 bit
    - The chaining value is structured as five 32-bit registers A, B, C, D, E
    - Initialization: **`A = 0x 67 45 23 01`** **`B = 0x EF CD AB 89`**
      `C = 0x 98 BA DC FE` **`D = 0x 10 32 54 76`**
      `E = 0x C3 D2 E1 F0`
    - The values are stored in big-endian format
  - Each block y_i of the message is processed together with CV_i in a module realizing the compression function *f* in four rounds of 20 steps each.
    - The rounds have a similar structure but each round uses a different primitive logical function f_1, f_2, f_3, f_4
    - Each step makes use of a fixed additive constant K_t, which remains unchanged during one round
  - The text block y_i which consists of 16 32-bits words is „stretched“ with a recurrent linear function in order to make 80 32-bits out of it, which are required for the 80 steps:
    - $t \in \{0, ..., 15\} \quad \Rightarrow W_t := y_i[t]$
    - $t \in \{16, ..., 79\} \quad \Rightarrow W_t := CLS_1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$

The Secure Hash Algorithm SHA-1 (2) - One Step



- After step 79 each register A, B, C, D, E is added modulo 2^{32} with the value of the corresponding register before step 0 to compute CV_{i+1}

The Secure Hash Algorithm SHA-1 (3)

- The SHA-1 value over a message is the content of the chaining value CV after processing the final message block
- Security of SHA-1:
  - As SHA-1 produces a hash value of length 160 bit, it offers better security than MD5 with its 128 bits.
  - In February 2005, 3 Chinese Scientists published a paper where they break SHA-1 collision resistance within 2^{69} steps, which is much less than expected from a cryptographic hash function with an output of 160 bits (2^{80}).
  - Meanwhile down to 2^{52} steps (EuroCrypt 2009 Rump Session).
  - Up to now, no attacks on the pre-image resistance of SHA-1 have been published.

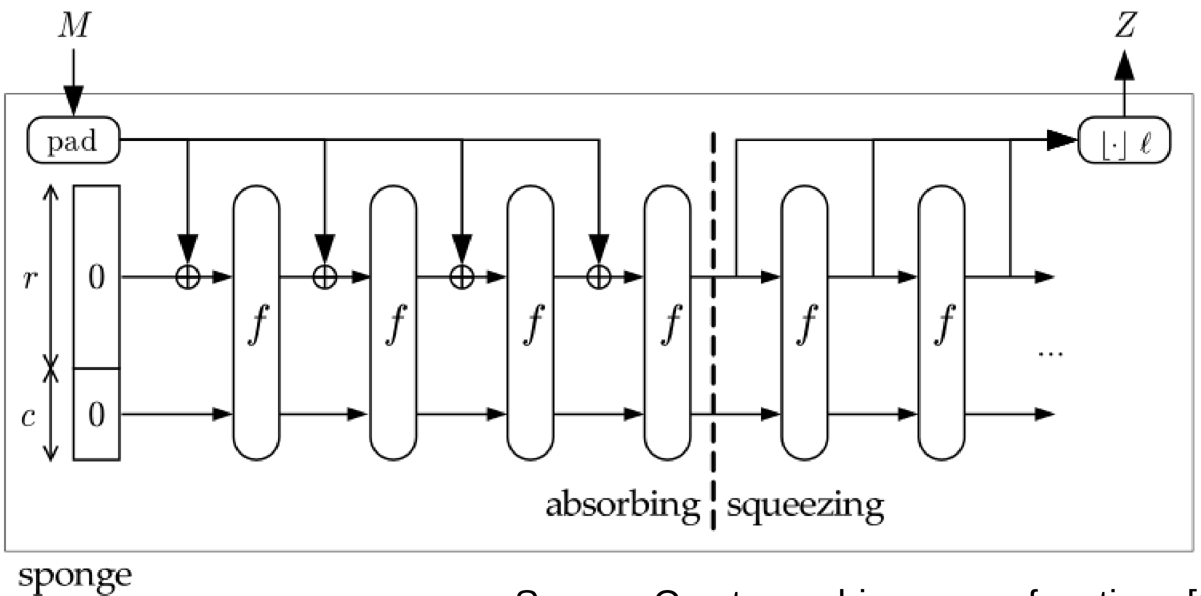
SHA-3 – a new hash standard

- MD5 is considered broken and SHA-1 is under heavy attack.
- Performance of SHA-1 worse than performance of up-to-date symmetric ciphers like AES or Twofish.

➔ NIST started a competition for a new hash function standard that will be called SHA-3 in 2007.

- NIST SHA-3 competetition
  - Requirement: fast and secure!
  - Round1: 51 candidates accepted, 13 rejected. (December 2008)
  - Round2: 14 candidates survivded. (July 2009)
  - Round3 (final): 5 candidates (BLAKE, Grostl, JH, Keccak, Skein) (December 2010)
  - **Winner (October 2012): Keccak**

SHA-3 / Keccak / Sponge Construction



Source: Cryptographic sponge functions [CSF], January 2011, http://sponge.noekeon.org/ by Keccak authors

- SHA-3 (Keccak)
  - Follows the sponge construction
  - M is padded to a multiple of the block length r
  - r=0, c=0
  - For each block i, compute f(r+mi | ci) (= Absorbing phase)
  - In squeezing phase concatenate the ri until output length reached.

SHA-3 / Keccak / Sponge Construction

- The function f follows a block cipher-like concept.
- Internal state:
  - 3d state space, 5x5 64-bit words (400 Bits)
- 256 Bit and 512 Bit blocks, 24 rounds with each 5 subrounds
- Round operations include
  - Parity in columns of the state space
  - Bitwise rotation in words
  - Permutation of words
  - A non-linear bitwise combination operation
  - XOR with round constant
- Authenticated Encryption and Tree Hash support proposed, not standardized.

SHA-3 candidate Skein

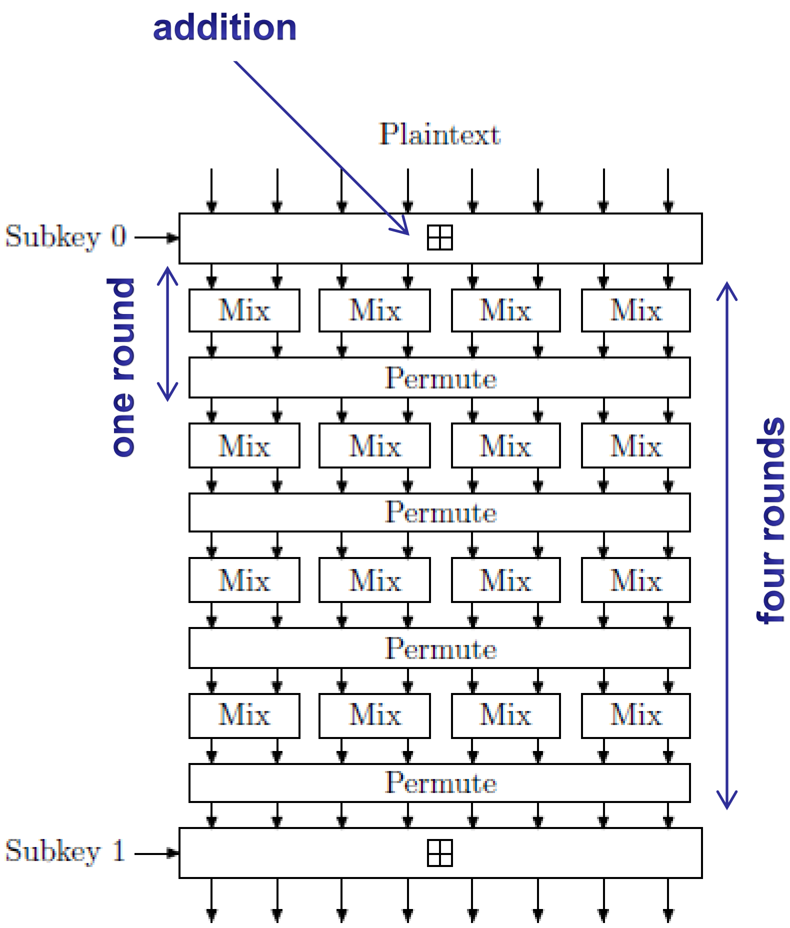
- In addition to SHA-3 finalist Skein might also get wide support in libraries and protocols due to its prominent authors .
- Variants Skein-n / Skein-n-m
  - n = size of internal state (relates to the strength of the hash function)
    - n = 512 (default), n = 1024 (conservative), n = 256 (low memory)
  - m = size of hash output
- Concept
  - Build hash function out of tweakable block cipher
  - Uses block cipher Threefish
    - 512, 1024, 256 bits key length and block length (depending on variant)
  - Unique Block Iteration (UBI) as chaining mode
    - Variable input and fixed (configurable) output size
  - Optional Argument System
    - Key, Configuration, Personalization, Public Key, Key Derivation Identifier, Nonce, Message, Output
  - Support for Tree Hashing
    - Option to process large plaintexts on parallel CPUs / machines in a tree rather than linear processing (cannot be parallelized)

Tweak

- Tweak in Skein
  - Overall size = 128 bits
  - 96 bits counter for message length
    - Incremented for each block
  - 6 bits type information
  - Bit indicates padding
  - Bit indicates first block
  - Bit indicates last block
  - Makes hash result for a plaintext subsequence position-dependent
    - E.g. harder to insert blocks that do not change chaining value to next block
    - E.g. harder to extend message and compute new MAC
    - Etc.

Threefish

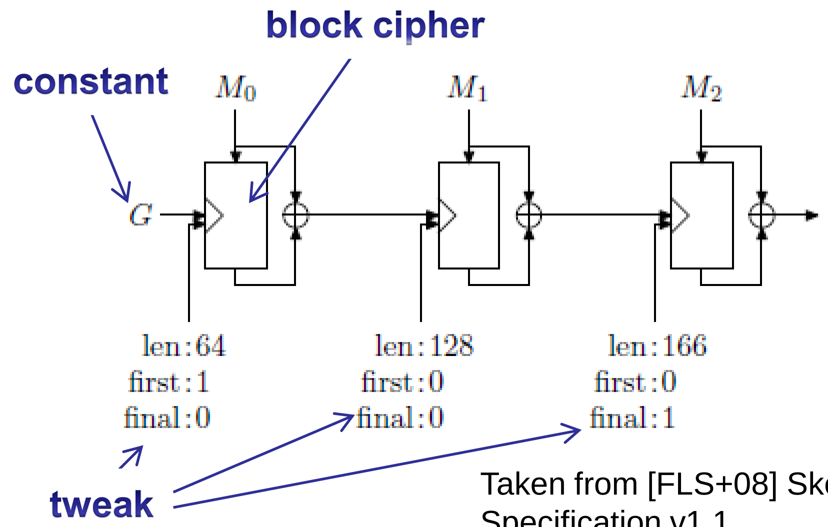
- Block size 256, 512, or 1024 bits
- Key size = block size
- Tweak size = 128 bits
- All operations on 64 bit words
- Mix operation uses
  - XOR, addition (mod 2^64), constant rotation (round and word-specific)
- 72 rounds (80 rounds for 1024 bit version)
- Subkeys
  - Are round-specific and derived from key (4, 8, or 16 words) and tweak (128 bits = 2 words)



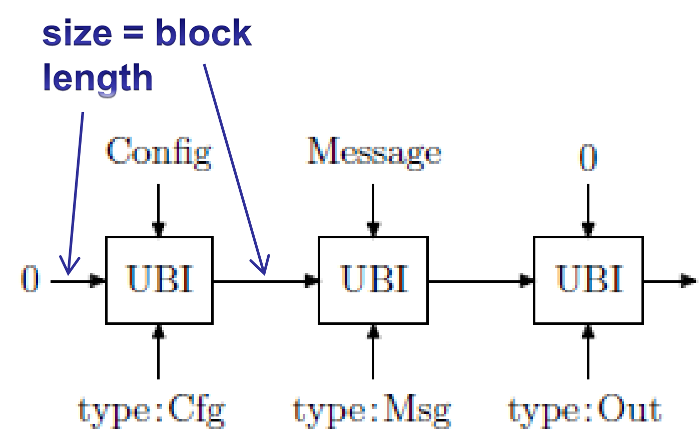
Taken from [FLS+08] Skein Specification v1.1
http://www.skein-hash.info/sites/default/files/skein1.1.pdf

Unique Block Iteration (UBI) Chaining Mode

- Unique Block Iteration (UBI)
  - Block cipher
    - Input: Message Blocks
    - Key: Tweak and chaining value
  - Chaining Value
    - XOR of output and input of block cipher
  - Tweak
    - „Counts bytes until now“ (len field)
    - Indicates first block / finalblock
- UBI in Skein
  - type field
    - Config
      - 32 byte configuration string containing fields like output length
    - Message
      - Plaintext
    - Out
      - Generates final output, input is 0.



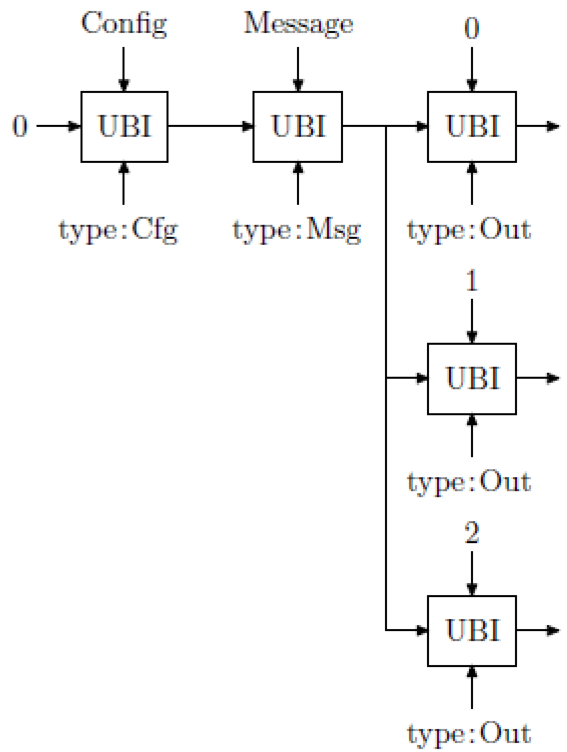
Taken from [FLS+08] Skein Specification v1.1



Taken from [FLS+08] Skein Specification v1.1
http://www.skein-hash.info/sites/default/files/skein1.1.pdf

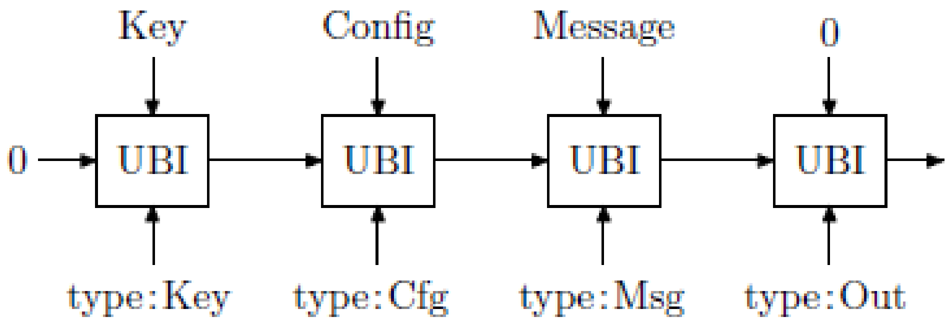
UBI in Skein – Output Generation

- Increase the output size by applying a counter mode for the output computation



Taken from [FLS+08] Skein Specification v1.1

Skein-MAC



Taken from [FLS+08] Skein Specification v1.1

- MAC usage
  - Skein can be used with HMAC and similar functions, requires two hashes
  - Faster option: use Skein with optional argument „key“
    - The key input are processed by an UBI block with the key as input, 0 as constant / initial chaining value and the tweak type information „Key“
    - This does not suffer the same weaknesses mentioned before like adding a key to the plaintext as in some weaker MAC constructions like H(k,m,k).

Overview

- Constructing MACs
  - HMAC
  - CBC-MACs
  - CMAC

Constructing a MAC from a Cryptographic Hash Functions (1)

- Reasons for constructing MACs from cryptographic hash functions :
  - Cryptographic hash functions generally execute faster than symmetric block ciphers (Note: with AES this isn't much of a problem today)
  - There are no export restrictions to cryptographic hash functions
- Basic idea: "mix" a secret key *K* with the input and compute a hash value
- The assumption that an attacker needs to know *K* to produce a valid MAC nevertheless raises some cryptographic concern:
  - The construction $H(K \mid m)$ is not secure
  - The construction $H(m, K)$ is not secure
  - The construction $H(K, p, m, K)$ with *p* denoting an additional padding field does not offer sufficient security

Constructing a MAC from a Cryptographic Hash Functions (2)

- The construction *H(K | m | K),* called prefix-suffix mode, has been used for a while.
  - See for example [RFC 1828]
  - It has been also used in earlier implementations of the Secure Socket Layer (SSL) protocol (until SSL 3.0)
  - However, it is now considered vulnerable to attack by the cryptographic community.

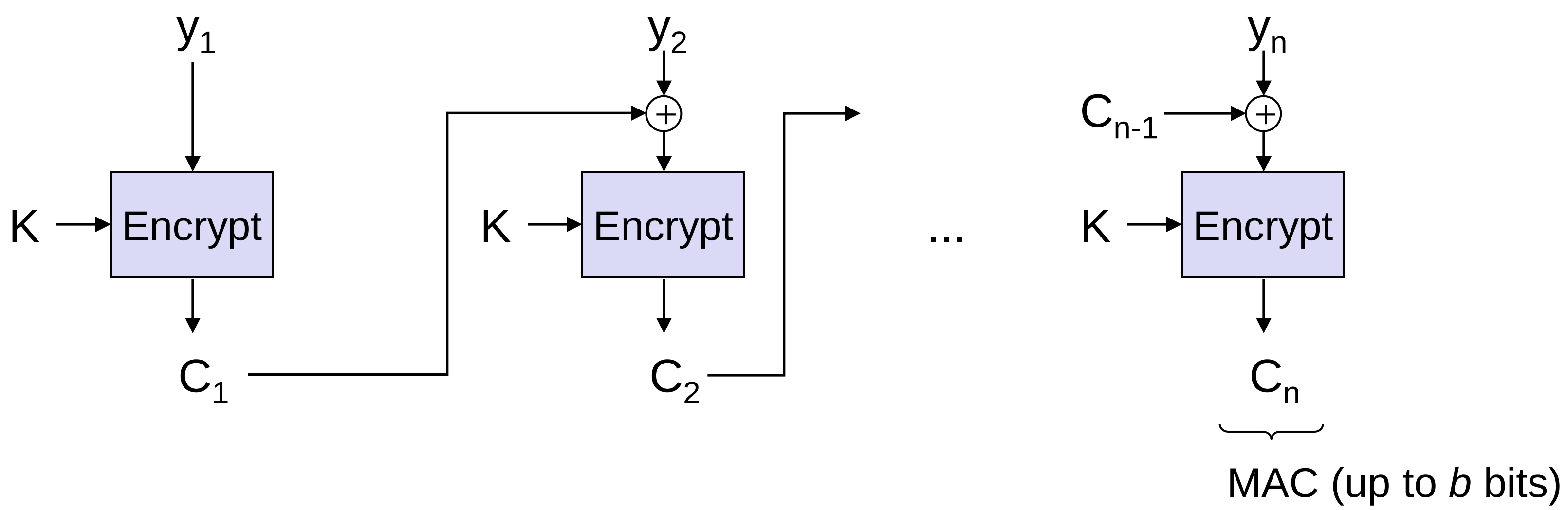
- The most used construction is **HMAC**:

$$H(K \oplus opad \mid H(K \oplus ipad \mid m))$$

  - The length of the key *K* is first extended to the block length required for the input of the hash function *H* by appending zero bytes.
  - Then it is xor'ed respectively with two constants *opad* and *ipad*
  - The hash function is applied twice in a nested way.
  - Currently no attacks have been discovered on this MAC function. (see note 9.67 in [Men97a])
  - It is standardized in RFC 2104 [Kra97a] and is called ***HMAC***

Cipher Block Chaining Message Authentication Codes (1)

- A CBC-MAC is computed by encrypting a message in CBC Mode and taking the last ciphertext block or a part of it as the MAC:



- This MAC needs not to be signed any further, as it has already been produced using a shared secret K.
- This scheme works with any block cipher (AES, Twofish, 3DES, ...)
- It is used, e.g., for IEEE 802.11 (WLAN) WPA2, many modes in SSL / IPSec use some CBC-MAC construction.

Cipher Block Chaining Message Authentication Codes (2)

- CBC-MAC security
  - CBC-MAC must NOT be used with the same key as for the encryption
  - In particular, if CBC mode is used for encryption, and CBC-MAC for integrity with the same key, the MAC will be equal to the last cipher text block
  - If the length of a message is unknown or no other protection exists, CBC-MAC can be prone to length extension attacks. CMAC resolves the issue.
- CBC-MAC performance
  - Older symmetric block ciphers (such as DES) require more computing effort than dedicated cryptographic hash functions, e.g. MD5, SHA-1 therefore, these schemes are considered to be slower.
  - However, newer symmetric block ciphers (AES) is faster than conventional cryptographic hash functions.
  - Therefore, AES-CBC-MAC is becoming popular.

Cipher-based MAC (CMAC)

- CMAC is a modification of CBC-MAC
  - Compute keys k1 and k2 from shared key k.
  - Within the CBC processing
    - XOR complete blocks before encryption with k1
    - XOR incomplete blocks before encryption with k2
    - k is used for the block encryption
  - Output is the last encrypted block or the l most significant bits of the last block.
  - AES-CMAC is standardized by IETF as RFC 4493 and its truncated form in RFC 4494.
- XCBC-MAC (e.g. found in TLS) is a predecessor of CMAC where k1 and k2 are input to algorithm and not derived from k.

Overview

- Integrity Check and Digital Signature

Integrity check with hash function / MAC

Alice (A)

share symmetric key *K*

Bob (B)

m, $MAC_K(m)$

- Alice protects her message *m* with a MAC function
- Alice has to send *m* and the MAC value to Bob.

Examples for potential MAC constructions:

- HMAC $H(K \oplus opad \,|\, H(K \oplus ipad \,|\, m))$
- CBC-MAC / CMAC
- Enc_K(h(m)) ← NO!!

Integrity check with hash function / MAC

Alice (A)

share symmetric key K

Bob (B)

m, $MAC_K(m)$

- Alice „signs“ her data m with the Message Authentication Code.
- Bob can verify the MAC code by using the shared key.
  - He reads Alice‘s $MAC_K(m)$
  - He can check if his $MAC_K(m)$ matches the one Alice signed.
  - Only Alice and Bob who know K can do this.

Take home message: for integrity checks the receiver needs to know m <u>and</u> a modification check value that it can compare.

- Think about it: Why is Enc_K(m) usually not sufficient?

Additional References I

(Beyond the scope of examination)

[Cos06] B. Coskun, N. Memon, "Confusion/Diffusion Capabilities of Some Robust Hash Functions", CISS 2006: Conference on Information Sciences and Systems, March 22-24, 2006, Princeton, NJ

[Kra97a] H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication.* Internet RFC 2104, February 1997.

[Mer89a] R. Merkle. *One Way Hash Functions and DES.* Proceedings of Crypto '89, Springer, 1989.

[Ferg03] Niels Ferguson, Bruce Schneier, „Practical Cryptography", John Wiley & Sons, 2003

[PSMD5] Peter Selinger, http://www.mscs.dal.ca/~selinger/md5collision/

[RFC1828] P. Metzger, „IP Authentication using Keyed MD5", IETF RFC 1828, August 1995

[Riv92a] R. L. Rivest. *The MD5 Message Digest Algorithm.* Internet RFC 1321, April 1992.

[Rob96a] M. Robshaw. *On Recent Results for MD2, MD4 and MD5.* RSA Laboratories' Bulletin, No. 4, November 1996.

[Yiqun05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, "Collision Search Attacks on SHA1", February 2005, <http://theory.csail.mit.edu/~yiqun/shanote.pdf>

[Yuv79a] G. Yuval. *How to Swindle Rabin.* Cryptologia, July 1979.

Additional References II

[FLS+08] Niels Ferguson, Stefan Lucks, Bruce Schneier, et. al.: Skein Specification v1.1. http://www.skein-hash.info/sites/default/files/skein1.1.pdf (accessed on 31/10/2011)

[Skein] http://www.skein-hash.info

[SHA-3] NIST (National Institute for Standards and Technology (USA)): CRYPTOGRAPHIC HASH ALGORITHM COMPETITION. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[CSF] G. Bertoni, J. Daemen, M. Peeters und G. Van Assche: Cryptographic Sponge Functions. http://sponge.noekeon.org/CSF-0.1.pdf

[Keccak3] G. Bertoni, J. Daemen, M. Peeters und G. Van Assche: Keccak Reference (version 3.0). http://keccak.noekeon.org/Keccak-reference-3.0.pdf

[Keccak] G. Bertoni, J. Daemen, M. Peeters und G. Van Assche: Keccak sponge function family main document. http://keccak.noekeon.org/Keccak-main-2.1.pdf