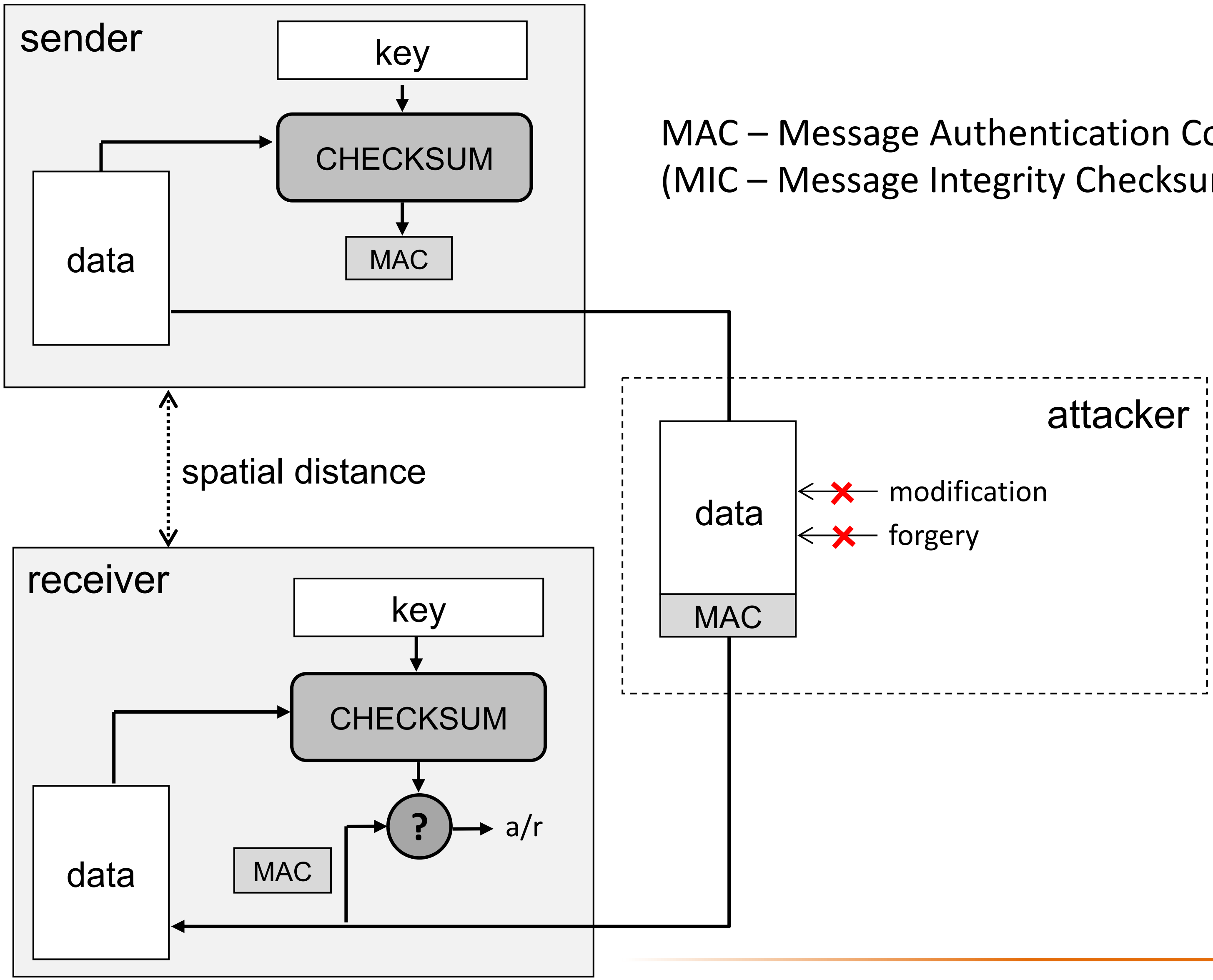
Symmetric Key Message Authentication

Levente Buttyán
CrySyS Lab, BME
buttyan@crysys.hu

Model



MAC – Message Authentication Code
(MIC – Message Integrity Checksum)

Contents

- Cryptographic hash functions
- MAC functions
- Authenticated encryption

Cryptographic hash functions

- a hash function is a function that maps any arbitrary long message into a fixed length output (n bits)

- notation and terminology:
  - x – (input) message
  - y = H(x) – hash value, message digest, fingerprint

- typical applications:
  - the hash value of a message can serve as a compact representative image of the message (similar to fingerprints)
  - increase the efficiency of digital signatures by signing the hash instead of the message (expensive operation is performed on small data)
  - store password hashes instead of cleartext passwords on servers

- examples:
  - (MD5, SHA-1), SHA-2 family, SHA-3 family

Desired properties of hash functions

- ease of computation
  - given an input x, the hash value H(x) of x is easy to compute
- **weak collision resistance** (2^{nd} preimage resistance)
  - given an input x, it is computationally infeasible to find a second input x' such that H(x') = H(x)
- **strong collision resistance** (collision resistance)
  - it is computationally infeasible to find any two distinct inputs x and x' such that H(x) = H(x')
- **preimage resistance** (one-way property)
  - given a hash value y (for which no preimage is known), it is computationally infeasible to find any input x such that H(x) = y

Random function model

- collision resistant hash functions can typically be modeled as a random function (similar to block ciphers)

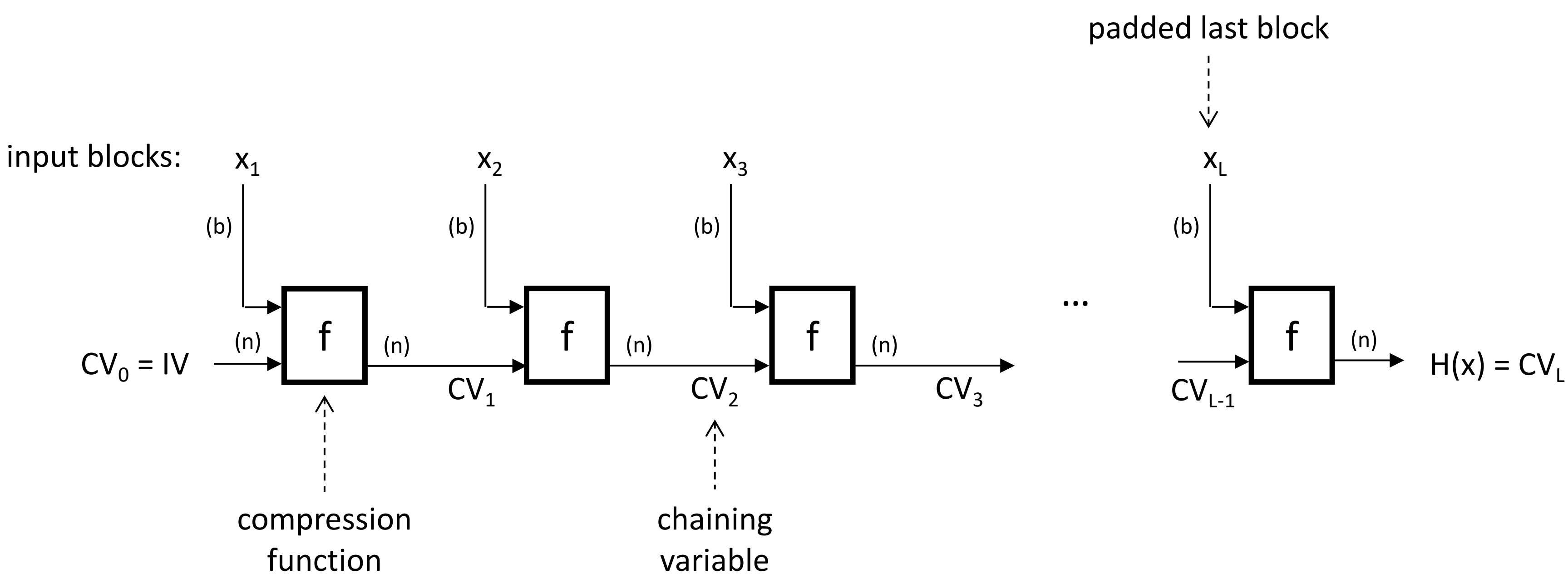
- illustration:
  - SHA1("The quick brown fox jumps over the lazy dog")
    gives hexadecimal: `2fd4e1c67a2d28fced849ee1bb76e7391b93eb12`
  - SHA1("The quick brown fox jumps over the lazy cog")
    gives hexadecimal: `de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3`
  - the second hash differs from the first one in 81 (out of 160) bits (**avalanche effect**)

Birthday attack on hash functions

- Birthday paradox
  - when drawing elements randomly (with replacement) from a set of N elements, a repeated element will be encountered with high probability after ~sqrt(N) selections
- Birthday attack on hash functions
  - brute force attack (similar to exhaustive key search in case of ciphers)
  - generate inputs randomly and hash them
  - after about ~sqrt(2^n) = $2^{n/2}$ randomly chosen inputs, a collision pair will occur with high probability
- in order to resist birthday attacks, n/2 should be large enough
- <u>note</u>: it is easier to find collisions than to find preimages or 2nd preimages for a given hash value (which have complexity 2^n)

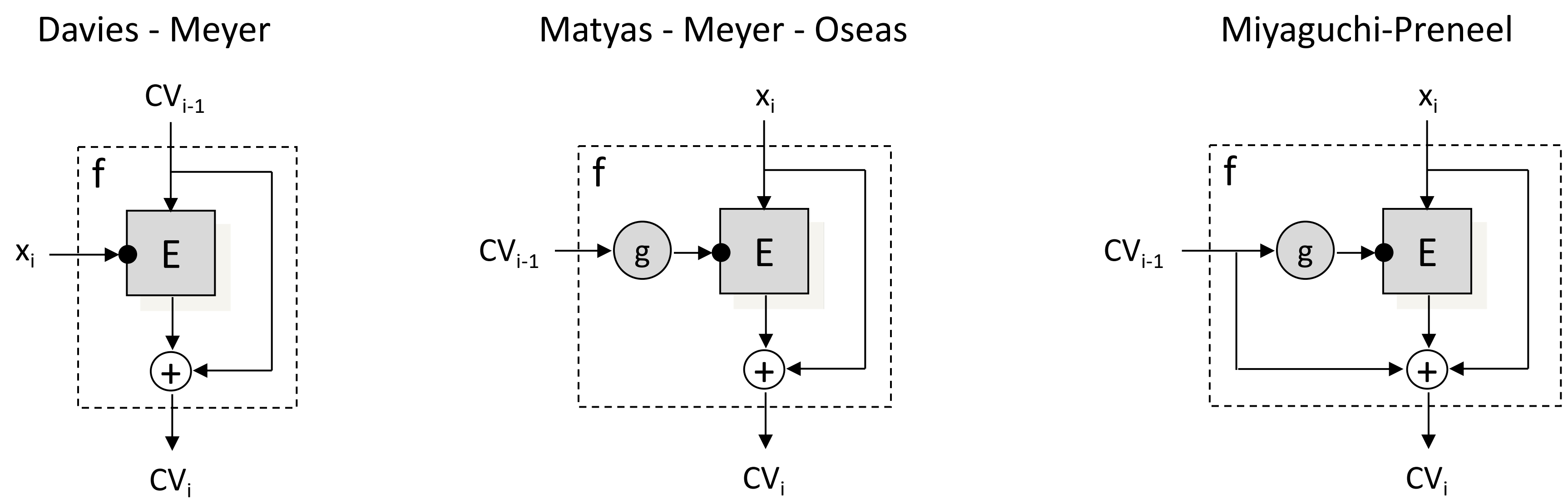
Iterative hash functions

- operation:
  - input is divided into fixed length blocks
  - last block is padded if necessary
  - each input block is processed according to the following scheme:



- example: MD5, SHA-1, SHA-2

Block cipher based compression functions

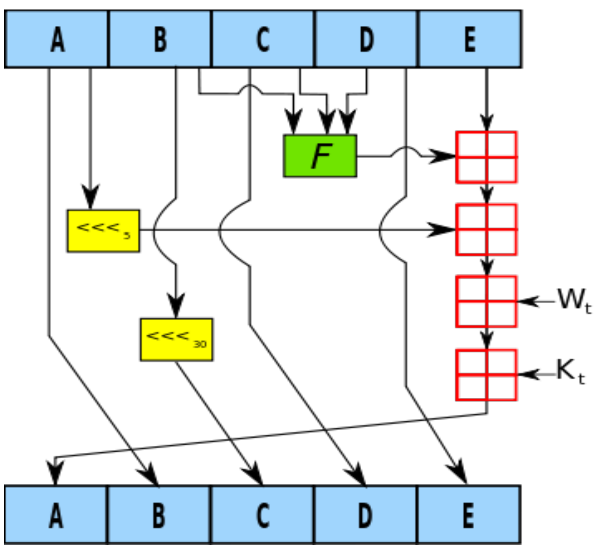


<u>a potential problem:</u>

the hash size is equal to the block size of the cipher, which is in practice not sufficiently large (e.g., 128 bits) → vulnerable to the birthday attack

SHA-1

- SHA stands for Secure Hash Algorithm
- SHA-1 is an iterative hash function with a hash size of 160 bits and input block size of 512 bits
- the compression function f consists of 80 iterations of the following computation:

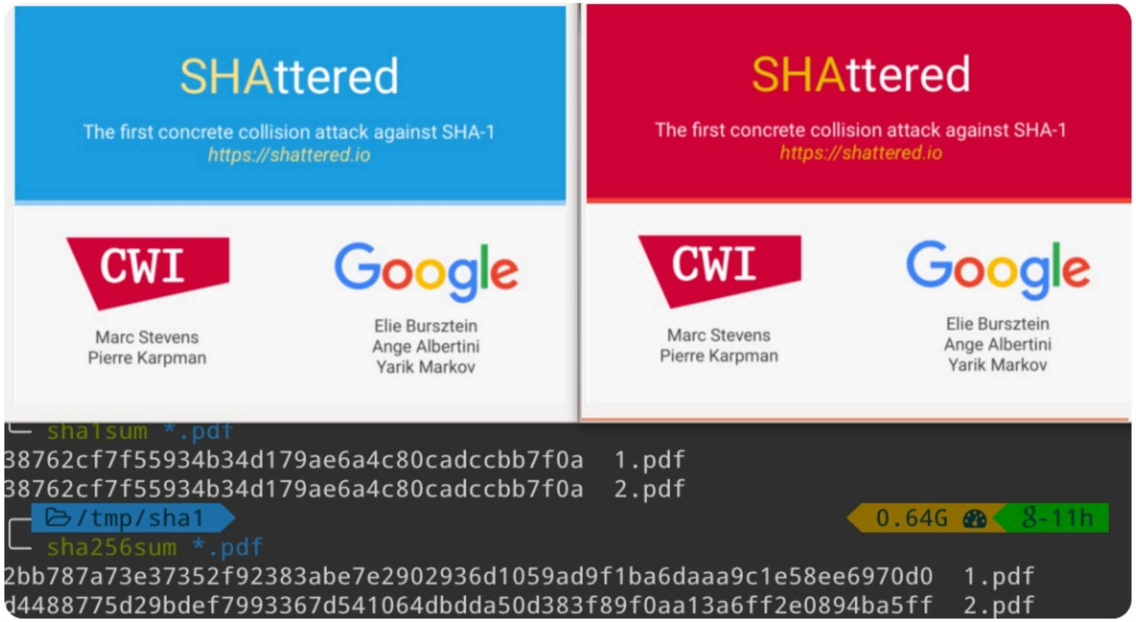


where

- A, B, ..., E are 32-bit words
- F is a nonlinear function that uses logical operations
- W's contain the input block
- K's are constants
- + is addition modulo 2^{32}

SHA-1 has been broken!

- details: https://shattered.io



- attack complexity
  - 9,223,372,036,854,775,808 SHA-1 compressions performed
  - on 110 GPUs for 1 year
  - brute force attack would have needed 12,000,000 GPUs and 1 year

SHA-2

- SHA-2 is a family of iterative hash functions published by NIST in 2001
  - **SHA-256** and **SHA-512** are new hash functions with identical structure but different word size (32 and 64 bits, respectively)
  - **SHA-224** and **SHA-384** are truncated versions of SHA-256 and SHA-512, respectively, and they use different CV_0 values
  - **SHA-512/224** and **SHA-512/256** are also truncated versions of SHA-512, but the CV_0 values are generated using a special method described in the FIPS 180-4 standard
  - numbers in the name specify the output size, e.g., SHA-256 produces 256 bit hash values

- patented by the US, but can be used under royalty-free license
- more info: https://en.wikipedia.org/wiki/SHA-2

SHA-3

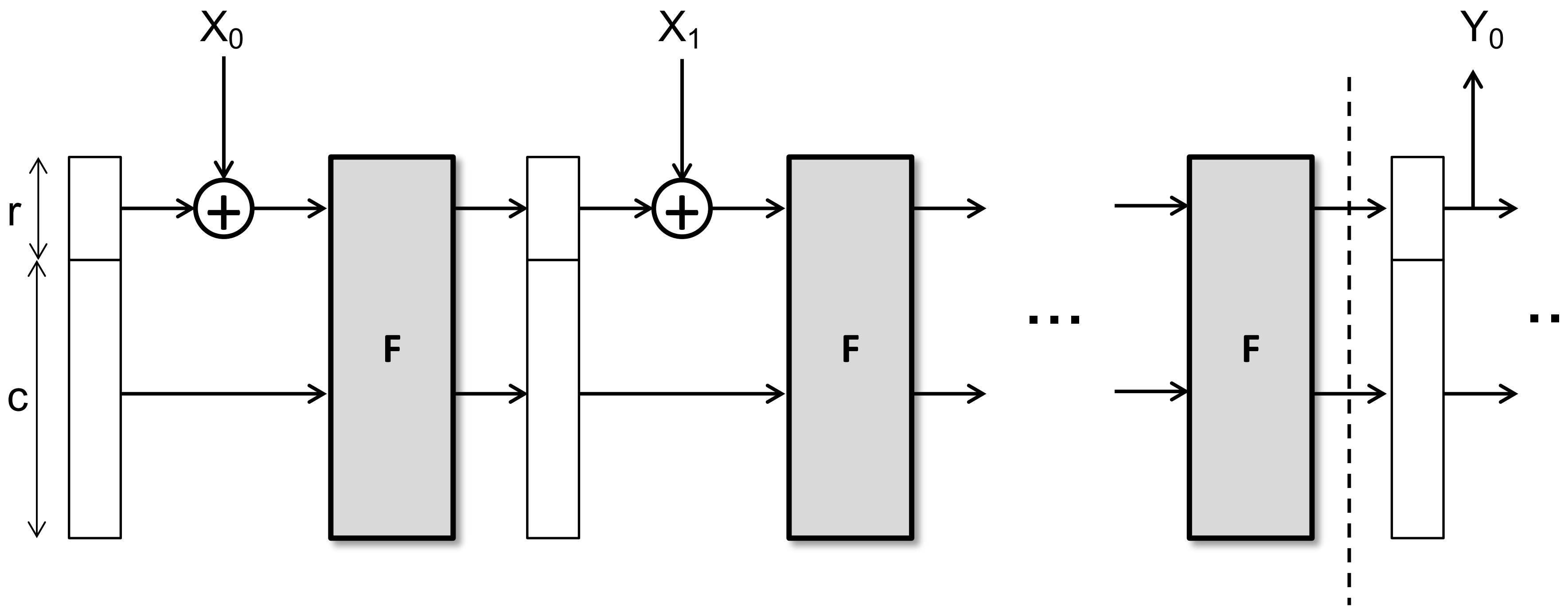
- based on Keccak, which was selected as the winner of the NIST hash function competition in 2012

- not meant to replace SHA-2, but provides an alternative that is dissimilar to SHA-2

- uses the so called ”sponge construction”
  - message blocks are XORed into a subset of the internal state, which is then permuted as a whole

- SHA-3 parameter sizes:
  - internal state (bits): 1600
  - output size (bits): 224 256 384 512
  - input block size (bits): 1152 1088 832 576

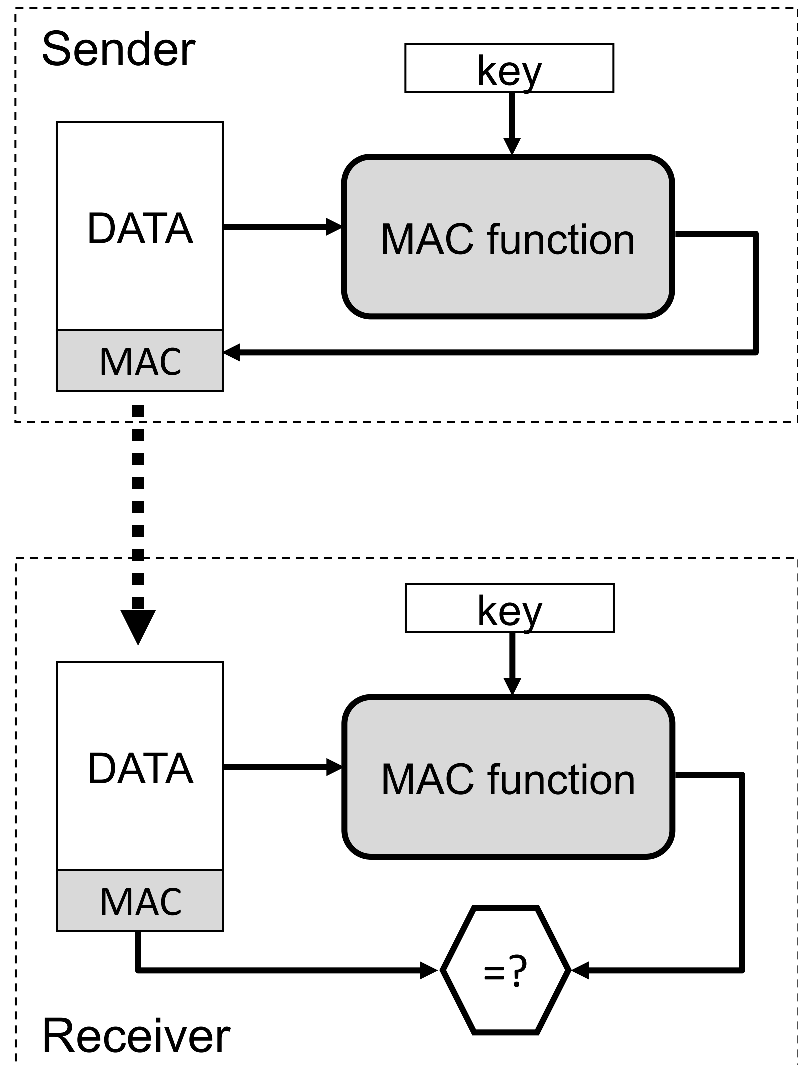
Sponge construction



– F is a permutation (like a block cipher)
– input is porcessed in r bit blocks (r is called ”rate”)
– output is taken from the first r bits of the final state
  » if more output bits are needed, then F is iterated on the state
– number of state bits not touched by input or output is c (c is called ”capacity”)
– for SHA-3, c = 2n, where n is the output hash size
  → rate also depends on the hash size (r = 1600 – 2n)

MAC functions

- a MAC function is a function that maps an arbitrary long message and a key (k bits) into a fixed length output (n bits)
- can be viewed as a hash function with an additional input (the key)
- services:
  - **message authentication and integrity protection:** after successful verification of the MAC value, the receiver is assured that the message has been generated by the sender and it has not been altered in transit
- examples:
  - HMAC, CBC-MAC schemes



Security of MAC functions

- attacker models
  - possible objectives:
    - » forge valid MAC value(s) on a (set of) message(s)
    - » recover the MAC key
  - available knowledge:
    - » known message-MAC pairs
    - » MAC values for (adaptively) chosen messages

- desired MAC function properties
  - key non-recovery
    - » it should be hard to recover the secret key K, given (observed or obtained) one or more message-MAC pairs (m_i, M_i) for that K
  - computation resistance
    - » given (observed or obtained) zero or more message-MAC pairs (m_i, M_i), it is hard to find a valid message-MAC pair (m, M) for any new message $m \neq m_i$
    - » computation resistance implies key non-recovery but the reverse is not true in general

Brute force attacks on MAC functions

- guessing a correct MAC for a given message or a message for a given MAC have probability 2^{-n} (--» complexity 2^n)
  - an important difference between MACs and hash functions is that message-MAC guesses cannot be verified off-line, but one needs access to a MAC verification oracle

- a brute force attack on the key has complexity 2^k
  - we assume, we have message-MAC pairs (m_i, M_i) available
  - for each possible key, compute the MAC value of m_i and compare it to M_i
  - if they don't match, try the next key
  - if they match, try the key on the other pairs too
  - if the key works for every pair, it is very likely that we found the right key

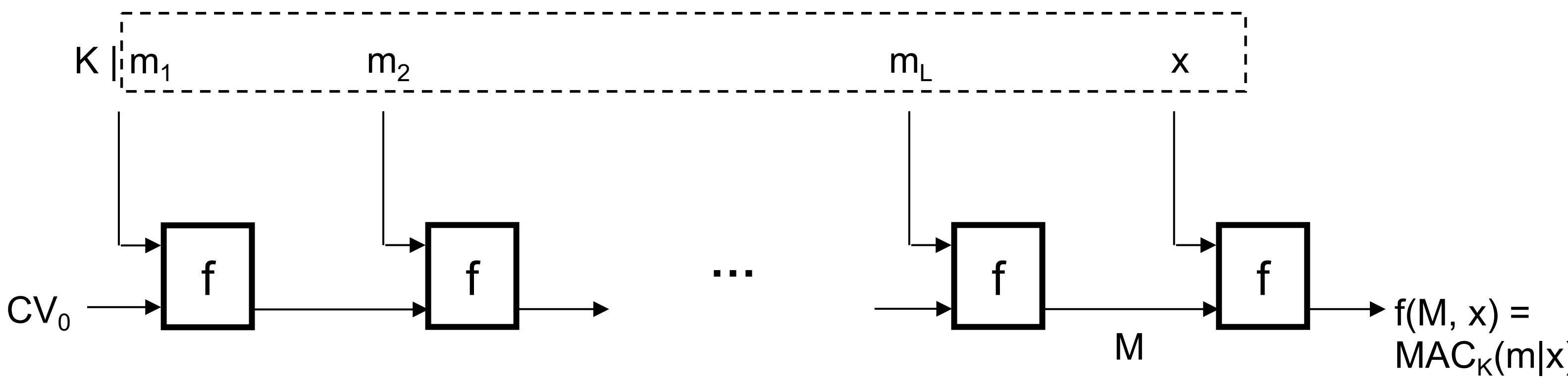
→ min(2^k, 2^n) should be sufficiently large

Meet-in-the-middle attack

- let's assume that the attacker expects a message m* to be sent by a party (e.g., a status message of known content) at the beginning of a communication session
- the attacker pre-computes MAC values of m* with $2^{k/2}$ randomly chosen keys, and stores the results and corresponding keys, i.e., a table of ($MAC_K(m^*)$, K) pairs
- when the party starts the session, it sends $m^* | MAC_{K^*}(m^*)$
- the attacker checks the MAC value againts his pre-computed values:
  - if there's a matching entry ($MAC_{K'}(m^*)$, K'), then it is very likely that K* = K', so the attacker obtained the MAC key
    - he can forge and modify further messages in the session!
  - if no matching entry found, then the attacker waits for another session
- the attacker needs to wait appr. $2^{k/2}$ sessions until he finds a match
- complexity of the attack:
  - computation: $2^{k/2}$, storage: $2^{k/2}$, time: $2^{k/2}$
  - $2^{k/2} << 2^k$

Naïve MAC constructions

- secret prefix method: $MAC_K(m) = H(K|m)$
  - insecure !
    - » assume an attacker knows the MAC on m: $M = H(K|m)$
    - » he can produce the MAC on $m|x$ as $M' = f(M, x)$, where f is the compression function of H



- a similar mistake: $MAC_K(m) = H_K(m)$ (where $H_K(.)$ is $H(.)$ with $CV_0 = K$)
  - insecure !

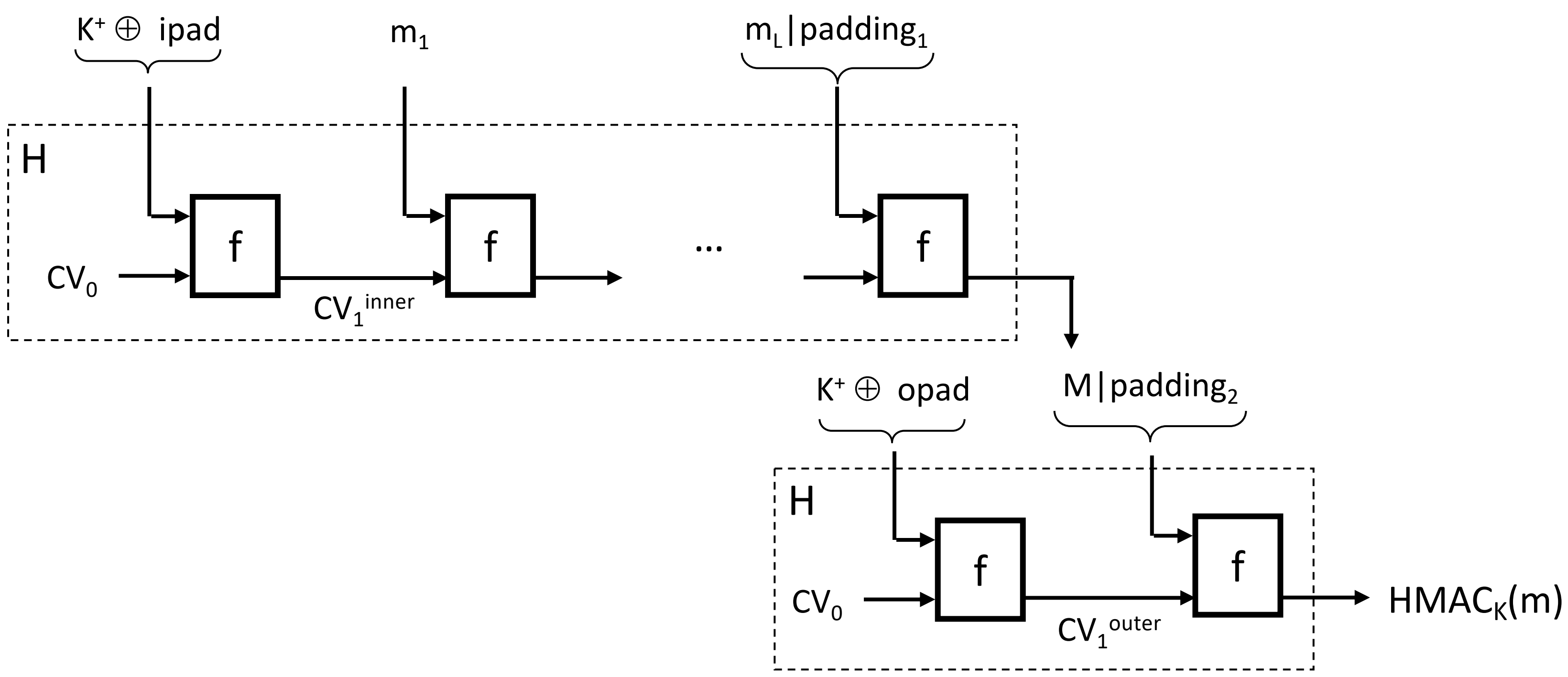
Naïve MAC constructions

- encrypted hash: $MAC_K(m) = E_K(H(m))$
  - not recommended to use !
    - » two messages having the same hash value will have the same MAC value under all keys
    - » off-line search for messages with colliding MAC values is possible without knowledge of the key → H must be collision resistant !
    - » collision resistant hash functions usually have larger output size than the block size of the block cipher → which mode to use to encrypt the hash?

HMAC

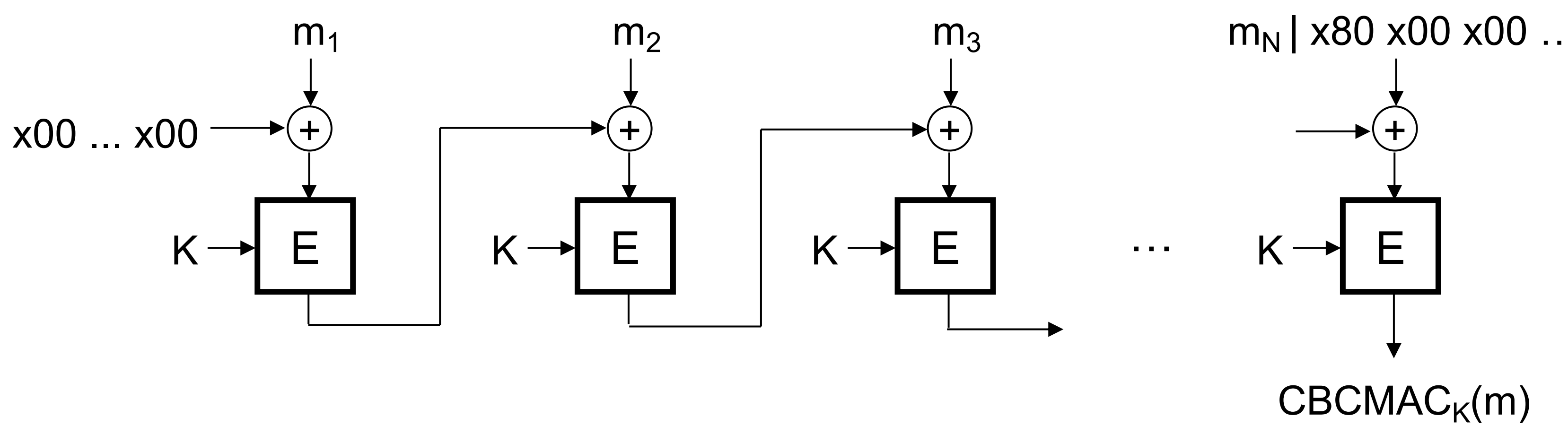
$$HMAC_K(m) = H(\ (K^+ \oplus\ opad)\ |\ H(\ (K^+ \oplus\ ipad)\ |\ m\ )\ )$$

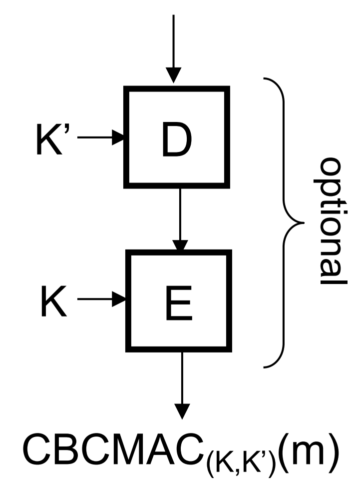
- H is an iterative hash function with input block size b and output size n
- K^+ is K padded with 0s to obtain a length of b bits
- ipad and opad are constants



more info: https://tools.ietf.org/html/rfc2104

CBC-MAC

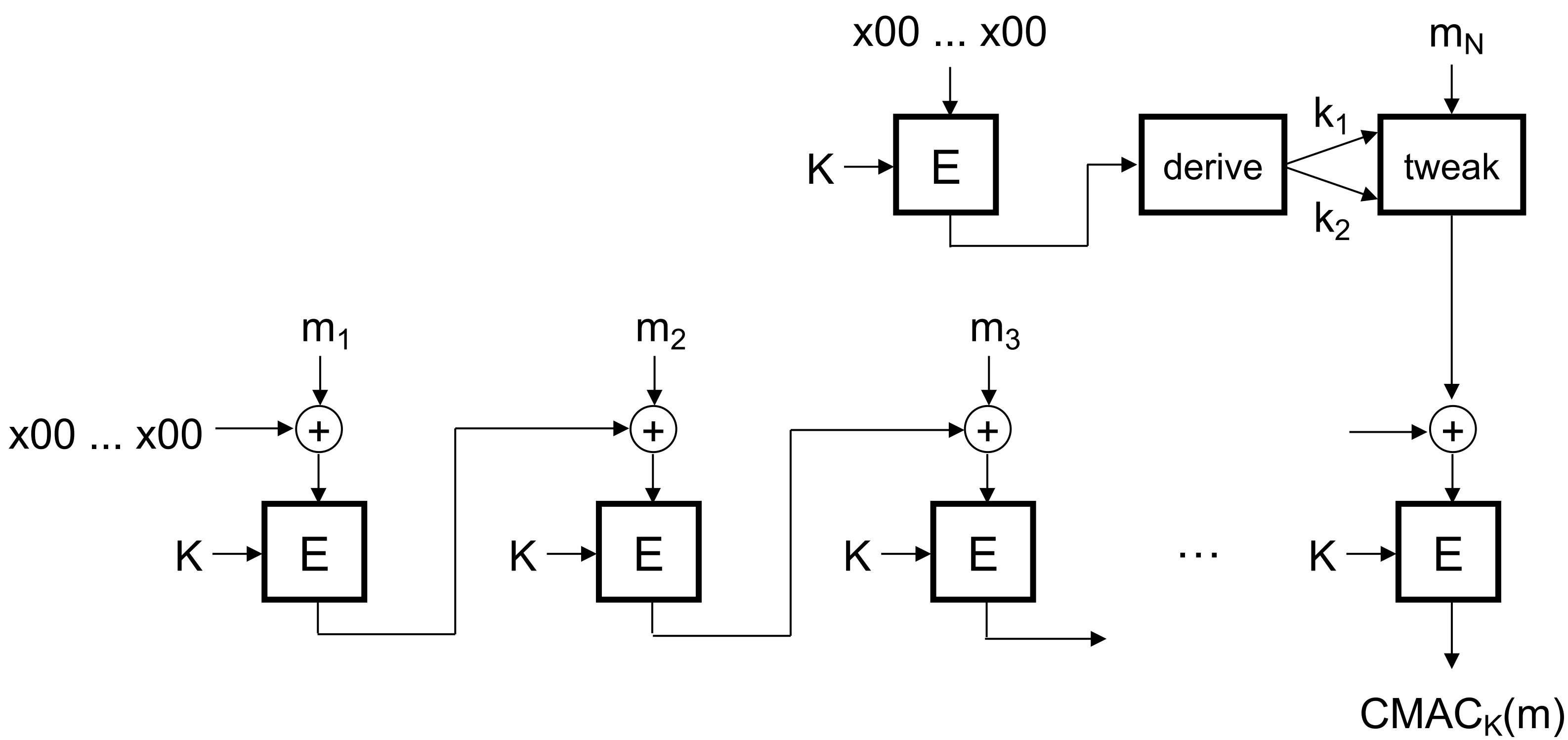




- the basic CBC MAC scheme is vulnerable to forgery attacks
- countermeasures:
  1. use the optional final encryption
     - » increased key length
     - » marginal overhead
  2. prepend to the message a block containing the length of the message before MAC computation
  3. use K to encrypt the message length and use the result as the MAC key

CMAC (Cipher-based MAC)

tweak:
- if m_N is a complete block, then output $m_N + k_1$
- else output $(m_N \,|\, x80\ x00\ x00 \ldots) + k_2$



Authenticated encryption schemes

- simultaneously guarantee the confidentiality, integrity, and authenticity of a message

- approaches:

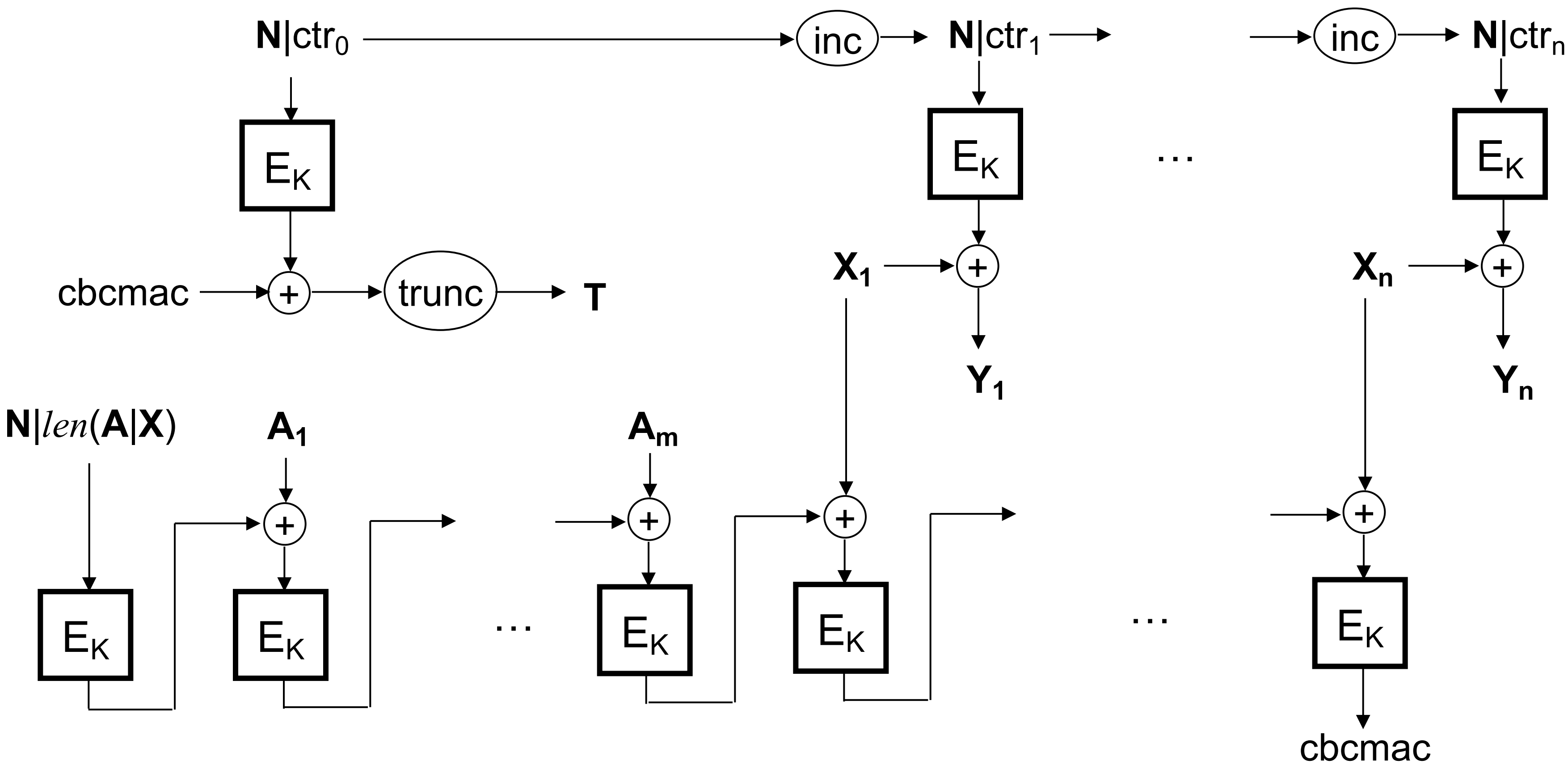
  1. different combinations of an encryption function and a MAC function
     - » use two different keys for the two different functions (encryption and MAC)
     - » always require two passes to process the message (one for MAC computation and one for encryption)
     - » may be vulnerable to some chosen ciphertext attacks (e.g., padding oracle attacks)

  2. specialized authenticated encryption schemes (e.g., CCM, GCM)
     - » use a single key
     - » some schemes process the message in a single pass
     - » even two-pass schemes can be more efficient than generic combinations of MAC and encryption
     - » prevent chosen-ciphertext attacks (e.g., padding oracle attacks)

Generic combinations

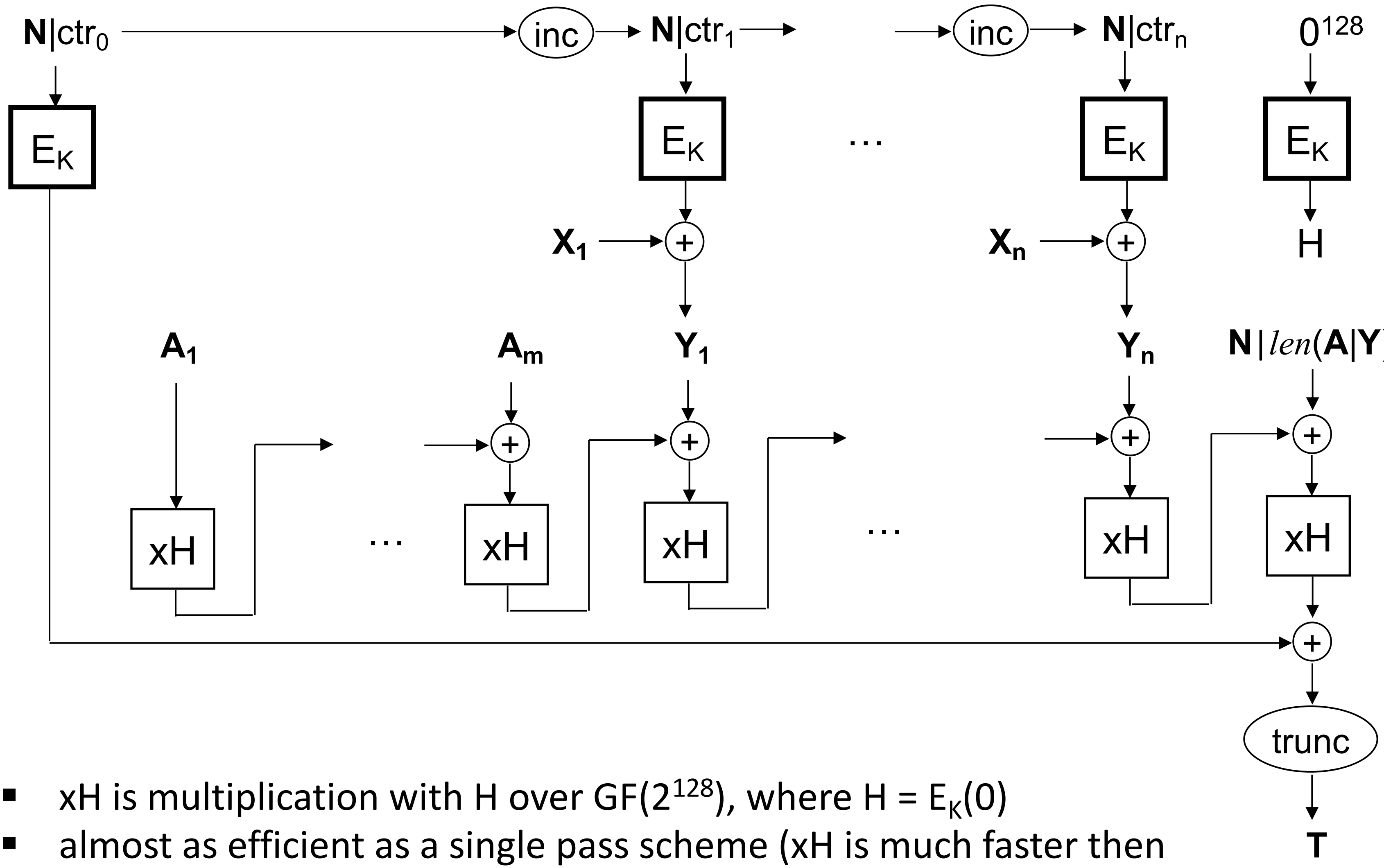
- notation:
  - ENC() denotes any symmetric key encryption (using a stream cipher or a block cipher in any modes)
  - MAC() denotes a MAC function
  - Ke and Km are the encryption and the MAC keys, respectively
- $ENC_{Ke}(m \mid MAC_{Km}(m))$
  - the MAC is verified only after decrypting the message and checking padding
  - if information on correctness of padding is leaked out, then a padding oracle attack is possible
- $ENC_{Ke}(m) \mid MAC_{Km}(m)$
  - the MAC is verified only after decrypting the message and checking padding
  - padding oracle attack may be possible
- $ENC_{Ke}(m) \mid MAC_{Km}(ENC_{Ke}(m))$
  - no decryption and verification of padding before MAC verification
  - chosen ciphertext attacks (including padding oracle attacks) will fail!

CCM – CTR mode with CBC-MAC



- <u>input</u>: message X, key K, associated data A, nonce N
- <u>output</u>: encrypted message Y, authentication tag T (encrypted CBC-MAC value)
- single key, two passes (2n+m+2 invocations of the block cipher)

GCM – Galois/Counter Mode



- xH is multiplication with H over GF(2^{128}), where H = $E_K(0)$
- almost as efficient as a single pass scheme (xH is much faster then encryption E)

Summary

- new model for message authentication (active attacker, message modification and forgery)
- new primitives:
  - (hash functions)
  - MAC functions
  - Authenticated Encryption

- hash functions
  - map arbitrary long inputs to a fixed length output (hash value)
  - non-invertible, collisions are unavoidable
  - 3 important properties: (strong) collision resistance, 2^{nd} preimage resistance (weak collision resistance), preimage resistance (one-wayness)
  - iterative constructions, different families (SHA-2, SHA-3)

Summary

- MAC functions
  - similar to hash functions, but have a key input too
  - attacker model:
    - <u>goals</u>: forge MAC values systematically or recover the MAC key
    - <u>capabilities</u>: observe message-MAC pairs, or (adaptively) chosen message attacks (using a MAC generating oracle)
  - avoid naïve constructions
  - use standards: HMAC, CBC-MAC (with caution), CMAC

- combining encryption and authentication…
  - generic approach of using an ENC and a MAC function in some order
  - special authenticated encryption schemes (e.g., CCM, GCM)

Control questions

- Hash functions
  - What is a cryptographic hash function?
  - What are the 3 main security requirements on crypto hash functions?
  - What is the Birthday Paradox and how is it related to hash functions?
  - How iterative hash functions work? (scheme)
  - Describe the so called ”sponge construction”!

- MAC functions
  - How are MAC functions used to ensure message authentication? (basic operating principle)
  - What attacker models do you know for MAC functions?
  - What are the desired security properties of MAC functions?
  - How do brute force attacks against MAC functions work?
  - What is the meet-in-the-middle attack in the context of MAC functions?

Control questions

  - How does HMAC work? (scheme)
  - How does CBC-MAC work (scheme), and what is its potential problem? How to strengthen the CBC-MAC scheme?
  - How does the CMAC scheme work?

- Authenticated Encryption
  - What is authenticated encryption? What are the 2 main approaches to achieve authenticated encryption?
  - Describe the generic combinations of encryption and MAC! Which one is the most secure?
  - How does the CCM authenticated encryption scheme work?
  - What are the advantages of special authenticated encryption schemes?