Network Security

Chapter 5

Modification Check Values

Motivation

- It is common practice in data communications to compute some kind of *error detection code* over messages, that enables the receiver to check if a message was altered during transmission
  - Examples: Parity, Bit-Interleaved Parity, Cyclic Redundancy Check (CRC)
- This leads to the wish of having a similar value that allows to check, if a message has been modified during transmission
- But it is a big difference, if we assume that the message will be altered by more or less random errors or *modified on purpose:*
  - If somebody wants to intentionally modify a message which is protected with a CRC value he can re-compute the CRC value after modification or modify the message in a way that it leads to the same CRC value
- So, a *modification check value* will have to fulfill some additional properties that will make it impossible for attackers to forge it
  - Two main categories of modification check values:
    - *Modification Detection Code (MDC)*
    - *Message Authentication Code (MAC)*

Cryptographic Hash Functions

- Definition: *hash function*
  - A *hash function* is a function h which has the following two properties:
    - *Compression:* h maps an input x of arbitrary finite bit length, to an output $h(x)$ of fixed bit length n.
    - *Ease of computation:* Given h and x it is *easy* to compute $h(x)$
- Definition: *cryptographic hash function*
  - A *cryptographic hash function h* is a hash function which additionally satisfies among others the following properties:
    - *Pre-image resistance:* for essentially all pre-specified outputs y, it is computationally infeasible to find an x such that $h(x) = y$
    - *2nd pre-image resistance:* given x it is computationally infeasible to find any second input x' with $x \neq x'$ such that $h(x) = h(x')$
    - *Collision resistance:* it is computationally infeasible to find any pair (x, x') with $x \neq x'$ such that $h(x) = h(x')$
  - *Cryptographic hash functions* are used to compute *modification detection codes (MDC)*

Message Authentication Codes (MAC)

- Definition: *message authentication code*
  - A *message authentication code algorithm* is a family of functions h_k parameterized by a secret key k with the following properties:
    - *Compression:* h_k maps an input x of arbitrary finite bitlength to an output $h_k(x)$ of fixed bitlength, called the MAC
    - *Ease of computation:* given k, x and a known function family h_k the value $h_k(x)$ is easy to compute
    - *Computation-resistance:* for every fixed, allowed, but unknown value of k, given zero or more text-MAC pairs $(x_i, h_k(x_i))$ it is computationally infeasible to compute a text-MAC pair $(x, h_k(x))$ for any new input $x \neq x_i$
  - Please note that *computation-resistance* implies the property of *key non-recovery*, that is k can not be recovered from pairs $(x_i, h_k(x_i))$, but computation resistance can not be deduced from key non-recovery, as the key k need not always to be recovered to forge new MACs

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 64-bit values, and key k
  - Compute $\Delta(m) := x_1 \oplus x_2 \oplus ... \oplus x_n$ with $\oplus$ denoting bitwise exclusive-or
  - Output: MAC $C_k(m) := E_k(\Delta(m))$ with $E_k(x)$ denoting DES encryption
- The key length is 56 bit and the MAC length is 64 bit, so we would expect an effort of about 2^{55} operations to obtain the key k and break the MAC (= being able to forge messages).
- Unfortunately the MAC definition is insecure:
  - Assume an attacker Eve who wants to forge messages exchanged between Alice and Bob obtains a message $(m, C_k(m))$ which has been "protected" by Alice using the secret key k shared with Bob
  - Eve can construct a message m' that yields the same MAC:
    - Let $y_1, y_2, ..., y_{n-1}$ be arbitrary 64-bit values
    - Define $y_n := y_1 \oplus y_2 \oplus ... \oplus y_{n-1} \oplus \Delta(m)$, and $m' := (y_1, y_2, ..., y_n)$
    - When Bob receives $(m', C_k(m))$ from Eve pretending to be Alice he will accept it as being originated by Alice as $C_k(m)$ is a valid MAC for m'

Applications to Cryptographic Hash Functions and MACs

- Principal application which led original design:
  - Message integrity:
    - An MDC represents a *digital fingerprint*, which can be signed with a private key, e.g. using the RSA or ElGamal algorithm, and it is not possible to construct two messages with the same fingerprint so that a given signed fingerprint can not be re-used by an attacker
    - A MAC over a message *m* directly certifies that the sender of the message possesses the secret key *k* and the message could not have been modified without knowledge of that key
- Other applications, which require some caution:
  - Confirmation of knowledge
  - Key derivation
  - Pseudo-random number generation
- Depending on the application, further requirements may have to be met:
  - *Partial pre-image resistance:* even if only a part of the input, say *t* bit, is unknown, it should take on the average 2^{t-1} operations to find these bits

Attacks Based on the Birthday Phenomenon (1)

- The Birthday Phenomenon:
  - How many people need to be in a room such that the possibility that there are at least two people with the same birthday is greater than 0.5?
  - For simplicity, we don't care about February, 29, and assume that each birthday is equally likely
- Define *P(n, k)* := Pr[at least one duplicate in *k* items, with each item able to take on of n equally likely values between 1 and *n*]
- Define *Q(n, k)* := Pr[no duplicate in *k* items, each item between 1 and *n*]
  - We are able to choose the first item from *n* possible values, the second item from *n* - 1 possible values, etc.
  - Hence, the number of different ways to choose k items out of n values with no duplicates is: $N = n \times (n - 1) \times \ldots \times (n - k + 1) = n! / (n - k)!$
  - The number of different ways to choose *k* items out of *n* values, with or without duplicates is: n^k
  - So, $Q(n, k) = N / n^k = n! / ((n - k)! \times n^k)$

Attacks Based on the Birthday Phenomenon (2)

- We have: $$P(n,k) = 1 - Q(n,k) = 1 - \frac{n!}{(n-k)! \times n^k}$$
$$= 1 - \frac{n \times (n-1) \times \ldots \times (n-k+1)}{n^k}$$
$$= 1 - \left[\frac{n-1}{n} \times \frac{n-2}{n} \times \ldots \times \frac{n-k+1}{n}\right]$$
$$= 1 - \left[\left(1 - \frac{1}{n}\right) \times \left(1 - \frac{2}{n}\right) \times \ldots \times \left(1 - \frac{k-1}{n}\right)\right]$$
- We will use the following inequality: $(1 - x) \leq e^{-x}$ for all $x \geq 0$
- So: $$P(n,k) > 1 - \left[\left(e^{-1/n}\right) \times \left(e^{-2/n}\right) \times \ldots \times \left(e^{-(k-1)/n}\right)\right]$$
$$= 1 - e^{-\left[(1/n) + (2/n) + \ldots + (k-1/n)\right]}$$
$$= 1 - e^{-k \times (k-1)/2n}$$

Attacks Based on the Birthday Phenomenon (3)

- In the last step, we used the equality: $1 + 2 + ... + (k - 1) = (k^2 - k) / 2$
  - Exercise: proof the above equality by induction
- Let's go back to our original question: how many people *k* have to be in one room such that there are at least two people with the same birthday (out of *n* = 365 possible) with probability ≥ 0,5?
  - So, we want to solve: $\frac{1}{2} = 1 - e^{-k \times (k-1)/2n}$

$$\Leftrightarrow \quad 2 = e^{k \times (k-1)/2n}$$

$$\Leftrightarrow \ln(2) = \frac{k \times (k-1)}{2n}$$

  - For large *k* we can approximate $k \times (k - 1)$ by k^2, and we get:

$$k = \sqrt{2\ln(2)n} \approx 1.18\sqrt{n}$$

  - For *n* = 365, we get *k* = 22.54 which is quite close to the correct answer 23

Attacks Based on the Birthday Phenomenon (4)

- What does this have to do with MDCs?
- We have shown, that if there are *n* possible different values, the number *k* of values one needs to randomly choose in order to obtain at least one pair of identical values, is on the order of $\sqrt{n}$
- Now, consider the following attack [Yuv79a]:
  - Eve wants Alice to sign a message *m1*, Alice normally never would sign. Eve knows that Alice uses the function MDC1(*m*) to compute an MDC of *m* which has length *r* bit before she signs this MDC with her private key yielding her digital signature.
  - First, Eve produces her message *m1*. If she would now compute MDC1(*m*1) and then try to find a second harmless message *m2* which leads to the same MDC her search effort in the average case would be on the order of $2^{(r-1)}$.
  - Instead she takes any harmless message *m2* and starts producing variations *m1'* and *m2'* of the two messages, e.g. by adding <space> <backspace> combinations or varying with semantically identical words.

Attacks Based on the Birthday Phenomenon (5)

- As we learned from the birthday phenomenon, she will just have to produce about $\sqrt{2^r} = 2^{r/2}$ variations of each of the two messages such that the probability that she obtains two messages *m1'* and *m2'* with the same MDC is at least 0.5
- As she has to store the messages together with their MDCs in order to find a match, the memory requirement of her attack is on the order of $2^{r/2}$ and its computation time requirement is on the same order
- After she has found *m1'* and *m2'* with *MDC1(m1') = MDC1(m2')* she asks Alice to sign *m2'*. Eve can then take this signature and claim that Alice signed *m1'*.
- Attacks following this method are called *birthday attacks*
- Consider now, that Alice uses RSA with keys of length 2048 bit and a cryptographic hash function which produces MDCs of length 96 bit.
  - Eves average effort to produce two messages *m1'* and *m2'* as described above is on the order of 2^{48}, which is feasible today. Breaking RSA keys of length 2048 bit is far out of reach with today's algorithms and technology.

Overview of Commonly Used MDCs

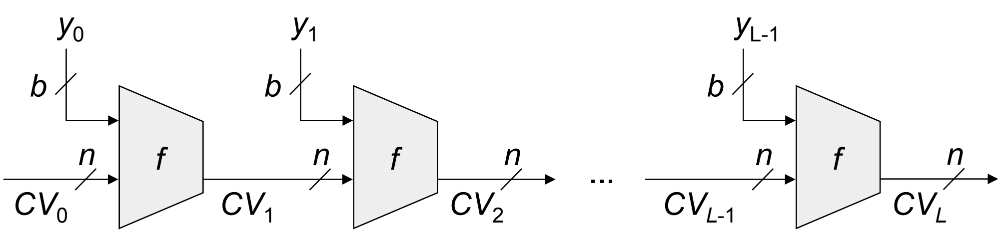
- Cryptographic Hash Functions for creating MDCs:
  - Message Digest 5 (MD5):
    - Invented by R. Rivest
    - Successor to MD4
  - Secure Hash Algorithm 1 (SHA-1):
    - Invented by the National Security Agency (NSA)
    - The design was inspired by MD4
  - Secure Hash Algorithm 2 (SHA-2 also SHA-256 & SHA-512)
    - Also designed by the National Security Agency (NSA)
    - Also Merkle-Dåmgard-Contruction
    - Larger block size & more complex round function
  - Secure Hash Algorithm 3 (SHA-3, Keccak)
    - Winner of an open competition
    - So-called Sponge construction
    - Much more versatile than previous hash functions

Overview of Commonly Used MACs

- Message Authentication Codes (MACs):
  - DES-CBC-MAC:
    - Uses the Data Encryption Standard in Cipher Block Chaining mode
    - In general, the CBC-MAC construction can be used with any block cipher
  - MACs constructed from MDCs:
    - This very common approach raises some cryptographic concern as it makes some implicit but unverified assumptions about the properties of the MDC
- Authenticated Encryption with Associated Data (AEAD)
  - Galois-Counter-Mode (GCM)
    - Uses a block-cipher to encrypt and authenticate data
    - Fast in networking applications
  - Sponge Wrap
    - Uses a SHA-3 like hash function to encrypt and authenticate data

Common Structure of Cryptographic Hash Functions (1)

- Like many of today's block ciphers follow the general structure of a Feistel network, many cryptographic hash functions in use today follow a common structure, the so-called Merkle-Dåmgard structure:
  - Let y be an arbitrary message. Usually, the length of the message is appended to the message and it is 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 $MDC(y) := CV_L$
  - f is a specific compression function which compresses $(n + b)$ bit to n bit

Common Structure of Cryptographic Hash Functions (2)

- The hash function *H* can be summarized as follows:

  $CV_0 = IV$ = initial n-bit value

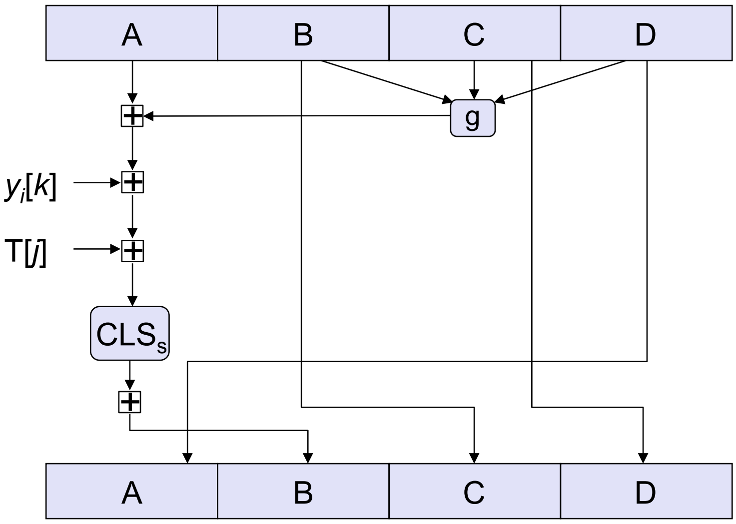
  $CV_i = f(CV_{i-1}, y_{i-1})$ $1 \leq i \leq L$

  $H(y) = CV_L$
- It has been shown [Mer89a] that if the compression function *f* is collision resistant, then the resulting iterated hash function *H* is also collision resistant.
- Cryptanalysis of cryptographic hash functions thus concentrates on the internal structure of the function *f* and finding efficient techniques to produce collisions for a single execution of *f*
- Primarily motivated by birthday attacks, a common minimum suggestion for *n*, the bitlength of the hash value, is 160 bit, as this implies an effort of order 2^{80} to attack which is considered infeasible today

The Message Digest 5 (1)

- MD5 follows the common structure outlined before (e.g. [Riv92a]):
  - The message y is padded by a "1" followed by 0 to 511 "0" bits such that the length of the resulting message is congruent 448 modulo 512
  - The length of the original message is added as a 64-bit value resulting in a message that has length which is an integer multiple of 512 bit
  - This new message is divided into blocks of length *b* = 512 bit
  - The length of the chaining value is *n* = 128 bit
    - The chaining value is "structured" as four 32-bit registers A, B, C, D
    - Initialization: `A := 0x 01 23 45 67` `B := 0x 89 AB CD EF`
      `C := 0x FE DC BA 98` `D := 0x 76 54 32 10`
  - Each block of the message y_i is processed with the chaining value CV_i with the function *f* which is internally realized by 4 rounds of 16 steps each
    - Each round uses a similar structure and makes use of a table T containing 64 constant values of 32-bit each,
    - Each of the four rounds uses a specific logical function g

The Message Digest 5 (2) - Structure of One Step



- The function g is one of four different logical functions
- $y_i[k]$ denotes the k^{th} 32-bit word of message block i
- T[j] is the j^{th} entry of table t with j incremented modulo 64 every step
- CLS_s denotes cyclical left shift by s bits with s following some schedule

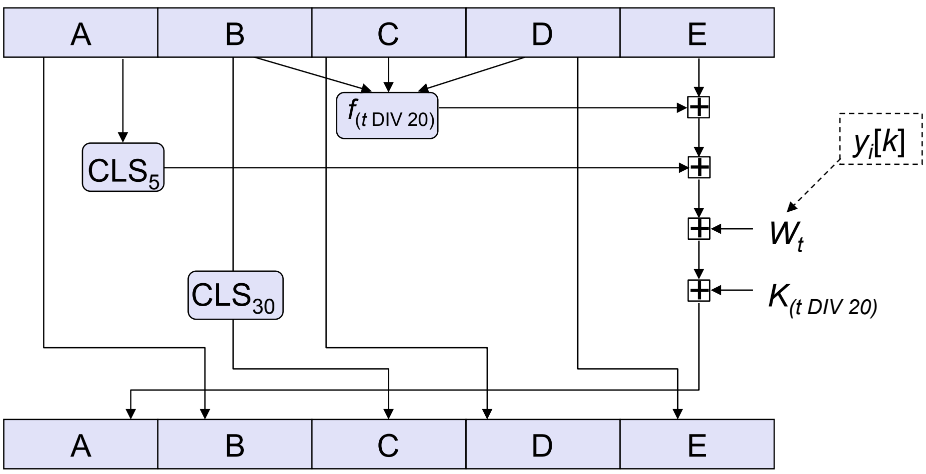
The Message Digest 5 (3)

- The MD5-MDC over a message is the content of the chaining value CV after processing the final message block
- Security of MD5:
  - Every bit of the 128-bit hash code is a function of every input bit
  - In 1996 H. Dobbertin published an attack that allows to generate a collision for the function f (realized by the 64 steps described above).
  - Took until 2004 before a first collision was found [WLYF04]
  - By now it is possible to generate collisions within seconds on general purpose hardware [Kl06]
  - **MD5 must not be considered if collision resistance is required!**
    - This is often the case!
    - Examples: Two postscripts with different texts but equal hashes [LD05], Certificates one for an assured domain and one for an own certificate authority [LWW05], Any message that is extendable [KK06]
  - The resistance against preimage attacks is with $2^{123.4}$ calculations still o.k [SA09]

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
  - As its design was also inspired by the MD4 algorithm, its initialization is basically the same like that of MD5:
    - 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
    - Intialization: **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 Secure Hash Algorithm SHA-1 (2) - One Step



- $t \in \{0, ..., 15\} \Rightarrow W_t := y_i[t]$
  $t \in \{16, ..., 79\} \Rightarrow W_t := CLS_1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$
- 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}

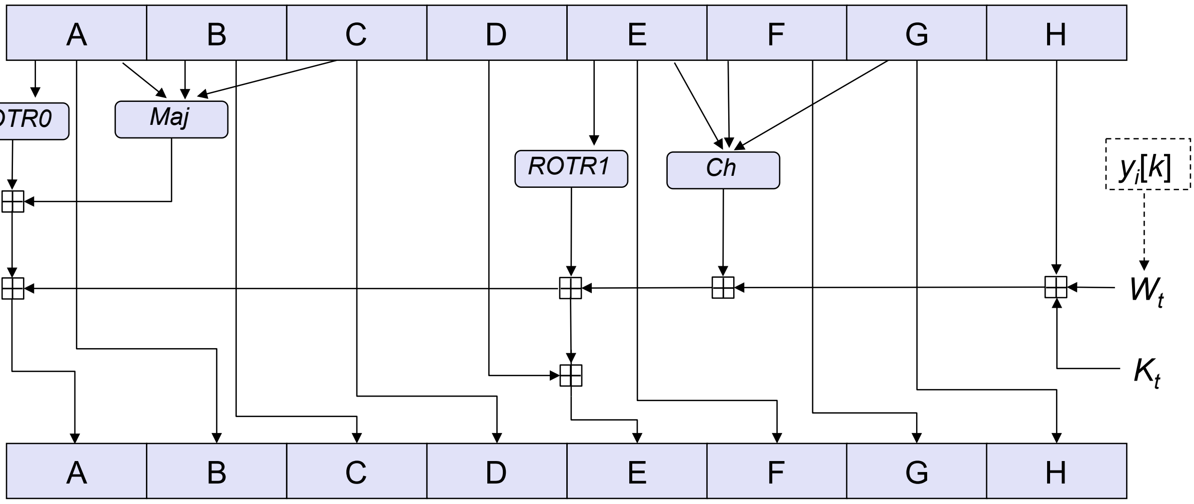
The Secure Hash Algorithm SHA-1 (3)

- The SHA-1-MDC over a message is the content of the chaining value CV after processing the final message block
- Comparison between SHA-1 and MD5:
  - Speed: SHA-1 is about 25% slower than MD5 (CV is about 25% bigger)
  - Simplicity and compactness: both algorithms are simple to describe and implement and do not require large programs or substitution tables
- Security of SHA-1:
  - As SHA-1 produces MDCs of length 160 bit, it is expected to offer better security against brute-force and birthday attacks than MD5
  - In February 2005, X. Wang et. al. published an attack that allows to find a collision with an effort of 2^{69} that was improved to 2^{63} in the months to follow and published in [WYY05a]
  - Research continues (e.g. [Man11]) however no practical collisions are known yet
  - Some inherent weaknesses of Merkle-Dåmgard constructions, e.g. [KK06], apply

The Secure Hash Algorithm SHA-2 family (1)

- In 2001, the NIST published a new standard FIPS PUB 180-2 containing new variants, called *SHA-256*, *SHA-384*, and *SHA-512* [NIST02] with 256, 384, and 512 bits output
  - SHA-224 was added in 2004
- SHA-224 and SHA-384 are truncated versions of SHA-256 and SHA-512 with different initialization values
- SHA-2 uses also Merkle-Dåmgard construction with a block size of 512 bits (SHA-256) and 1024 bits (SHA-512)
- The internal state is organized in 8 registers of 32 bit (SHA-256) and 64 bit (SHA-512)
- 64 rounds (SHA-256) or 80 rounds (SHA-512)

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



- $t \in \{0, ..., 15\}$ $\Rightarrow W_t := y_i[t]$
  $t \in \{16, ..., r\}$ $\Rightarrow W_t := W_{t-16} \oplus \sigma_0(W_{t-15}) \oplus W_{t-7} \oplus \sigma_1(W_{t-2})$
- K_t is the fractional part of the cube root of the t^{th} prime number
- The ROTR and σ functions XOR different shifts of the input value
- Ch and Maj are logic combinations of the input values

The Secure Hash Algorithm SHA-2 family (3)

- All-in-all design very similar to SHA-1
- Due to size and more complicated round functions about 30-50 percent slower than SHA-1 (varies for 64-bit and 32-bit systems!)
- Security discussion:
  - Already in 2004 it was discovered that a simplified version of the algorithm (with XOR instead of addition and symmetric constants) generates highly correlated output [GH04]
  - For round-reduced versions of SHA-2 pre-image attacks exists that are faster than brute-force, but highly impractical (e.g. [AGM09])
  - Even though size and complexity do not allow for attacks currently the situation is uncomfortable
  - Led to the need for a new SHA-3 standard

The Secure Hash Algorithm SHA-3 (1)

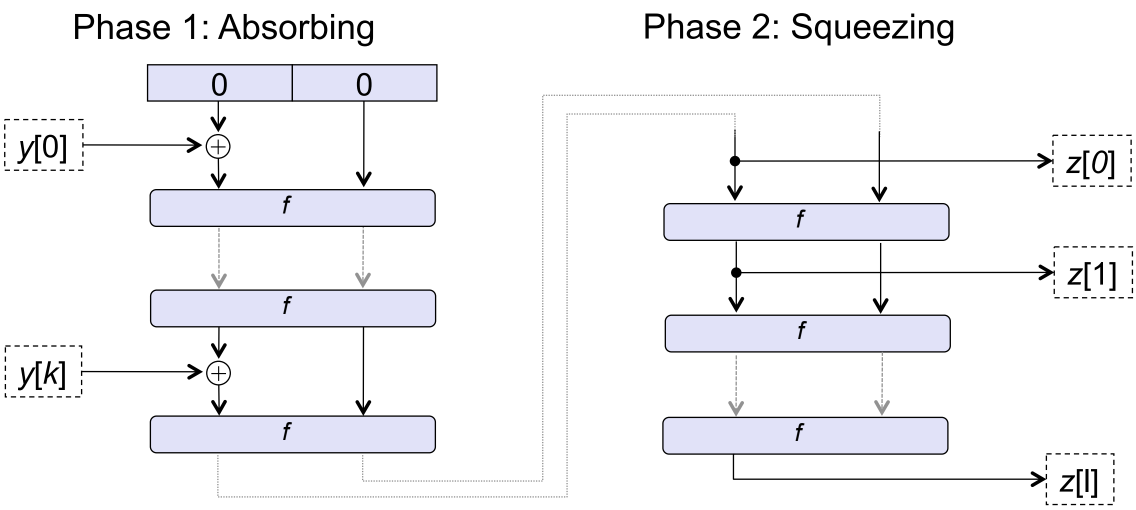
- Security concerns about SHA-1 and SHA-2 led to an open competition by the NIST which started in 2007
- 5 finalists without notable weaknesses

- October 2012: NIST announces Keccak to become SHA-3
- 4 European inventors
  - One is Joan Daemen, who co-designed AES
- SHA-3 is very fast, especially in hardware
- Very well documented and analyzable

The Secure Hash Algorithm SHA-3 (2)

- Keccak is based on a so-called *sponge* construction instead of the previous Merkle-Dåmgard constructs
  - Versatile design to implement nearly all symmetric cryptographic functions (however only the hashing is standardized)
- Usually works in 2 phases
  - "Absorbing" information of arbitrary length into 1600 bit of internal state
  - "Squeezing" (i.e. outputting) hashed-data of arbitrary length (only 224, 256, 384, and 512 bits standardized)
- The internal state is organized in 2 registers
  - One register of the size r is "public": input data is XORed to it in absorbing phase, output data is derived from it in squeezing phase
  - The register of size c is "private"; in- and output does not affect it directly
  - In Keccak the size of the registers is 1600 bits (i.e. c + r = 1600 bits)
  - The size of c is twice as large as the output block length
  - Both registers are initialized with "0"
- The hashing occurs due a function *f* that reads the registers and outputs a new state

SHA-3 (3) – Sponge Construction



- Absorbing phase: k + 1 input blocks of size *r* are mixed to the state
- Squeezing phase: l + 1 output blocks of size *r* are generated (often only one)
- The last input and output block may be padded or cropped

SHA-3 (4) – The function *f*

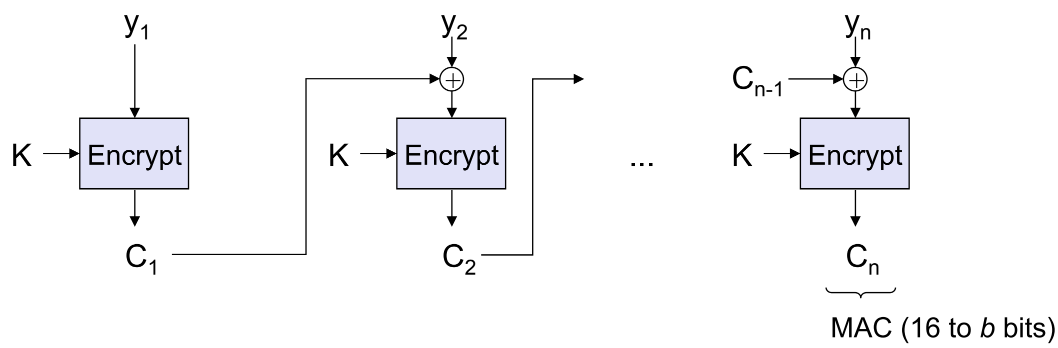
- Obviously, the security of a sponge construction depends on the security of *f*
- In Keccak uses 24 rounds of 5 different sub-functions (θ, ρ, π, χ, ι) to implement *f*
- Sub-functions operate on a “three-dimensional” bit array *a*[*5*][*5*][*w*] with *w* is chosen in correspondence with the size *r* and *c*
- All operations are performed over GF(2^n)
- Each of the sub-functions ensures certain properties, e.g.,
  - Fast diffusion of changed bits throughout the state (θ)
  - Long term diffusion (π)
  - Ensuring that *f* becomes non-linear (χ)
  - Round-specific substitution (ι)
- θ is executed first to ensure that secret and public state mix quickly before applying other sub-functions

SHA-3 (5) – Security

- Currently no notable weaknesses exist in SHA-3
  - Best known pre-image attacks work with up to 8-round function f only
  - To protect against internal collisions 11 rounds are supposed to be enough
- In comparison to SHA-1 and SHA-2 additional security properties are guaranteed as internal state is never made public
  - Prevents attacks were arbitrary information is added to a valid secret message
  - Provides *Chosen Target Forced Prefix (CTFP)* preimage resistance [KK06], i.e. it is not possible to construct a message m = P || S, where P is fixed and S is arbitrary chosen, s.t., H(m) = y
    - For Merkle-Dåmgard constructions this is only as hard as collision resistance
  - No fast way to generate multi-collisions quickly [Jou04]

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
  - However, it is not possible to say who exactly has created a MAC, as everybody (sender, receiver) who knows the secret key K can do so
- This scheme works with any block cipher (DES, IDEA, ...)

Cipher Block Chaining Message Authentication Codes (2)

- Security of CBC-MAC:
  - As an attacker does not know *K*, a birthday attack is much more difficult to launch (if not impossible)
  - Attacking a CBC-MAC requires known (message, MAC) pairs
  - This allows for shorter MACs
  - A CBC-MAC can optionally be strengthened by agreeing upon a second key K' ≠ K and performing a triple encryption on the *last* block:

$$\text{MAC} := E(K, D(K', E(K, C_{n-1})))$$

  - This doubles the key space while adding only little computing effort
  - **The construction is not secure, when message lengths vary!**
- There have also been some proposals to create MDCs from symmetric block ciphers with setting the key to a fixed (known) value:
  - Because of the relatively small block size of 64 bit of most common block ciphers, these schemes offer insufficient security against birthday attacks
  - As symmetric block ciphers require more computing effort than dedicated cryptographic hash functions, these schemes are relatively slow

Constructing a MAC from an MDC

- Reason to construct MACs from MDCs Cryptographic hash functions generally execute faster than symmetric block ciphers
- Basic idea: "mix" a secret key *K* with the input and compute an MDC
- The assumption that an attacker needs to know *K* to produce a valid MAC nevertheless raises some cryptographic concern (at least for Merkle-Dåmgard hash functions):
  - The construction *H(K, m)* is not secure (see note 9.64 in [Men97a])
  - The construction *H(m, K)* is not secure (see note 9.65 in [Men97a])
  - The construction *H(K, p, m, K)* with *p* denoting an additional padding field does not offer sufficient security (see note 9.66 in [Men97a])
- The most used construction is: $H(K \oplus p_1, H(K \oplus p_2, m))$
  - Key is padded with 0's to fill up the key to one input block of the cryptographic hash function
  - Two different constant patterns p_1 and p_2 XORed to the padded key
  - This scheme seems to be secure (see note 9.67 in [Men97a])
  - It has been standardized in RFC 2104 [Kra97a] and is called *HMAC*

Authenticated Encryption with Associated Data (AEAD) Modes

- Usually it data is not authenticated or encrypted but encrypted AND authenticated (blocks $P_0 \ldots P_n$)
- Sometimes additional data needs to be authenticated (e.g. packet headers), in the following denoted $A_0 \ldots A_m$
- Led to the development of AEAD modes of operation
- Examples are
  - Galois/Counter Mode (GCM)
  - Counter with CBC-MAC (CCM)
  - Offset Codebook Mode (OCM)
  - SpongeWrap – a method to use Keccak for AEAD operation

Galois/Counter Mode (GCM) [MV04]

- Popular AEAD mode
- NIST standard, part of IEEE 802.1AE, IPsec, TLS, SSH etc.
- Free of patents
- Mainly used in networking applications for its high speed
  - Extremely efficient in hardware
  - Processor support on newer x86 CPUs
  - Time intensive tasks may be pre-calculated and parallelized
  - No need for padding

- Uses conventional block cipher with 128 bit block size (e.g. AES)
- Calculates MAC by multiplications and additions in $GF(2^{128})$ over the irreducible polynomial $x^{128}+x^7+x^2+x+1$
- Requires only n + 1 block cipher calls per packet (n = length of encrypted and authenticated data)

Small Excursion: Calculation Operations in GF(2^n) (I)

- Galois field arithmetic defined over terms (e.g. $a_3x^3+a_2x^2+a_1x+a_0$)
- Coefficients are elements of the field $\mathbb{Z}_2$, i.e. either 0 or 1
- Often only the coefficients are stored, so $x^4+x^2+x^1$ becomes 0x16

- Addition in GF(2^n) is simply the addition of terms
  - As equal coefficients map to 0, just XOR the values!
  - Extreme fast in hard- and software!

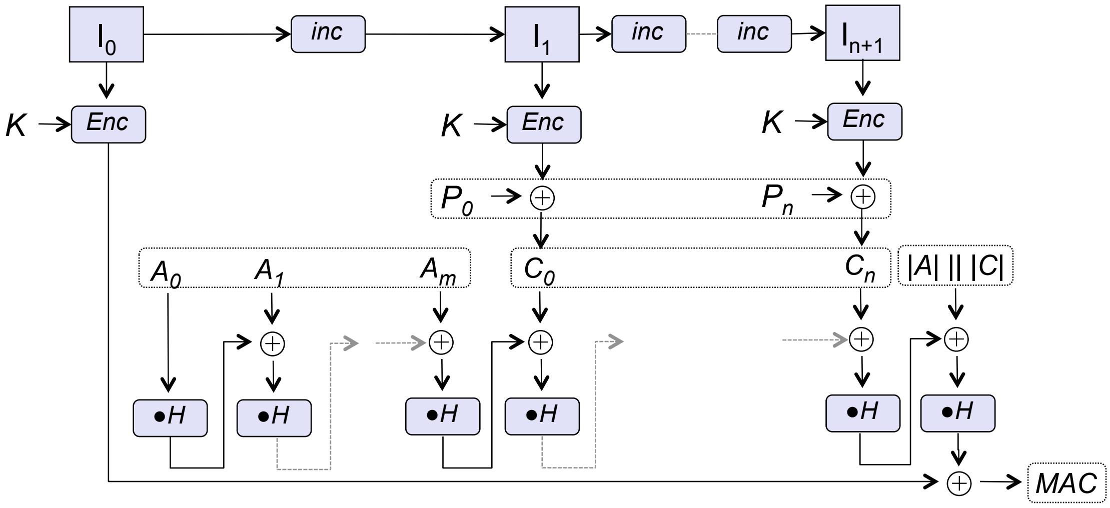
- Multiplication in GF(2^n) is polynomial multiplication and a subsequent modulo division by an irreducible polynomial of degree *n*
  - Irreducible polynomials are not divisible without remainder by any other polynomial except "1", somewhat like prime numbers in GF
  - Can be implemented by a series of shift and XOR operations
  - Very fast in hardware or on newer Intel CPUs (with CLMUL Operations)
  - Modulo operation could be performed like in a regular CRC calculation

Small Excursion: Calculation Operations in GF(2^n) (II)

- Addition Example:
  - $x^3+x+1 \oplus x^2+x = x^3+x^2+1 \leftrightarrow$ 0x0B XOR 0x06 = 0x0D
- Multiplication Example (over x^4+x+1):
  - $x^3+x+1 \bullet x^2+x = x^5+x^3+x^2 \oplus x^4+x^2+x \text{ MOD } x^4+x+1 =$ $x^5+x^4+x^3+x \text{ MOD } x^4+x+1 = x^3+x^2+x+1$

- Elements of GF(2^n) (except for 1 and the irreducible polynomial) may be a generator for the group
  - Example for x and the polynomial x^4+x+1: $x, x^2, x^3, x+1, x^2+x, x^3+x^2, x^3+x+1, x^2+1, x^3+x, x^2+x+1, x^3+x^2+x, x^3+x^2+x+1, x^3+x^2+1, x^3+1, 1, x, \ldots$
- Other concepts of finite groups also apply, e.g., every element has a multiplicative inverse element
  - May be found by an adapted version of the Extended Euclidian Algorithm

Galois/Counter Mode (GCM) (2)



- I_0 is initialized with the IV and a padding, or a hash of the IV (if it is not 96 bits)
- •H is $GF(2^{128})$ multiplication with $H = E(K, 0^{128})$
- Input blocks A_m and P_n are padded to 128 bits
- A_m & C_n are truncated to original size before output
- The last authentication uses 64 bit encoded bit lengths of A and C

Galois/Counter Mode (GCM) (3) - Security

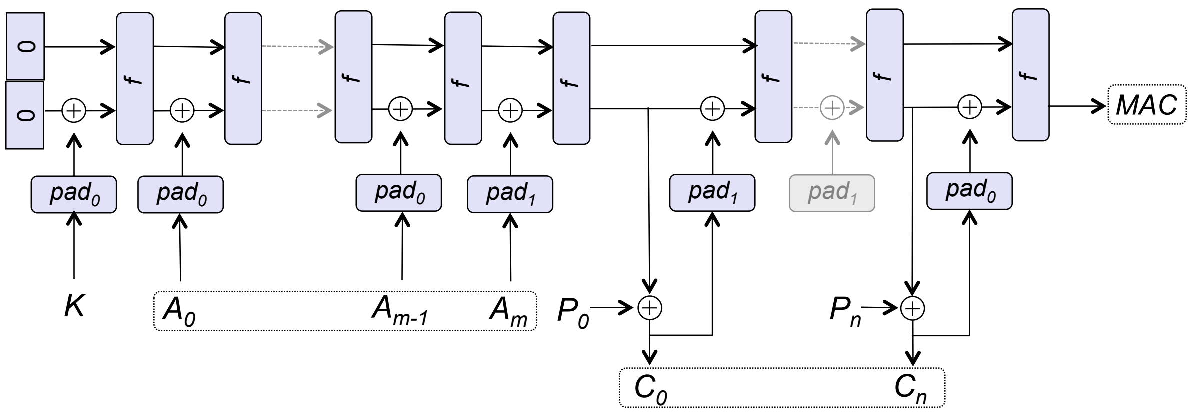
- Fast mode, but needs some care:
  - Proven to be secure (under preconditions, e.g. used block cipher is not distinguishable from random numbers), but construction is fragile:
  - IVs MUST NOT be reused, otherwise streams can be XORed and the XOR of the streams can be recovered, may lead to an instant recovery of the secret value "H"
  - H has a possible weak value 0^{128}, in this case authentication will not work and if IVs of a length other than 96 bits are used, C_0 will always be the same!
  - Some other keys generate hash keys with a low order, which must be avoided… [Saa11]
  - Successful forgery attempts may leak information about H, thus short MAC lengths MUST be avoided or risk-managed [Dwo07]
  - The achieved security is only 2^{t-k} not 2^t (for MAC length t and number of blocks k) as blocks may be modified to make to only change parts of the MAC [Fer05]

SpongeWrap

- By using SHA-3 it is also possible to implement an AEAD construct [BDP11a]
- Construction is very simple and comparably easy to understand
- Uses so-called *duplex mode* for sponge functions, where data write and read operations are interleaved
- Does not require padding of data to a specific block size
- Cannot be parallelized

- Security:
  - Not widely used yet, but several aspects proven to be as secure as SHA-3 in standardized mode
  - If the authenticated data A does not contain a unique IV the same key stream will be generated (allows the recovery of one block of XORed encrypted data)

SpongeWrap - Operation



- Simplified version, where key and MAC length must be smaller than block-size
- Paddings with a single “0” or “1” bit ensure that different data blocks types are well separated

Additional References

[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.

[Men97a] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone. *Handbook of Applied Cryptography*, CRC Press Series on Discrete Mathematics and Its Applications, Hardcover, 816 pages, CRC Press, 1997.

[NIST02] National Institute of Standards and Technology (NIST). *Secure Hash Standard.* Federal Information Processing Standards Publication (FIPS PUB), 180-2, 2002.

[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.

[WYY05a] X. Wang, Y. L. Yin, H. Yu. *Finding collisions in the full SHA-1.* In Advances in Cryptology - CRYPTO'05, pages 18-36, 2005.

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

Additional References

[WLYF04] X. Wang, D. Feng, X. Lai, H. Yu. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD.* IACR Eprint archive, 2004.

[LWW05] A. Lenstra, X. Wang, B. de Weger. *Colliding X.509 Certificates.* Cryptology ePrint Archive: Report 2005/067. 2005.

[LD05] S. Lucks, M. Daum. *The Story of Alice and her Boss.* In Rump session of Eurocrypt'05. 2005.

[Kl06] V. Klima. Tunnels in Hash Functions: *MD5 Collisions Within a Minute* (extended abstract), Cryptology ePrint Archive: Report 2006/105, 2006.

[SA09] Y. Sasaki, K. Aoki. *Finding Preimages in Full MD5 Faster Than Exhaustive Search.* Advances in Cryptology – EUROCRYPT'09. 2009

[Man11] M. Manuel. Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1. Journal Designs, Codes and Cryptography. Volume 59, Issue 1-3, pages 247-263, 2011.

[GH04] H. Gilbert, H. Handschuh. *Security Analysis of SHA-256 and Sisters*. Lecture Notes in Computer Science, 2004, Volume 3006/2004, pages 175-193. 2004.

[AGM09] K. Aoki, J. Guo, K. Matusiewicz, V. Sasaki, L. Wang. Preimages for Step-Reduced SHA-2. Advances in Cryptology - ASIACRYPT 2009. pages 578-597, 2009.

[KK06] J. Kelsey, T. Kohno. *Herding Hash Functions and the Nostradamus Attack.* Advances in Cryptology – EUROCRYPT'06. 2006.

[Jou04] A. Joux: *Multicollisions in Iterated Hash Functions.* Application to Cascaded Constructions. CRYPTO 2004: pages 306-316. 2004.

Additional References

[MV04] D. McGrew, J. Viega. The *Security and Performance of the Galois/Counter Mode (GCM) of Operation (Full Version)*. http://eprint.iacr.org/2004/193.

[Fer05] N. Ferguson. *Authentication weaknesses in GCM*. 2005.

[Dwo07] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. NIST Special Publication 800-38D. 2007

[Saa11] M. Saarinen. *GCM, GHASH and Weak Keys*. Cryptology ePrint Archive, Report 2011/202, http://eprint.iacr.org/2011/202, 2011.

[BDP07] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. *Sponge Functions.* Ecrypt Hash Workshop 2007.

[BDP11a] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Cryptographic sponge functions. Research report. Version 0.1. 2011

[BDP11b] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. The Keccak reference. Research report. Version 3.0. 2011