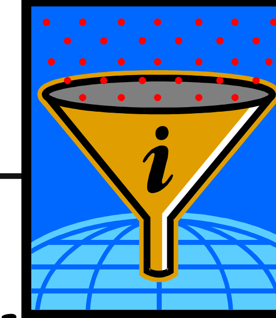
EE817/IS 893

Cryptography Engineering and Cryptocurrency

Yongdae Kim

한국과학기술원

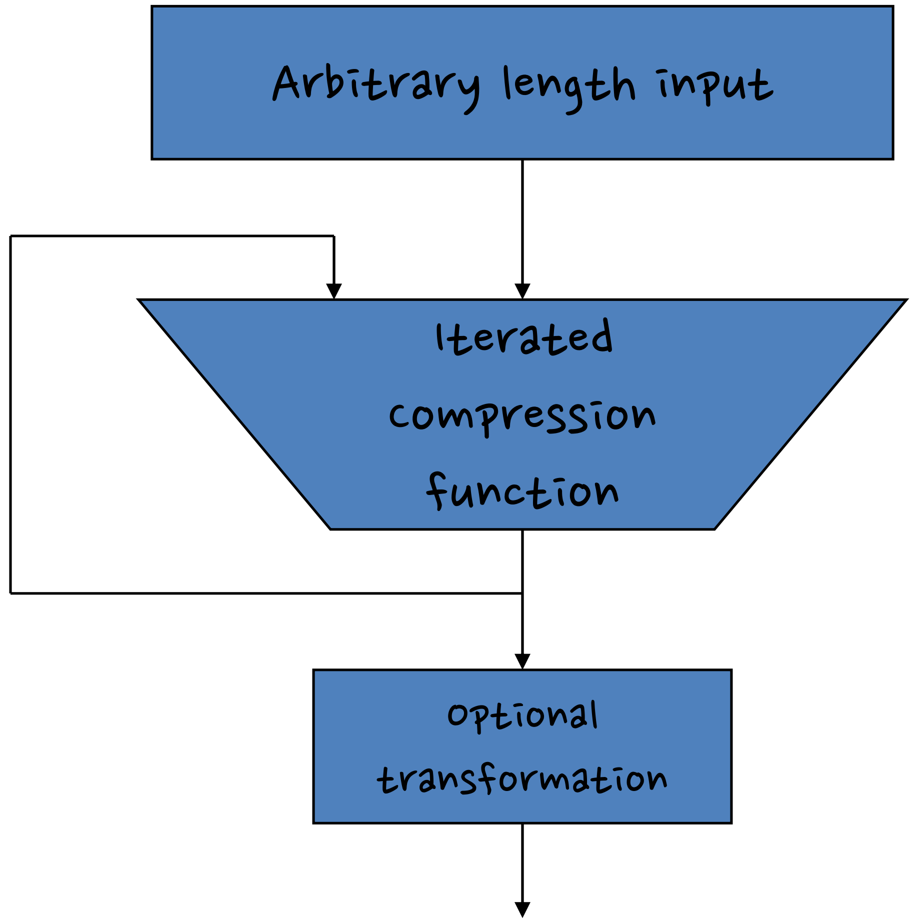
Definition



- A *hash function* is a function h
  - *compression* — h maps an input x of arbitrary finite bitlength, to an output h(x) of fixed bitlength n.
  - *ease of computation* — h(x) is easy to compute for given x and h
- Example: checksum
  - $c_i = \bigoplus_{i=1}^{m} b_{ji}$

    where
  - c_i = i-th bit of hash code
  - m = number of n-bit blocks in the input
  - b_{ij} = i-th bit in j-th block

General Model



MDC h with compression function f:

$H_0 = IV$, $H_i = f(H_{i-1}, x_i)$, $h(x) = H_t$

Basic properties

- *preimage resistance = one-way*
  - it is computationally infeasible to find any input which hashes to that output
  - for a given y, find x' such that h(x') = y
- *2nd-preimage resistance = weak collision resistance*
  - it is computationally infeasible to find any second input which has the same output as any specified input
  - for a given x, find x' such that h(x') = h(x)
- *collision resistance = strong collision resistance*
  - it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output
  - find x and x' such that h(x) = h(x').

Relation between properties

- collision resistance $\Longrightarrow$ weak collision resistance ?
  - Yes! Why?

- collision resistance $\Longrightarrow$ one-way ?
  - No! Why?
  - Let g collision resistant hash function, $g: \{0,1\}^* \longrightarrow \{0,1\}^n$
  - consider the function h defined as

    $h(x) = 1 \,||\, x$ if x has bit length n

    $\quad\quad = 0 \,||\, g(x)$ otherwise

    $h: \{0,1\}^* \longrightarrow \{0,1\}^{n+1}$
  - h(x) : collision and pre-image resistant (unique), but not one-way

Birthday Paradox (1)

- What is the probability that a student in this room has the same birthday as Yongdae?
  - 1/365. Why?
- What is the minimum value of k such that the probability is greater than 0.5 that at least 2 students in a group of k people have the same birthday?
  - $1\ (1 - 1/n)(1 - 2/n)\cdots(1 - (k-1)/n)$

    $\leq e^{-1/n}\, e^{-2/n} \cdots e^{-(k-1)/n}$ $\Leftarrow 1 + x \leq e^x$ Taylor series

    $= e^{-\sum i/n} = e^{-k(k-1)/2n}$

    $\leq 1/2$
  - $-\,k(k-1)/2n \leq \ln (1/2) \Rightarrow k \geq (1 + (1+ (8 \ln 2)\, n)^{1/2}) / 2$
  - For $n = 365$, $k \geq 23$

Birthday Paradox (II)

- Relation to Hash Function?
  - When n-bit hash function has uniformly random output
  - One-wayness: Pr[y = h(x)] ?
  - Weak collision resistance: Pr[h(x) = h(x') for given x] ?
  - Collision resistance: Pr[h(x) = h(x')] ?

What is a hash function?

- Arbitrary length input, fixed length output
- efficient
- one-wayness, 2nd preimage resistance, collision resistance
- What else?

Probability

- Recall that MD5 outputs 128-bit bitstrings.
- What is the probability that MD5("a")=0cc175b9c0f1b6a831c399e269772661?
  - Answer: 1 (I tested it yesterday.)

A random function?

- A hash function is a deterministic function, usually with a published succinct algorithm.
- As soon as Ron Rivest finalized his design, everything is determined and there's nothing really random about it!

Heuristically random?

- But we still regard hash functions more or less 'random'. The intuition is like:
- A hash function 'mixes up' the input too throughly, so for any x, unless you explicitly compute H(x), you have no idea about any bit of H(x) any better than pure guess

Heuristically random?

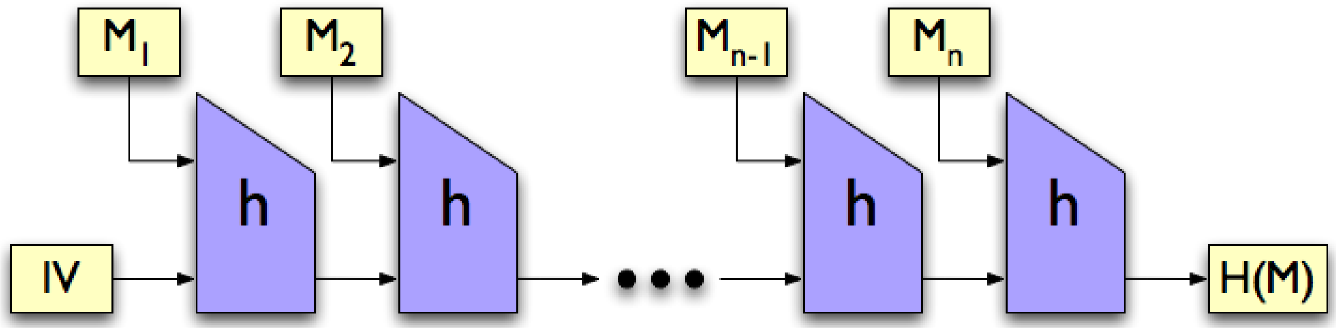
- We want more or less:
  - Even if x & x' are different in 1 bit, H(x) & H(x') should be independent (input is thoroughly mixed)
  - The best way to learn anything about H(x) is to compute H(x) directly
    - Knowing other H(y) doesn't help

How to design a hash function

- Phase 1: Design a 'compression function'
  - Which compresses only a single block of fixed size to a previous state variable
- Phase 2: 'combine' the action of the compression function to process messages of arbitrary lengths
- Similar to the case of encryption schemes

Merkle-Damgard Scheme

- The most popular and straightforward method for combining compression functions



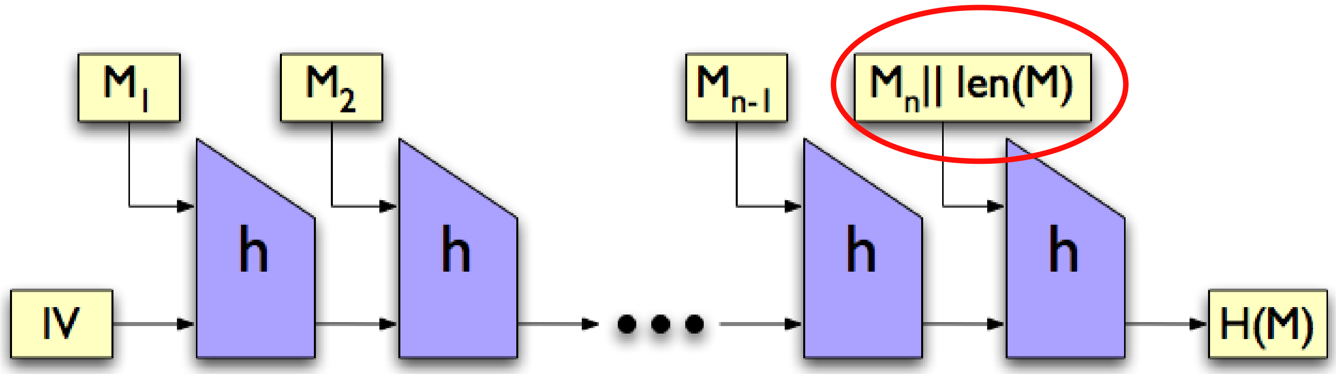
Merkle-Damgard Scheme

- $h(s, x)$: the compression function
  - s: 'state' variable in $\{0,1\}^n$
  - x: 'message block' variable in $\{0,1\}^m$
- $s_0 = IV$, $s_i = h(s_{i-1}, x_i)$
- $H(x_1 || x_2 || \ldots || x_n) = h(h(\ldots h(IV, x_1), x_2) \ldots, x_n) = s_n$

Merkle-Damgard Strengthening

- In the previous version, messages should be of length divisible by m, the block size
  - a padding scheme is needed: $x||p$ for some string p so that $m \mid \mathrm{len}(x||p)$
- Merkle-Damgard strengthening:
  - encode the message length $\mathrm{len}(x)$ into the padding string p

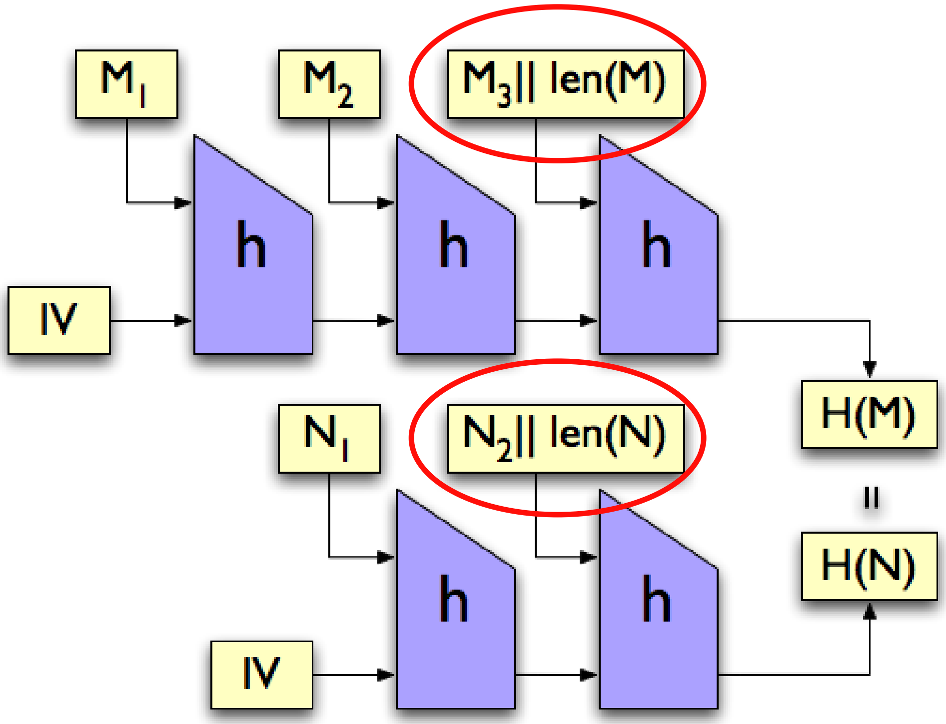
Strengthened Merkle-Damgard



collision resistance

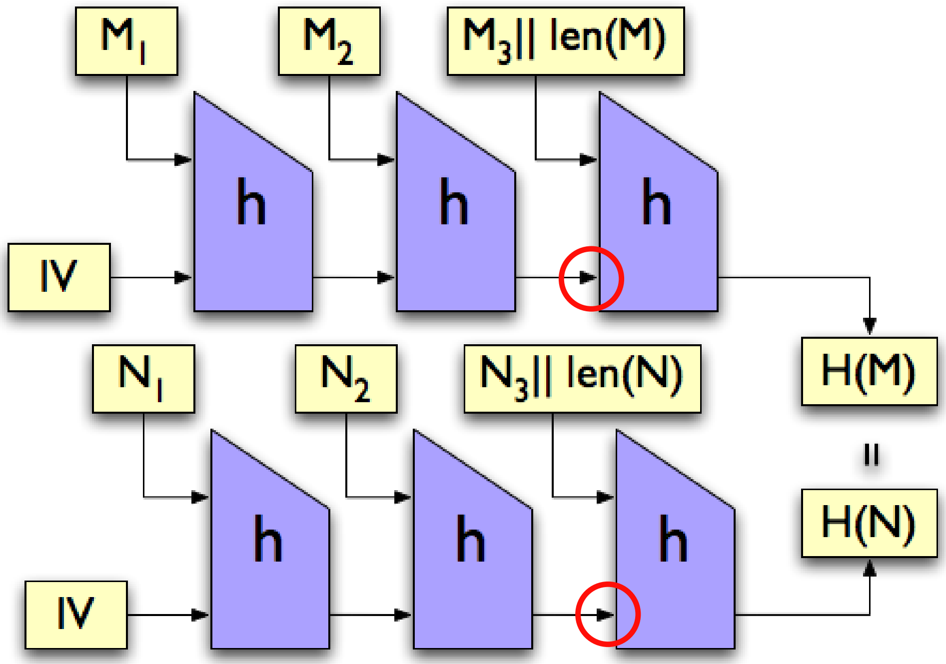
- If the compression function is collision resistant, then strengthened Merkle-Damgard hash function is also collision resistant
- collision of compression function:
  $f(s, x)=f(s', x')$ but $(s, x)\neq(s', x')$

collision resistance

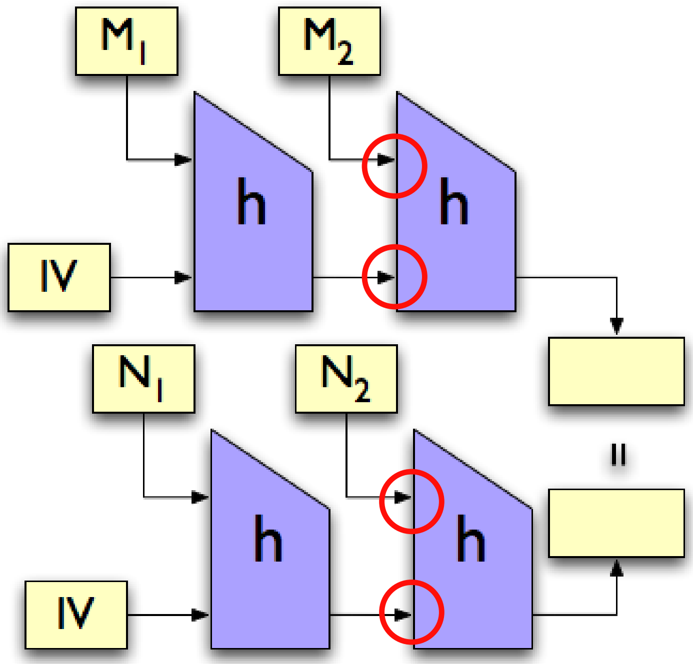


- If h(,) is collision resistant, and if H(M)=H(N), then len(M) should be len(N), and the last blocks should coincide

collision resistance

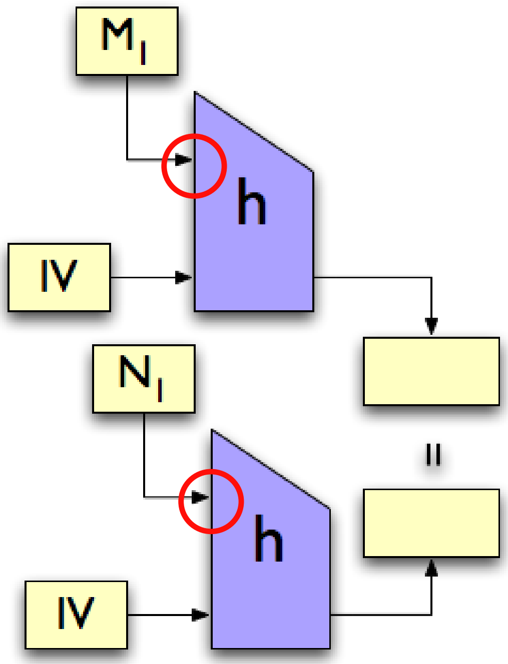


collision resistance



- And the penultimate blocks should agree, and,

collision resistance



- And the ones before the penultimate, too...
- So in fact M=N

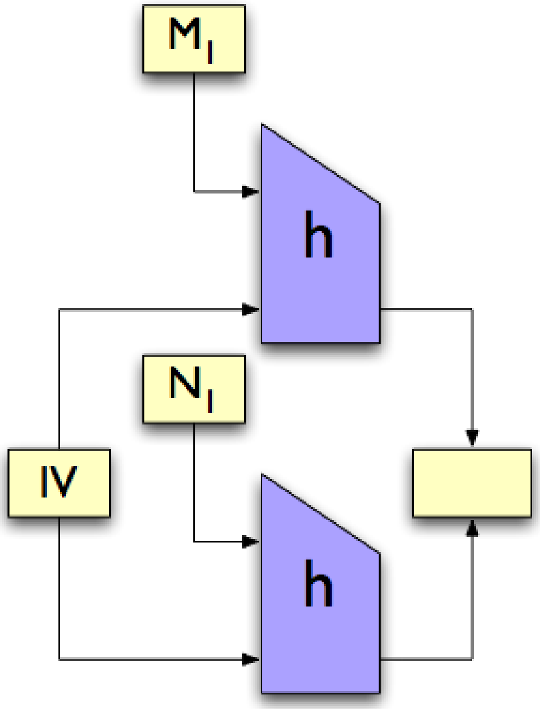
Multicollision

- H: a random function of output size n
- You have to compute about $2^{n/2}$ hash values until finding a collision with high probability
- You have to compute about $2^{n(r-1)/r}$ hash values until finding r-collision with high probability: $H(x_1) = H(x_2) = \ldots = H(x_r)$.

Multicollision attack

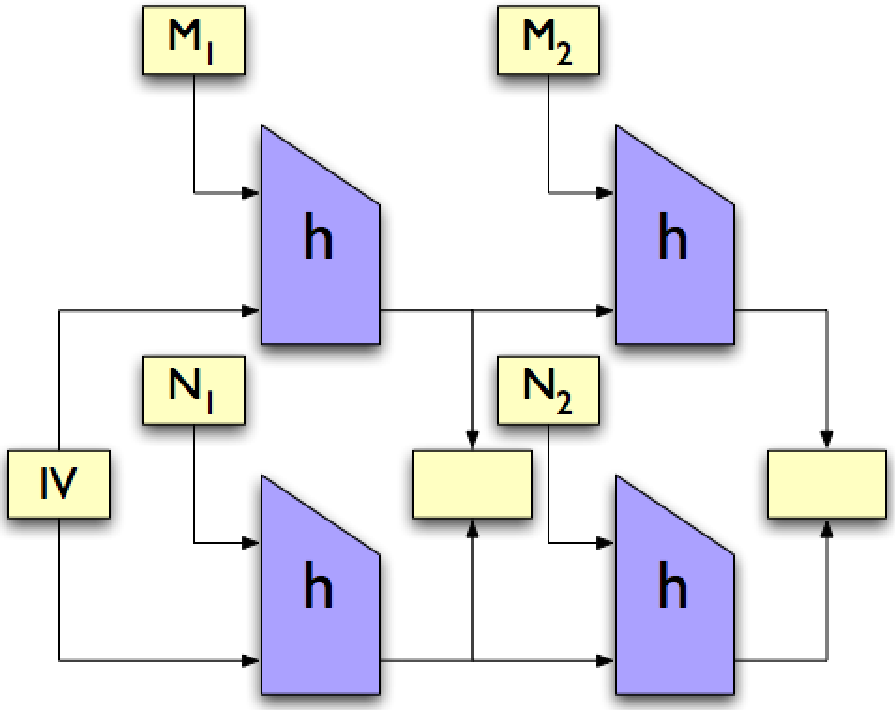
- H: a Merkle-Damgard hash function of output size n (with or without strengthening)
- It is possible to find r-collision about time $\log_2(r)2^{n/2}$, if $r=2^t$ for some t
- By Antoine Joux (2004)

Multicollision attack



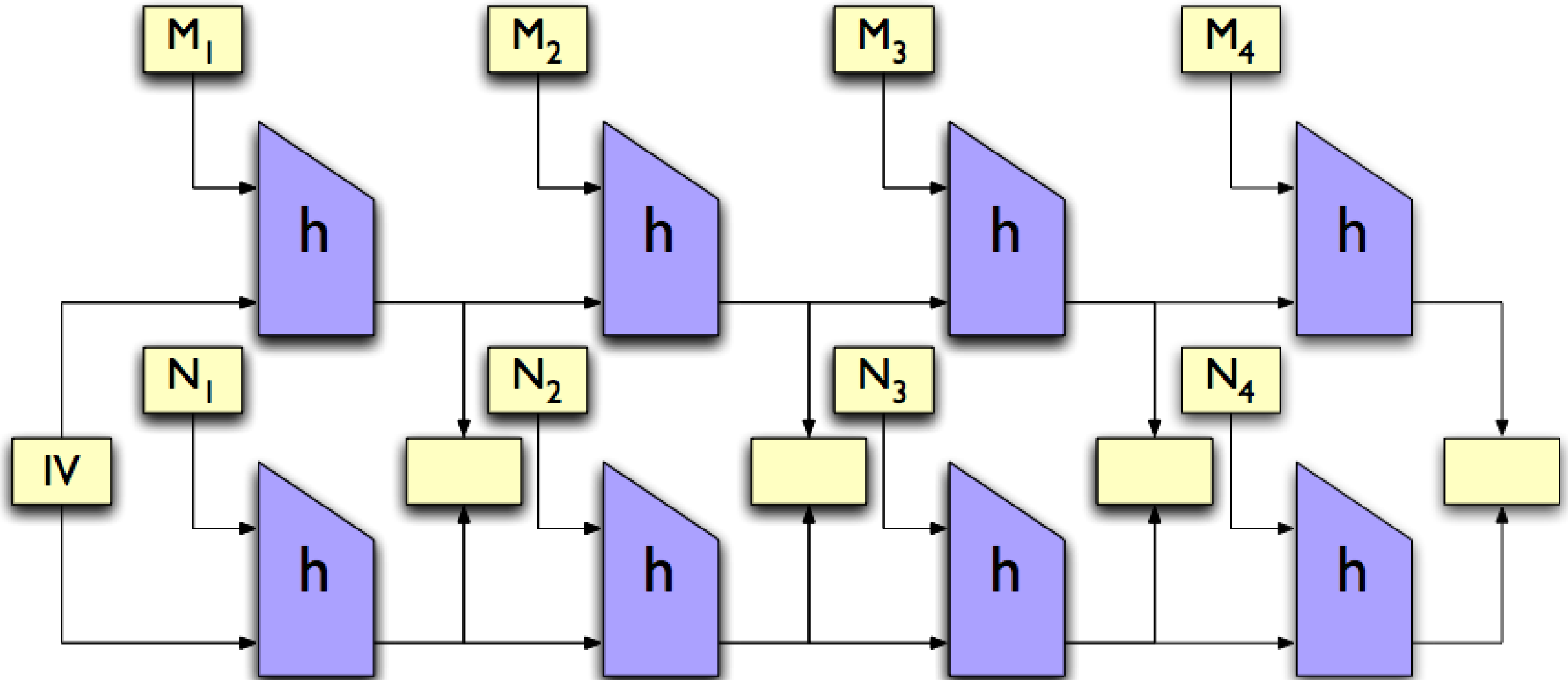
- Do birthday attack to find M_1, N_1 so that $h(IV, M_1) = h(IV, N_1)$

Multicollision attack

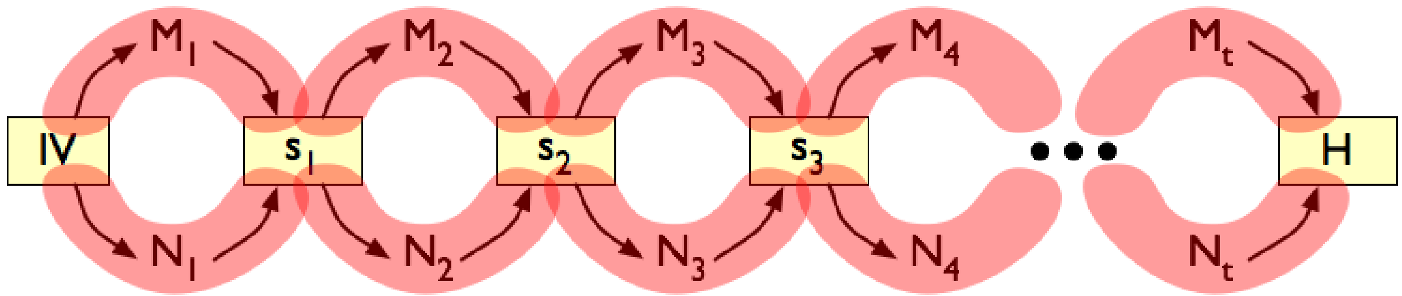


- Starting from the common previous output, do another birthday attack M_2, N_2 so that the next outputs agree

Multicollision attack



Multicollision attack



- Any of the 2^t possible paths all produce the same hash value
- Total workload: $t\ 2^{n/2}$ hash computations (actually compression function computations)

Extension property

- For a Merkle-Damgard hash function,

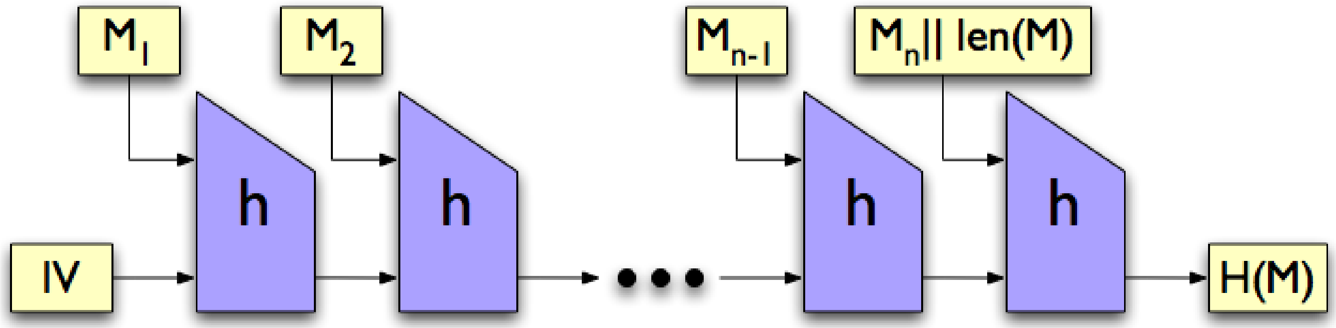
  $H(x, y) = h(H(x), y)$

  - Even if you don't know x, if you know H(x), you can compute H(x, y)
  - H(x, y) and H(x) are *related* by the formula
  - Would this be possible if H() was a random function?

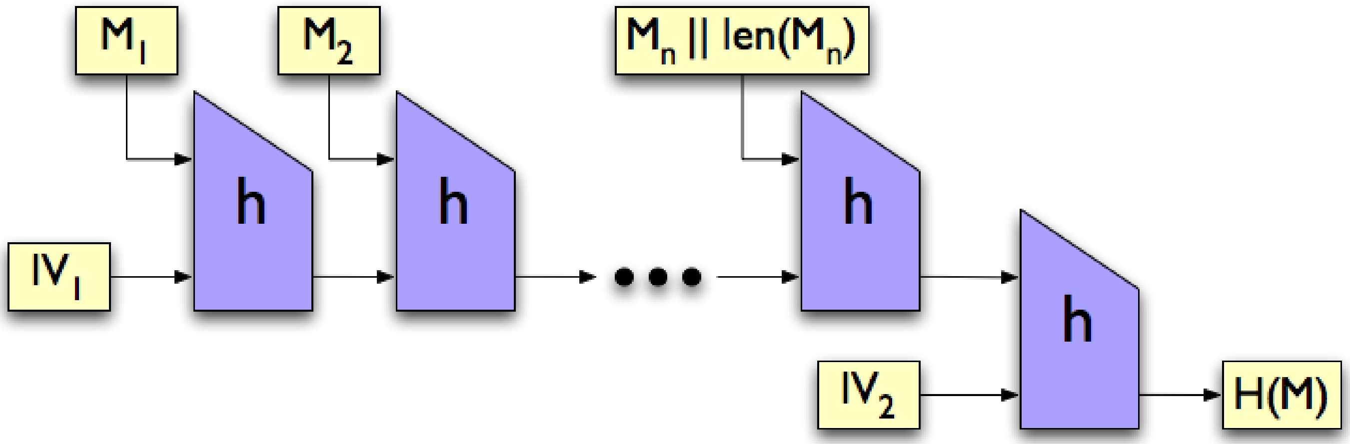
Fixing Merkle-Damgard

- Merkle-Damgard: historically important, still relevant, but likely will not be used in the future (like in SHA-3)
- clearly distinguishable from a random oracle
- How to fix it? Simple: do something completely different in the end

SMD

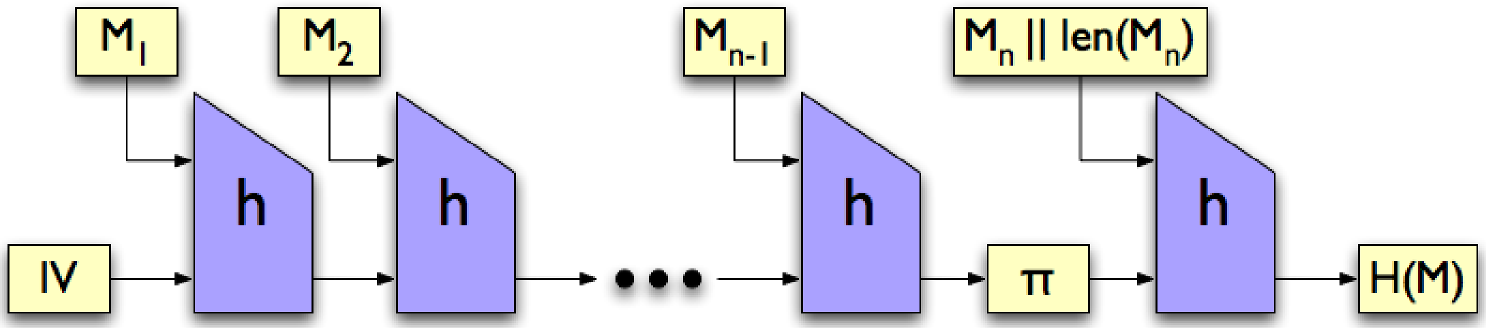


EMD



- $IV_1 \neq IV_2$

MDP



- π: a permutation with few fixed points
  - For example, $\pi(x)=x \oplus c$ for some $c \neq 0$

MAc & AE

MAC

- Message Authentication code
- 'keyed hash function' $H_k(x)$
  - k: secret key, x: message of any length, $H_k(x)$: fixed length (say, 128 bits)
  - deterministic
- Purpose: to 'prove' to someone who has the secret key k, that x is written by someone who also has the secret key k

How to use?

- A & B share a secret key k
- A sends the message x and the MAc $M \leftarrow H_k(x)$
- B receives x and M from A
- B computes $H_k(x)$ with received M
- B checks if $M=H_k(x)$

Attack scenario

- E may eavesdrop many communications (x, M) between A & B
- E then tries (possibly many times) to 'forge' (x', M') so that B accepts: $M' = H_k(x')$
- Question: what if E 'replays' old transmission (x, M)? Is this a successful forgery?

capabilities of attackers

- known-text attack
  - Simple eavesdropping
- chosen-text attack
  - Attacker influences Alice's messages
- Adaptive chosen-text attack
  - Attacker adaptively influences Alice

Types of forgery

- Universal forgery: attacker can forge a MAc for any message
- Selective forgery: attacker can forge a MAc for a message chosen before the attack
- Existential forgery: attacker can forge some message x but in general cannot choose x as he wishes

Security of MAc

- Should be secure against adaptively chosen-message existential forger
  - Attacker may watch many pairs $(x, H_k(x))$
  - May even try x of his choice
  - May try many verification attempts (x, M)
  - Still shouldn't be able to forge a new message at all

Two easy attacks

- Exhaustive key search
  - Given one pair (x, M), try different keys until M=Hk(x)
  - Lesson: key size should be large enough
- Pure guessing: try many different M with a fixed message x
  - Lesson: MAc length should be also large
- Question: which one is more serious?

Random function as MAC

- Suppose A and B share a random function $R(x)$, which assigns random 128-bit value to its input x
- Even if E sees many messages of form $(x, R(x))$, for a new y, $R(y)$ can be any of 2^{128} strings
- Successful forgery prob. $\leq 2^{-128}$

Random function as MAC

- It is a perfect MAC, but the 'key size' is too large: how many functions of form $R: \{0,1\}^m \longrightarrow \{0,1\}^n$? Answer: $2^\wedge(n\, 2^m)$
- But there are keyed functions which are 'indistinguishable' from random functions: called PRFs (PseudoRandom Functions)
- Designing a secure PRF is a good way to design a secure MAC

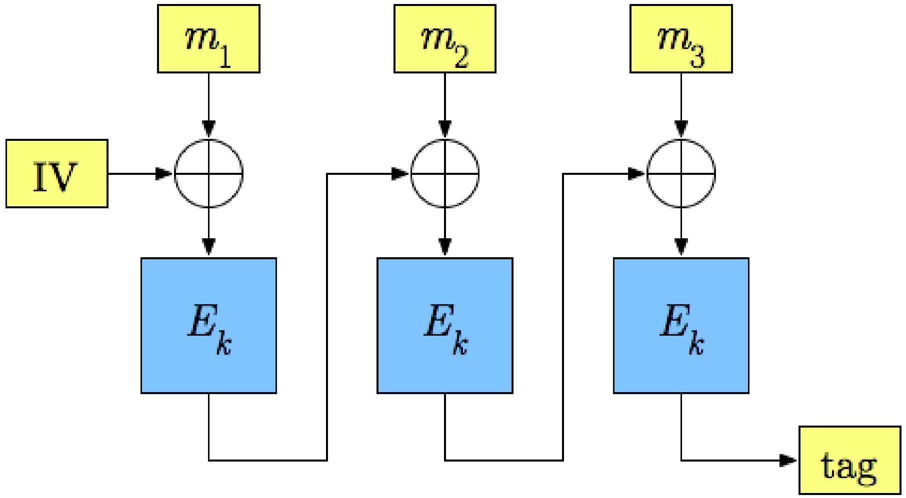
Truncation of MAC

- $H_k(x)$ is a secure MAC with 256-bit output
- $H'_k(x)$ = the first 128 bits of $H_k(x)$
- Question: is $H'_k(x)$ a secure MAC?
    - Answer: not in general, but secure if $H_k(x)$ is a secure PRF

Practical constructions

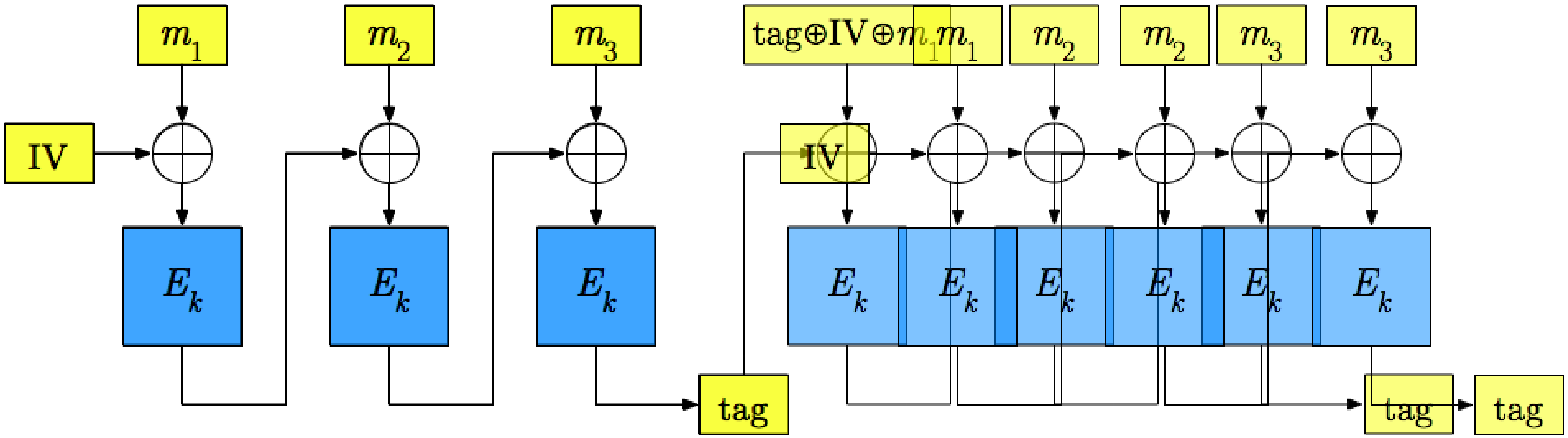
- Blockcipher based MACs
  - CBC-MAC
  - CMAC
- Hash function based MACs
  - secret prefix, secret suffix, envelop
  - HMAC

CBC-MAC



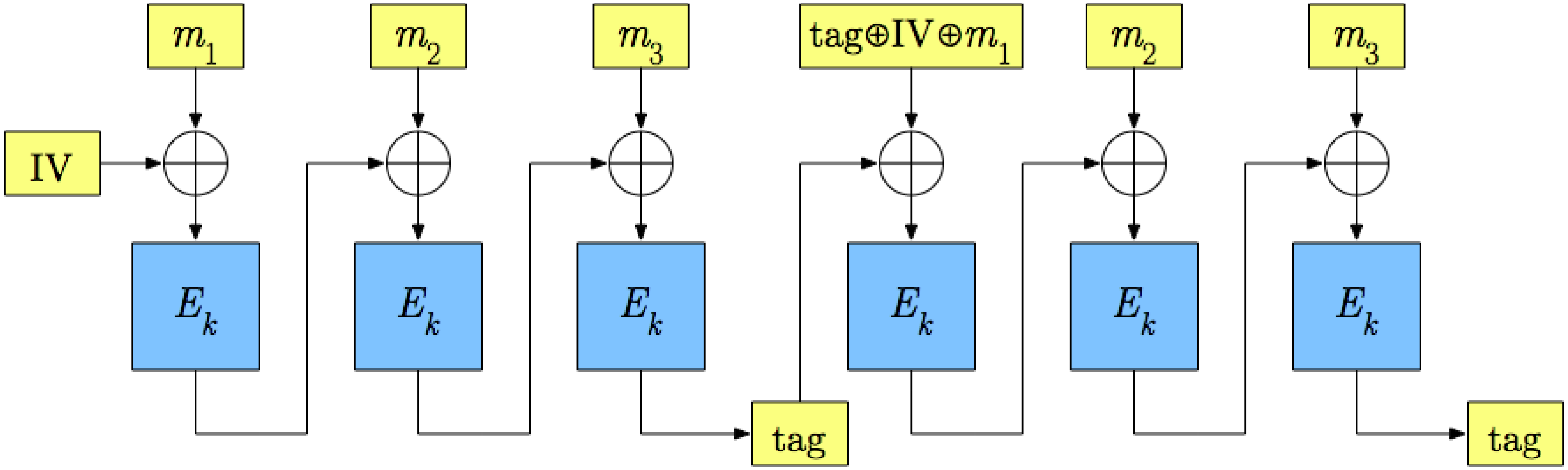
- CBC, with some fixed IV. Last 'ciphertext' is the MAC
- Block ciphers are already PRFs. CBC-MAC is just a way to combine them
- Secure as PRF, if message length is fixed

CBC-MAC



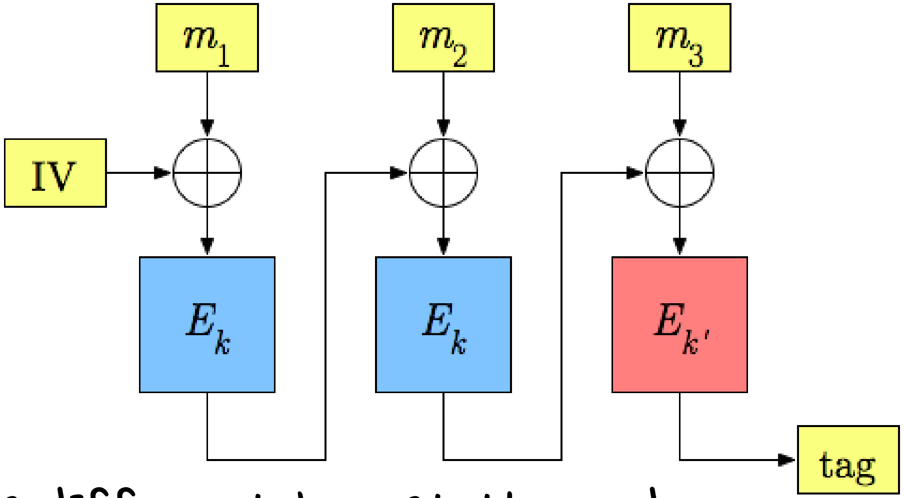
- Secure as PRF, if message length is fixed
- completely insecure if the length is variable!!!

CBC-MAC



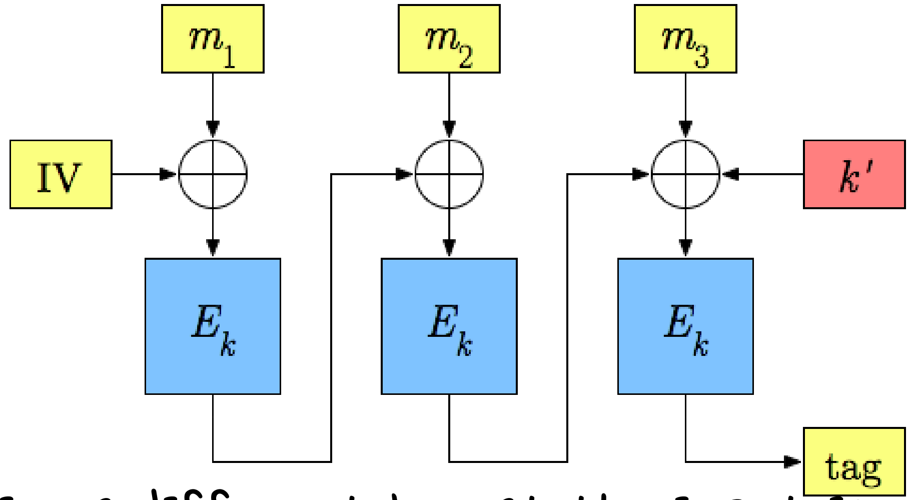
- 'Extension property' once more!
- How to fix it?
  - Again, do something different at the end to break the chain

Modification 1



- Use a different key at the end
- Good: this solves the problem
- Bad: switching block cipher key is bad

Modification 2



- XORing a different key at the input is indistinguishable from switching the block cipher key

CMAC

- NIST standard (2005)
- Solves two shortcomings of CBC-MAC
  - variable length support
  - message length doesn't have to be multiple of the blockcipher size

Some Hash-based MACs

- Secret prefix method: $H_k(x)=H(k, x)$
- Secret suffix method: $H_k(x)=H(x, k)$
- Envelope method with padding:

  $H_k(x)=H(k, p, x, k)$

Secret prefix method

- Secret prefix method: $H_k(x)=H(k, x)$
  - Secure if H is a random function
  - Insecure if H is a Merkle-Damgard hash function
    - $H_k(x, y)=h(H(k, x), y)=h(H_k(x), y)$

Secret suffix method

- Secret suffix method: $H_k(x)=H(x, k)$
  - Much securer than secret prefix, even if H is Merkle-Damgard
  - An attack of complexity $2^{n/2}$ exists:
    - Assume that H is Merkle-Damgard
    - Find hash collision $H(x)=H(y)$
    - $H_k(x) = h(H(x), k) = h(H(y), k) = H_k(y)$
    - off-line!

Envelope method

- Envelope method with padding:

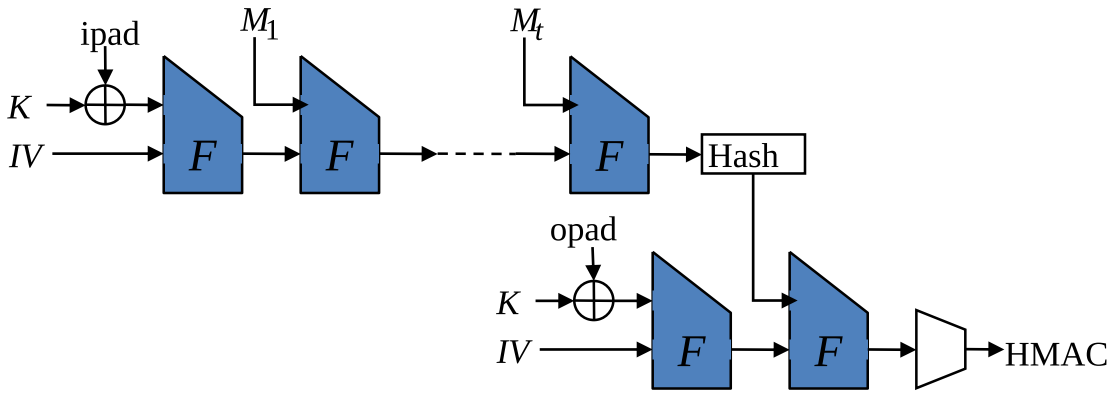
  $H_k(x)=H(k, p, x, k)$

  - For some padding p to make k||p at least one block

- Prevents both attacks

HMAC

- NIST standard (2002)
- $HMAC_k(x) = H(K \oplus opad \,||\, H(K \oplus ipad \,||\, x))$
- Proven secure as PRF, if the compression function h of H satisfies some properties



MAC vs Signature

- secret key vs. public key
- private verification vs. public verification
- MAC doesn't provide non-repudiation
  - Bob claims that Alice sends (x, M), showing that $M = H_k(x)$. Who else can write this message?

confidentiality & integrity

- Two symmetric key primitives
  - Encryption scheme: protects confidentiality
  - MAC: protects integrity
- Usually, what we want is to protect both

Encryption not enough?

- ‘It’s encrypted so nobody can alter it!’
- $c=E_k(P)$
- If any string is a valid ciphertext (e.g., a blockcipher), modifying c to c’ will alter your P (to P’, perhaps a garbage)
  - Question: is this a problem?

Giving redundancy

- Solution: not all strings are valid ciphertext
  - Format plaintext with some redundancy
  - Only correctly formatted plaintext is to be accepted
  - Example, $c=E_k(P \,||\, P)$, or $c=E_k(P \,||\, H(P))$
  - Be careful: what if $E_k()$ is a stream cipher?

Generic composition

- Instead of using an ad-hoc method,
- combine a secure encryption scheme (say, CBC, CTR) and a secure MAC (say, CMAC, HMAC)
  - Two keys are needed
  - How to combine two?
  - 'Generic' here means 'black-box'

Generic composition

- MAC-and-Encrypt: $E_{ke}(P) \mid\mid M_{km}(P)$
- MAC-then-Encrypt: $E_{ke}(P \mid\mid M_{km}(P))$
- Encrypt-then-MAC: $E_{ke}(P) \mid\mid M_{km}(E_{ke}(P))$

Generic composition

- Encrypt-then-MAC: $E_{ke}(P) \| M_{km}(E_{ke}(P))$
  - Most 'unintuitive', in a sense. Handbook gives mild criticism to this
  - Actually, proven to be most secure

Encrypt-then-MAc

- Encrypt-then-MAc: $E_{ke}(P) \mid\mid M_{km}(E_{ke}(P))$
- If the encryption scheme is secure against *chosen plaintext attack*, and MAc is secure, then the composition is secure against *chosen ciphertext attack*, and protects integrity of ciphertext

upgrade!

The other two

- MAC-and-Encrypt: $E_{ke}(P) \,||\, M_{km}(P)$
  - Protects integrity of plaintext, but MAC could leak some information on P
  - How?

The other two

- MAC-and-Encrypt: $E_{ke}(P) \| M_{km}(P)$
  - Protects integrity of plaintext, but MAC could leak some information on P
  - How?
    - What if $M_{km}(P) = P \| M'_{km}(P)$?

The other two

- MAC-then-Encrypt: $E_{ke}(P \,||\, M_{km}(P))$
  - Protects integrity of plaintext, and confidentiality against chosen plaintext attack
  - No problem, but no upgrade

Authenticated Encryption

- Shortcomings of generic composition:
  - Have to manage two keys
  - Takes two passes (one for Enc, one for MAC)
  - correct combination is responsibility of 'users' of the two primitives

Authenticated Encryption

- Authenticated Encryption Scheme
  - Performs both encryption and authentication, with one key
  - Usually comes with security proof
  - Packaged into a single API
  - Potentially, could be done in one-pass
  - Examples: OCB, GCM, …

Questions?

Yongdae Kim

- email: yongdaek@kaist.ac.kr
- Home: http://syssec.kaist.ac.kr/~yongdaek
- Facebook: https://www.facebook.com/y0ngdaek
- Twitter: https://twitter.com/yongdaek
- Google “Yongdae Kim”