Introduction to Cyber Security

Fall 2017 | Sherman Chow | CUHK IERG 4130

Chapter 7
Message Authentication

Message Authentication

- allows communicating parties to verify that received messages are authentic, namely…

- source is authentic: not from masquerading
  - It's from Alice, not from Carol
- contents unaltered: message has not been modified
  - Alice said "Love ya", but not "I hate you"
- timely sequencing: message isn't a replay of a previous one
  - Alice said "Love ya" when she was in kindergarten, not now in Uni.

What is an authentication scheme?

- Can we just use cyclic redundancy check (CRC) code?
  - Or even error-correcting code (ECC)?
- No! Random error vs. Malicious error
  - The adversary can make a valid code according to the algorithm.
- We need a key-ed function (*i.e.*, a function which takes in a key)
  - What key-ed function we have seen so far?
  - *e.g.*, Symmetric-key encryption (but there are other means)
- The message sender and the recipient share the same secret key
- We call it such a primitive "*message authentication code*" (MAC).

What constitutes a MAC scheme?

- Key Generation: KeyGen(*length*) → k
  - Use Internal randomness to generate a key
- Authentication: $\text{MAC}_k(m) \rightarrow t$
- Verification: $\text{Ver}_k\,(m, t) \rightarrow \{0\ (\text{invalid}), 1\ (\text{valid})\}$

- Message m is sent along with the MAC / tag t
  - For deterministic MAC, verification re-creates t and check if equal

How to construct a MAC scheme?

- Can we just use symmetric-key encryption (SKE) as is?
- Intuition: Decryption gives "garbage" → message altered
  - But what is "garbage"?
  - It requires the recipient to know a "correct" plaintext format
    - *e.g.*, English, checksum included for "random-looking" binary data
  - Does not work in general
- SKE does not provide authenticity, but it can be a building block

Forgery Types and Attack Types

- Universal Forgery: the attacker can forge a MAC on everything
- Existential Forgery: the attacker outputs a message he can do

- Known message attack: see some pairs of message and tag
- Chosen message attack: the attacker can choose some messages and see the corresponding tags

What is a “meaningful” forgery?

- In chosen message attack, the attacker can choose some messages and see the corresponding tags
- Just returning any message above and the corresponding tag is easy, and of course, doesn’t count as a “forgery”
- How about I see the tag for “Yes”, but I can forge the tag for “em… Yes”? (Of course, without asking for the tag for the latter)
- Yes! As long as the message is different, we consider it as a meaningful forgery.

Replay Attack

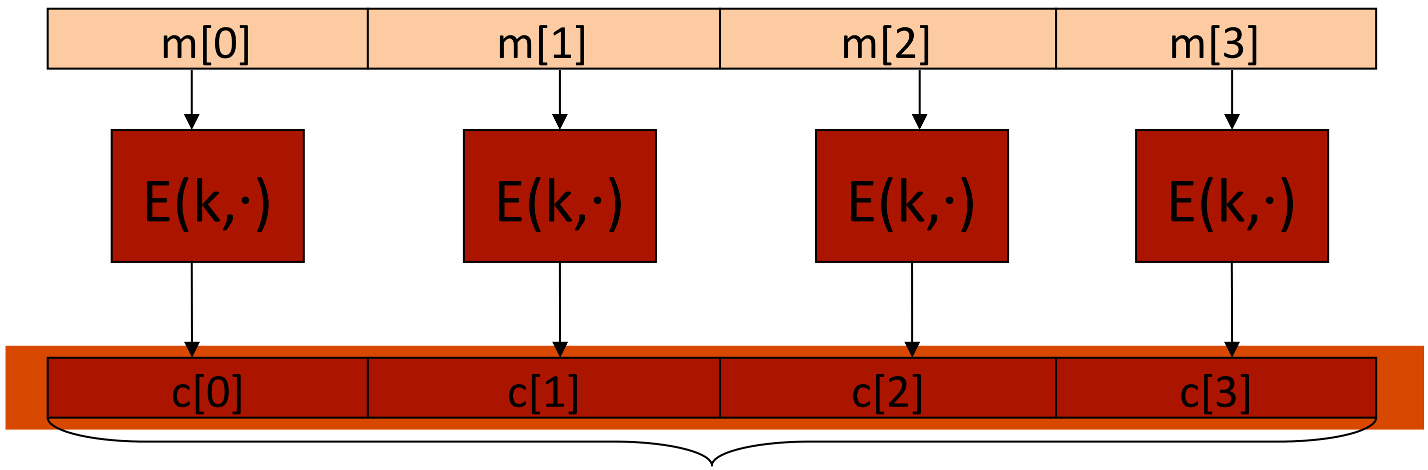
- I said “Yes” today, but I will not keep saying “Yes” forever.
- Include a *nonce* or a timestamp to avoid replay attack.
  - Nonce = *Number* used *Once*
  - May be given by a bigger application

- P.S. Again, encryption is designed for confidentiality
- So encryption doesn’t not care about “replay”

Mode of Operation for Authentication

- What if a message to be authenticated is too long?
  - *i.e.*, longer than the block length
  - *e.g.*, m[*i*] is a block and M:= m[0] || m[1] ||m[2] || m[3]
  - || denotes string concatenation
- Let's reconsider those mode of operations (ECB, CFB, OFB, CTR)
- *i.e.*, the "cipher" are treated as a MAC (or we call it a "tag")

ECB for Authentication?



Now treat them as the tag for the message M:= m[0] || m[1] ||m[2] || m[3]
(say, block-size = 1 letter and m[0] = ‘L’, m[1] = ‘O’, m[2] = ‘V’, m[3] = ‘E’)

After seeing M and the corresponding tag c[0] || … || c[3], can you forge?

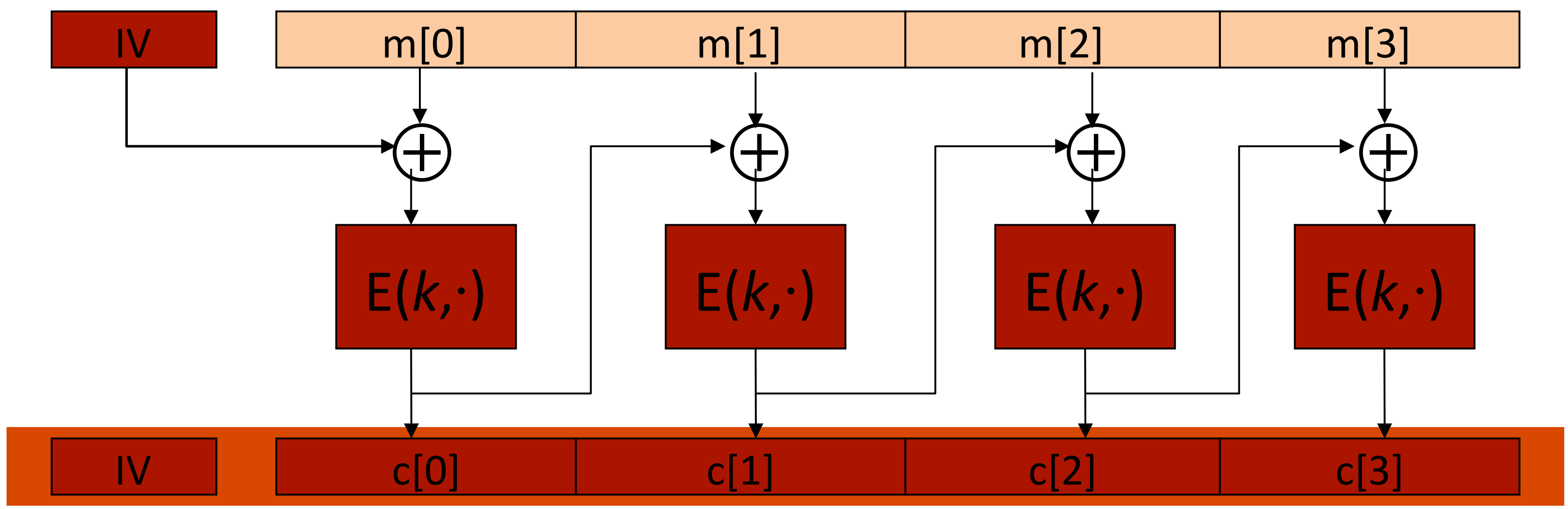
The tag c[0] || c[2] is authenticating a message “LV” *different from* M = “LOVE”

Outline

- MAC by cipher and Mode of operation
  - *e.g.*, insecure CBC-residue a.k.a. DAC, CBC-MAC
- Hash Function, Collision and Birthday Paradox
- Hash Mac (HMAC)
- Hash Function from Compression Function

CBC for Authentication

- Recall I said that IV can be sent in clear for CBC encryption?
- Do we really need the whole ciphertext for authentication?
- Do we need to invert E() for authentication?

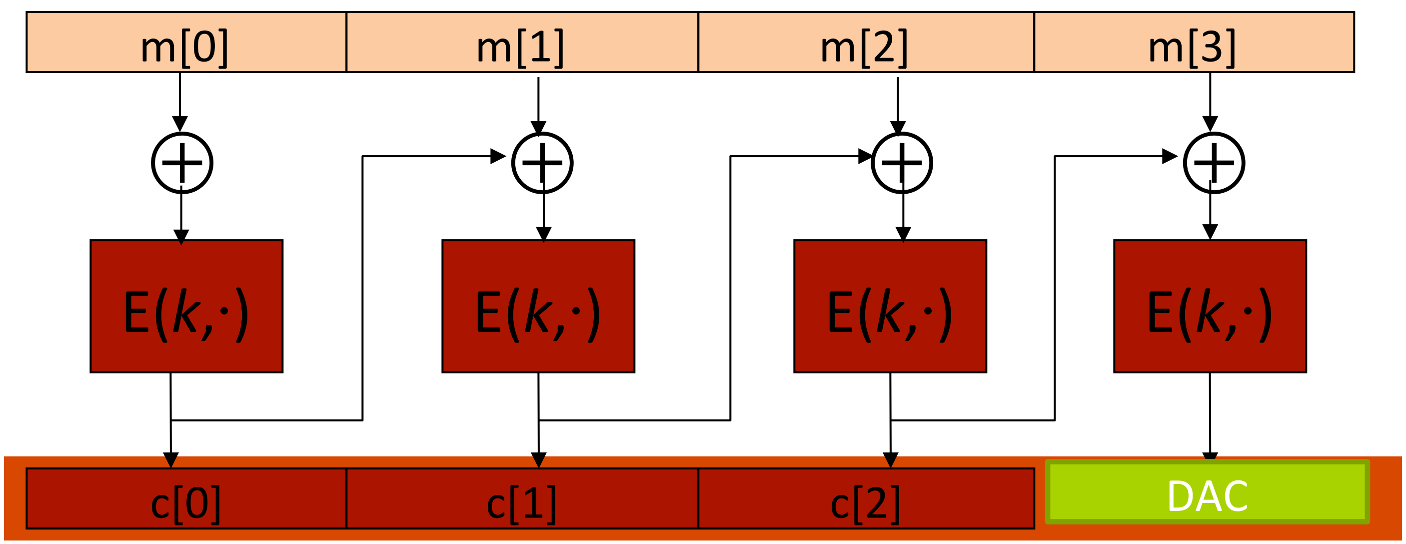


Design Principle of MAC from Mode of Op.

- Any IV used should also be sent for authentication purpose
- For MAC, the message is there already, one just needs to check.
  - Decryption for recovering the message requires the entire ciphertext.
  - For MAC, one does not really need to “decrypt” or invert E()
- Perhaps we may just use the last block as the MAC
  - How many bits should we use?
  - Extreme case: just use 1-bit, there are only 2 possible MACs → insecure
- Can’t we just re-use the whole ciphertext for both purposes?
  - No, the “ciphertext” from encryption may help you to do forgery

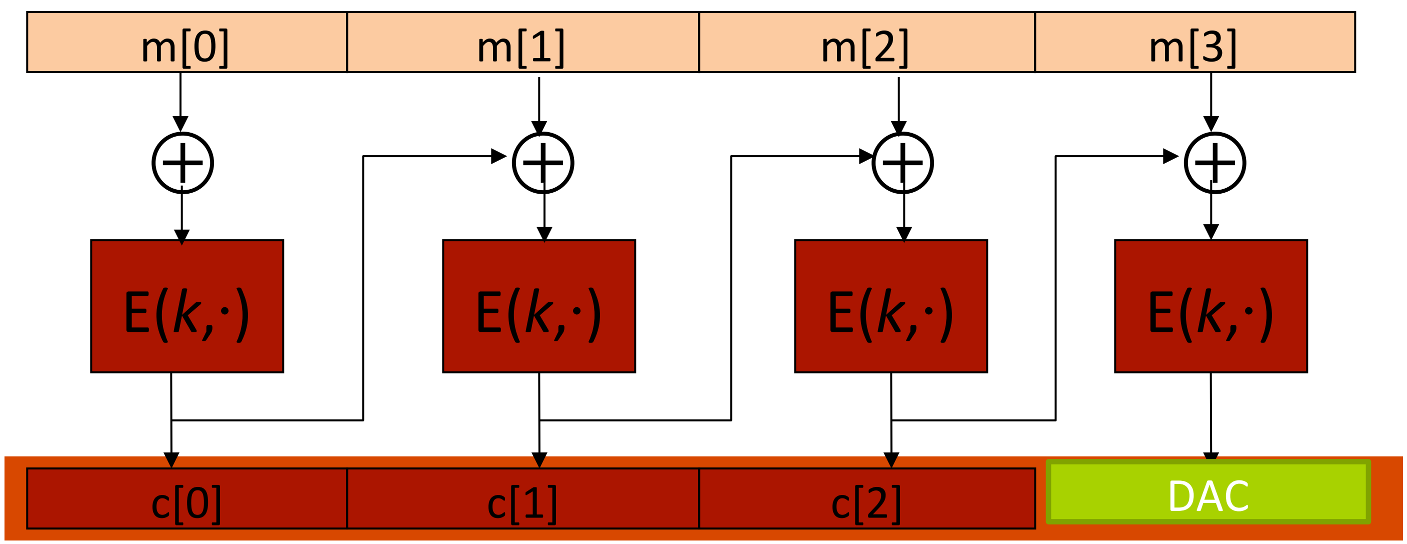
CBC-Residue (Data Authentication Code)

- Use diff. keys for diff. purposes: encryption and authentication
- Use the last block as MAC (which is called DAC in this scheme)



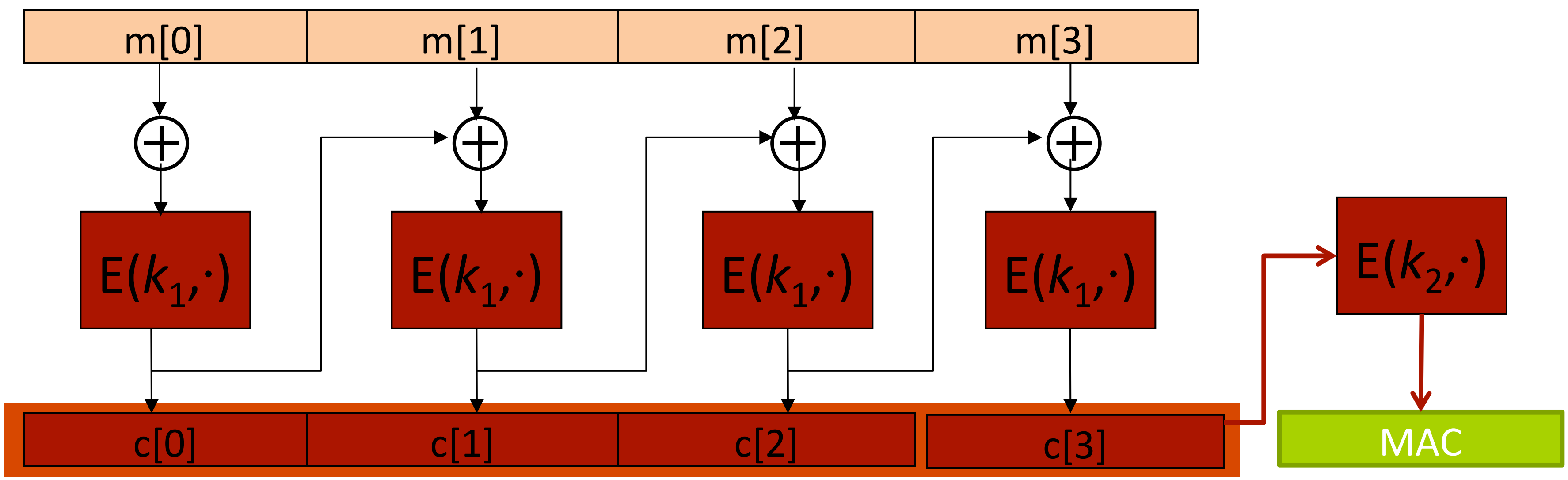
Existential Forgery under Known Message Attack

- CBC-Residue/DAC is vulnerable to “message extension attack”
  - By construction, c[0] is used as a valid MAC for m[0]
  - We can show that c[0] is also a valid MAC for (m[0], m[0] ⊕ c[0])
  - (Set m[1] := m[0] ⊕ c[0], c[1] = E(*k*, m[1] ⊕ c[0]) = E(*k*, m[0]) = c[0])



CBC-MAC fixes CBC-Residue

- Idea of the fix: add a final encryption
- Further fix: use "another key" for final encryption
- In another word, use a longer key $K = (k_1 \mid\mid k_2)$

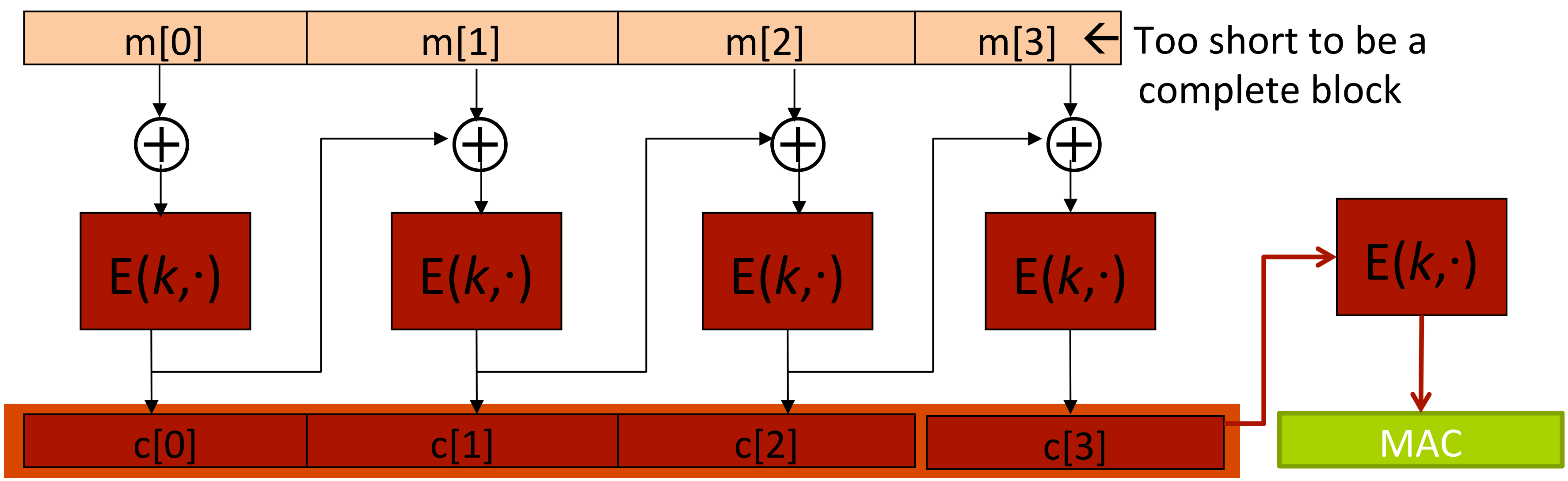


Why last encryption step?

- If we didn't do the last encryption step in CBC-MAC
- Given a MAC on one message
- It is easy to forge a MAC on a different message, *e.g.*, message extension
- High-level idea: adversary cannot see the intermediate value
  - Recall "Can't we just re-use the whole ciphertext?"
  - No, the "ciphertext" from encryption may help you to do forgery
  - This also explains why it is not always the longer the better

"Secure" Padding in CBC-MAC

- What if the message length is not a multiple of block size?
- Padding, specifically, "collision-free" padding
  - *e.g.*, padding with just 0's is bad (since MAC (m) = MAC (m000))



Invertible Padding

- If there is collision, two inputs lead to the same output
  - *i.e.*, for an output, inverting the function gives 2 different inputs
- For security, padding must be invertible!
- $m_0 \neq m_1 \Rightarrow \mathrm{pad}(m_0) \neq \mathrm{pad}(m_1)$
- ISO: pad with "1000…00"
  - Add new dummy block if needed.
  - The `1' indicates beginning of pad.

Using Hash instead of Mode of Op.

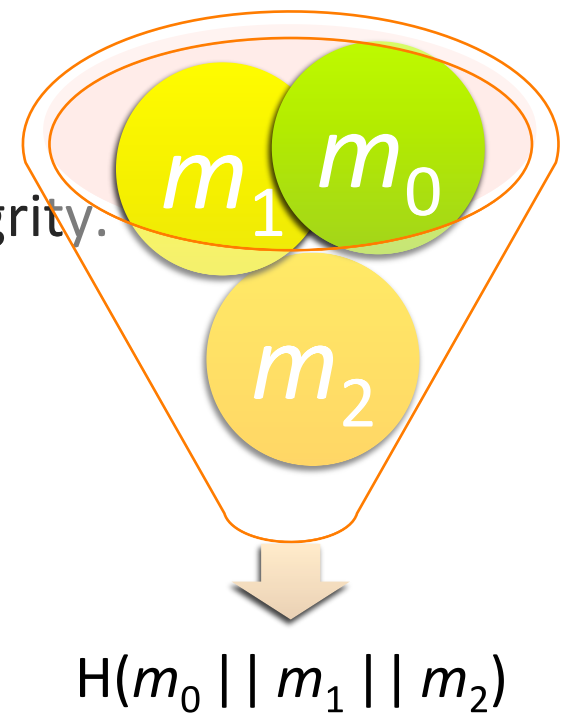
- Instead of using mode of operation,
- can we "pre-process" the message, and use a shorter string as the "digest" of the message to be authenticated?
- Yes, such pre-processing can be done by a "hash function" H()
- If one can find collision, *i.e.*, m_0, m_1, s.t. H(m_0) = H(m_1),
- the MAC using this H() is insecure!

Why not just use encryption?

- Encryption software is slow
- Encryption hardware costs aren't cheap
- Hardware optimized toward large data sizes
- Allows "recovery" / decryption, not needed in authentication
- Encryption algorithms are usually covered by patents
- Algorithms subject to US export control

Hash Function for Message Digest

- Hash function accepts a *variable* size message M as input and produces a *fixed-size* message digest $H(M)$ as output
- Message digest is sent with the message for authentication
- Produces a fingerprint of the message
- Hash function *helps* in application for integrity.
  - But it does not provide integrity by itself!
- No secret key is involved
  - So $H(M)$ is not a secure MAC of M!



MAC from One-Way Hash

- Candidate scheme of MAC: Set tag t = E(k, h)
  - How to verify? (Notice that this MAC scheme is deterministic)
  - Is it secure?
- Suppose h is the message digest H(M) of message M
- If H() is "one-way", why not just use H(k || M)?
  - Not secure, extension attack for specific H is possible
  - More details later…
  - We will see in the last slide a scheme called "HMAC"

Cryptographic Hash Function

1. Functional requirements:
   - H can be applied to a block of data of any size
   - H produces a fixed length output
   - $H(x)$ is relatively easy to compute
2. One-wayness --- For any given code h, it is computationally infeasible to find x such that $H(x) = h$ (*i.e.*, safe against “1st preimage attack”)
3. Weak collision-resistance (CR) --- For *randomly* chosen x, it is comp. infeasible to find $y \neq x$ s.t. $H(y) = H(x)$ (“2nd preimage resistance”)
4. Strong CR --- Comp. infeasible to find any (x,y) s.t. $H(x) = H(y)$

   Note: CR doesn’t imply one-wayness [**]

Application of CRHF

- *E.g.*, You want to download from http://www.openoffice.org
- What if the attacker replace the packets for the download?
- Check MD5 (assumed to be a CRHF)
- Reduced the problem (from checking the authenticity of a large file) to checking the authenticity of a digest
- *E.g.*, Unix command md5 (md5sum)

CRHF + Access Control = File Integrity

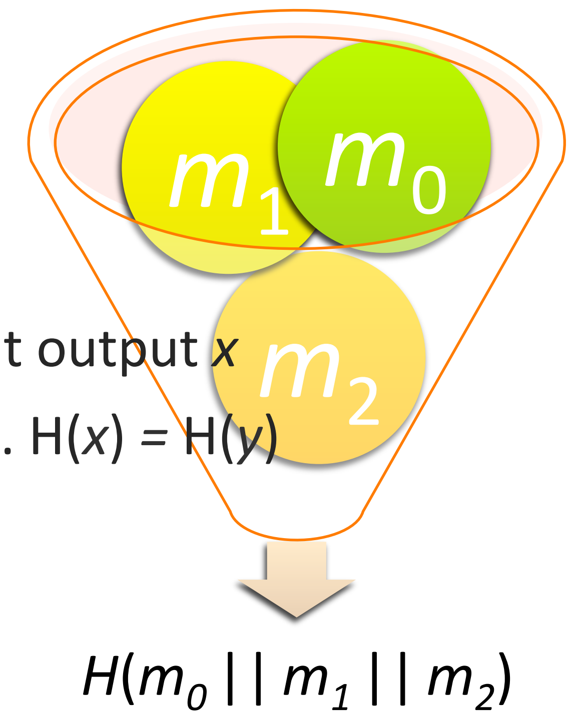
- Assume you are using an operating system with access control
- In particular, it can enforce read-only space
  - readable to public, not writeable in general
- To protect file *F*'s integrity, just place H(*F*) there
  - OS protection for a smaller file instead of a larger file
- Collision-resistant ⇒ attacker can't modify package w/o detection
- Un-keyed ⇒ Public-verifiability
  - (*cf.*, if a secret key is needed to verify, it is a private verification)
- No (secret) key ⇒ Can only trust the OS ('s read-only guarantee)
  - Or its contrapositive: if you do not want to trust on software, crypto can help

When are they useful?

- One-wayness: storing H(password) in Unix
  - *cf.*, LinkedIn (https://en.wikipedia.org/wiki/2012_LinkedIn_hack)
  - Not one-way → Can find inverse
- Consider using hash as a "commitment" for sealed-bid auction
  - Even though it may be one-way, it is "insecure" for a small message set
  - *E.g.*, If I know the bid is in the range of [1 … 100], can't I just test?
  - Remember salting in H(password)?
  - Salt comes from a large space (of randomness)
- Weak-CR: software distribution
  - Is software random? Strong-CR if in face of malicious developer

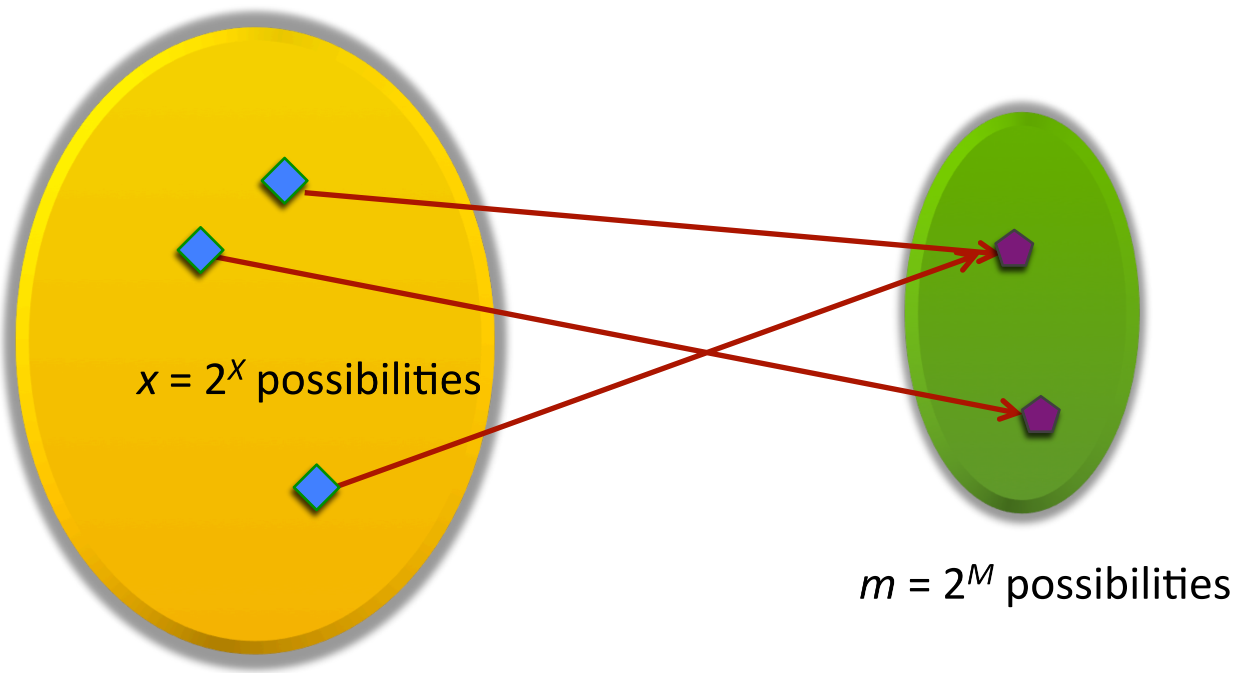
Summary so far

- Mode of Operations from Block Cipher
  - Can these be used to create MAC (message authentication code)?
- (Universal | Existential) Forgery against (Known | Chosen) Message Attack
- Collision-Resistant Hash Function (CRHF)
  - One-way: For a random *x*, given *h* = H(*x*) can't output *x*
  - Weak C.R.: For a random *x*, can't output *y* s.t. H(*x*) = H(*y*)
  - Strong C.R.: Can't output (*x*,*y*) s.t. H(*x*) = H(*y*)



How likely is collision?

- H: $\{0, 1\}^X \rightarrow \{0, 1\}^M$ (lossy compression function)
- Collisions are inevitable when $X > M$



Let's play a game again

- I randomly select a subset of size n of student from this class.
- I will pay you if no two of them share the same birthday.
- Otherwise you pay me.
- For what n you will enter this game?

- Rephrase: In a room with n people, what is the probability that we will find at least 2 people who have the same birthday?
- (there are m = 365 possible choices of birthday)

An Approximate Analysis

- Assuming birthdays are uniformly distributed over the entire year.
- For any given pair, the prob. of them sharing the same birthday = $1/m$
- There are ${}_{n}\mathbf{C}_{2} = n(n-1)/2$ ways to select a pair out of n people
- Let P be the Probability of at least one collision, $P \approx n(n-1) / (2m)$
- So, $P > ½$ when $n \geq 20$ (19 x 19 = 361)
- In general, $P > ½$ when n becomes $>= \sqrt{m}$
  - Not a good approximation when n approaches m

An Exact Analysis

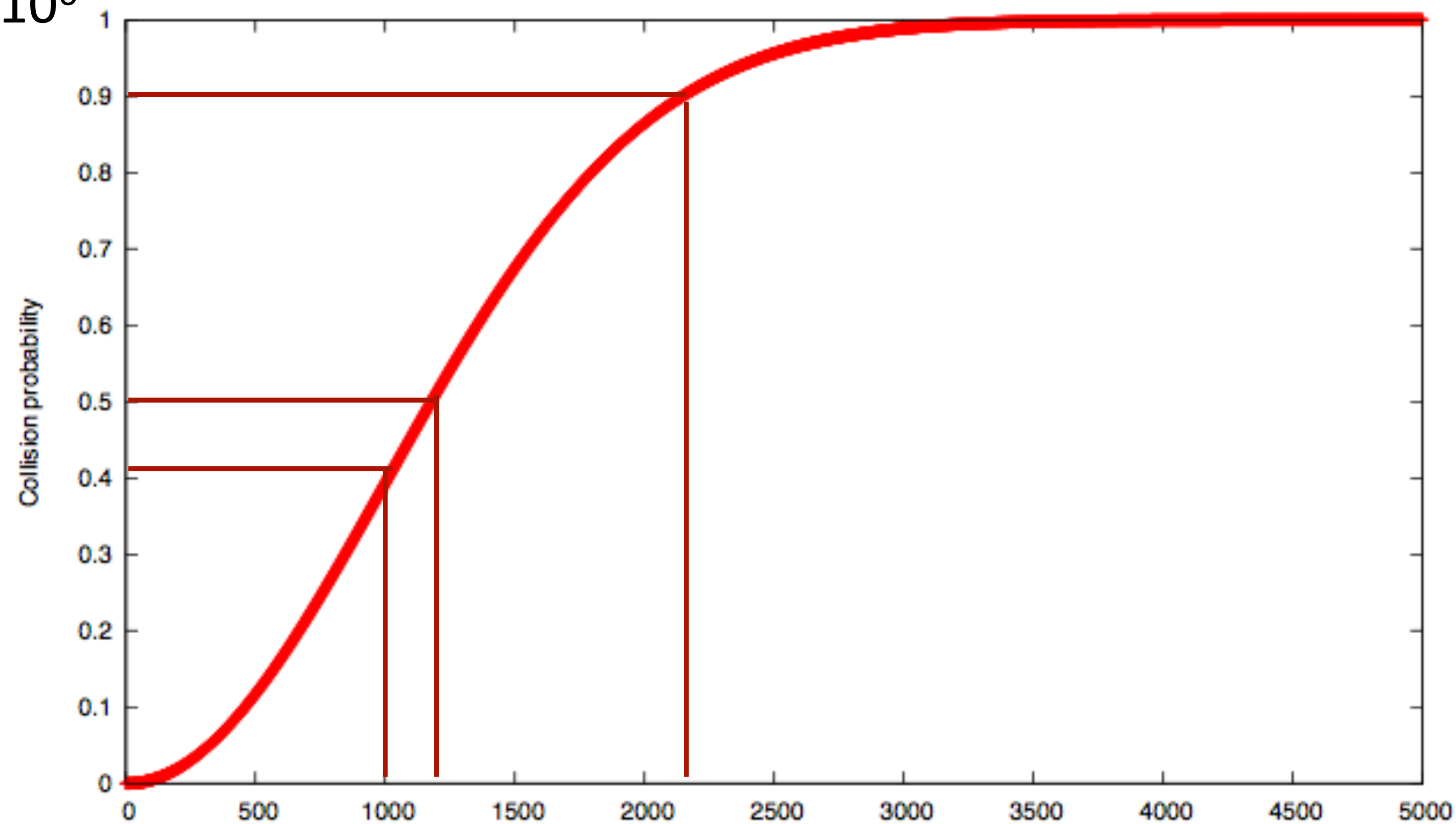
- Probability of zero collision
  = Probability that all of the n people have different birthdays
  = (m / m) x ((m-1) / m) x ((m-2) / m) … ((m-n+1) / m)
  = **m x (m-1)(m-2) … (m-n+1) / m^n**
  = 1 – n(n-1)/2m approximately when m>>n
- P = 1 – Probability of zero collision = n(n-1)/2m approximately
- "Birthday Paradox"

How difficult to find a collision?

- Just try ~ $n = \sqrt{m}$ inputs to H, have a gd. chance of a collision.
- *E.g.*, consider a hash function with 64-bit output.
- It only takes about $\sqrt{m} = 2^{32}$ tries to find a pair of inputs which will produce the same hash output, *i.e.*, a collision

How easy to find a collision?

$m = 10^6$



samples n

[from Dan Boneh]

Birthday Attack

- Generates 2^{32} variations of a valid message
  - all with essentially the same meaning
  - “doable” given current technology
- Generates 2^{32} variations of a desired fraudulent message
- Two sets are compared to find a pair with same hash output
- (by argument similar to the Birthday paradox, this probability > 0.5)
- Have the victim authenticate/sign the valid message
- The fraudulent message has the same authenticated message digest

Example

Type 1 message

I am writing {this memo | } to {demand | request | inform you} that {Fred | Mr. Fred Jones} {must | } be {fired | terminated} {at once | immediately}. As the {July 11 | 11 July} {memo | memorandum} {from | issued by} {personnel | human resources} states, to meet {our | the corporate} {quarterly | third quarter} budget {targets | goals}, {we must eliminate all discretionary spending | all discretionary spending must be eliminated}.

{Despite | Ignoring} that {memo | memorandum | order}, Fred {ordered | purchased} {Post-Its | nonessential supplies} in a flagrant disregard for the company's {budgetary crisis | current financial difficulties}.

Type 2 message

I am writing {this letter | this memo | this memorandum | } to {officially | } commend Fred {Jones | } for his {courage and independent thinking | independent thinking and courage}. {He | Fred} {clearly | } understands {the need | how} to get {the | his} job {done | accomplished} {at all costs | by whatever means necessary}, and {knows | can see} when to ignore bureaucratic {nonsense | impediments}. I {am hereby recommending | hereby recommend} {him | Fred} for {promotion | immediate advancement} and {further | } recommend a {hefty | large} {salary | compensation} increase.

Hash Function used in Practice: MD5

- By Ron Rivest in ’92, RFC 1321
- Input: *arbitrarily long*. Output: 128-bit digest
- was most widely used secure hash algorithm
- MD5 shows significant crack in summer 2004 by a Chinese Team including WANG Xiao Yun whom found a collision pair
- MD5 was totally broken by ’08
- all are collision attacks, no preimage attack is found so far

SHA-1

- SHA-0 designed by NIST & NSA in ’93, revised as SHA-1 in ’95
- Design criteria were not disclosed
- Input is processed in 512-bit blocks, output 160-bit
- Slower than MD5, was the generally preferred (over MD5)
- Considered to be Very Secure – Only until Feb 2005
- Wang *et al.* found a way to reduce the complexity of finding hash collisions from 2^{80} to 2^{68}
  - *i.e.*, a speed up of 4096 times

RIPEMD-160

- Original RIPEMD is from European RIPE Project – 1997
- From COSIC (Computer Security and Industrial Cryptography) group at Katholieke Universiteit Leuven
  - Led by Bart Preneel (contribute RIPEMD), Vincent Rijmen (contribute AES)
- Same Chinese group found collision on *original* RIPEMD
- Built from the experience gained by evaluating MD5, and RIPEMD
- Extended from 128 ($2^{64} \approx 2x10^{19}$ is insufficient) to 160-bit digest

Comparison of MD5, SHA-1, RIPEMD-160

	SHA-1	MD5	RIPEMD-160
Digest Length	160 bits	128 bits	160 bits
Basic Unit of Processing	512 bits	512 bits	512 bits
Number of steps	80 (4 rounds of 20)	64 (4 rounds of 16)	160 (5 pair rd. of 16)
Max. Message Size	2^{64}-1 bits	∞	∞
Sample Speed	6.88 Mbyte/sec	17.09 Mbyte/sec	5.69 Mbyte/sec

(Results obtained from 90MHz Pentium)

http://www.esat.kuleuven.ac.be/~bosselae/fast.html

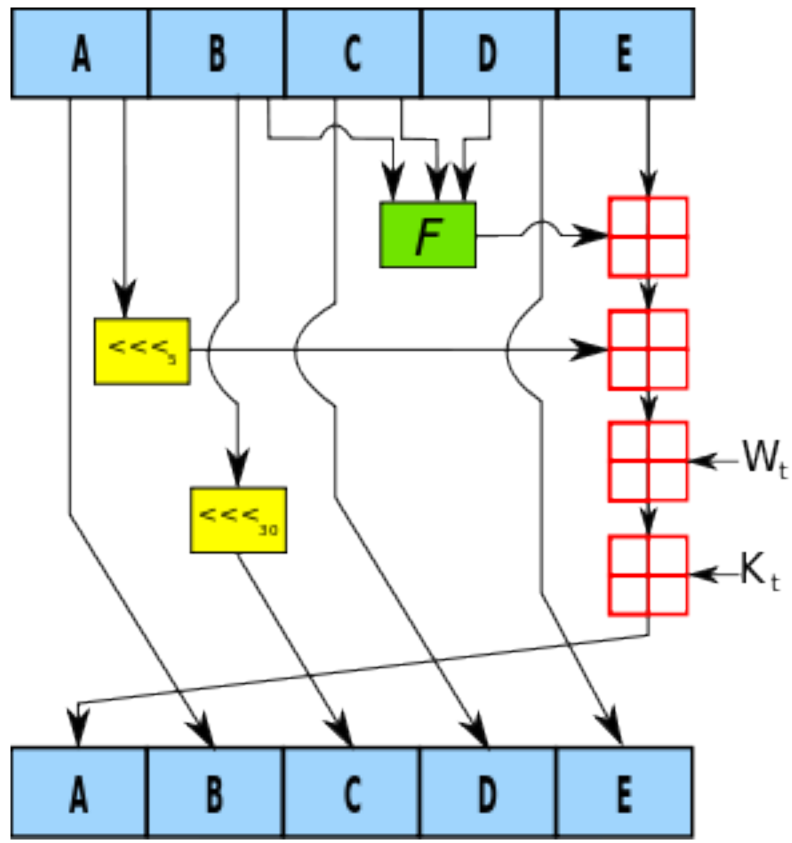
NIST SHA-3 Competition

- 2007 - 2012: http://csrc.nist.gov/groups/ST/hash/sha-3
- We talked about MD5, now MD6: http://groups.csail.mit.edu/cis/md6
- On Dec. 9, ’10, we have the Final FIVE candidates for the Round 3:
  - http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/documents/Email_Announcing_Finalists.pdf
  - http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html
- On Oct. 2, ’12, Keccak, (pronounced “catch-ack”) won
  - Designed by a team of researchers from Belgium and Italy
  - http://www.nist.gov/itl/csd/sha-100212.cfm

Compression Function in SHA1

- AND, OR, XOR, Not
- + (mod 2^32) [+]
- Circular shift (<<<)
- etc.

- Much faster than encryption
- Actual details omitted [**]



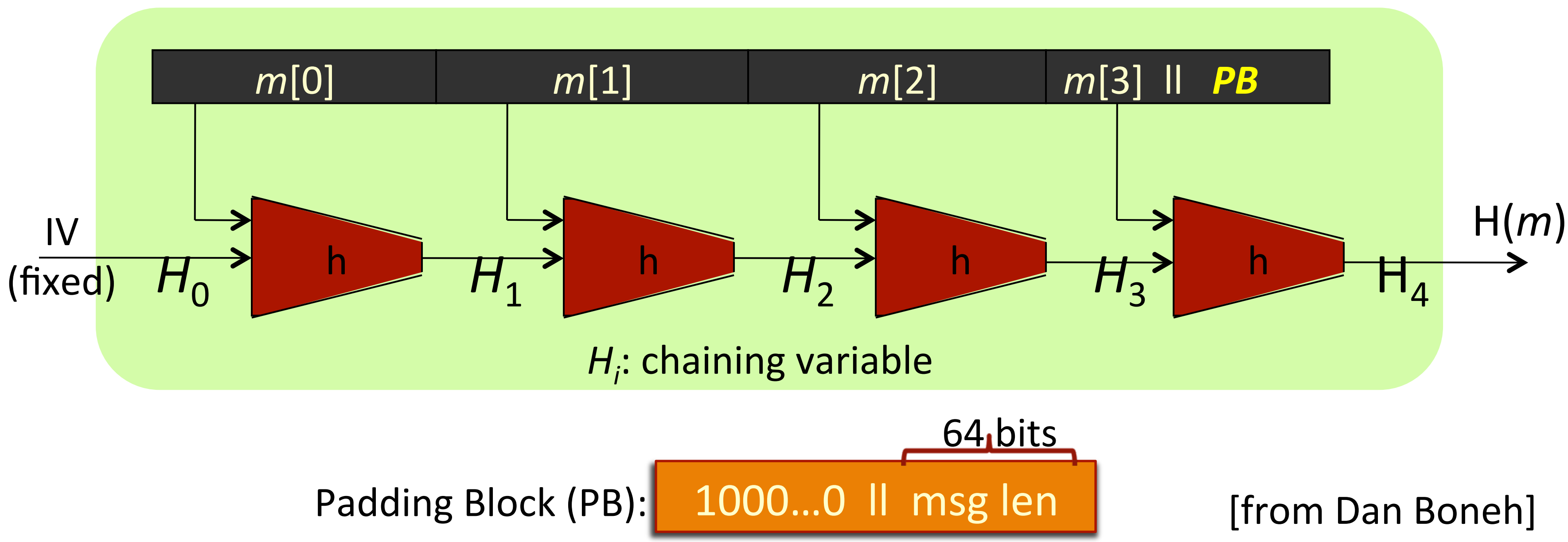
[from Wikipedia]

Extending the Domain of CRHF

- Recall those modes of operations “extend” block cipher.
- How can MD5, SHA1, SHA2 process an arbitrarily long message?
- Given a CRHF for *short* messages
- + a “compression function” (not zip, rar, *etc.* which are recoverable)
  - like hash function, it is also a public function (if the input is also public)
- We can construct a CRHF for *long* messages
- Via Merkle–Damgård construction
  - A general design in MD5, SHA-1, and SHA-2
  - described in Merkle's PhD thesis in ‘79, Merkle and Damgård independently proved that the structure is “sound” [**]

Merkle–Damgård Construction

- Given h: T × X ⟶ T (compression function), we want
- H: $X^{\leq L}$ ⟶ T (at most L elements, each from X; map it to T)

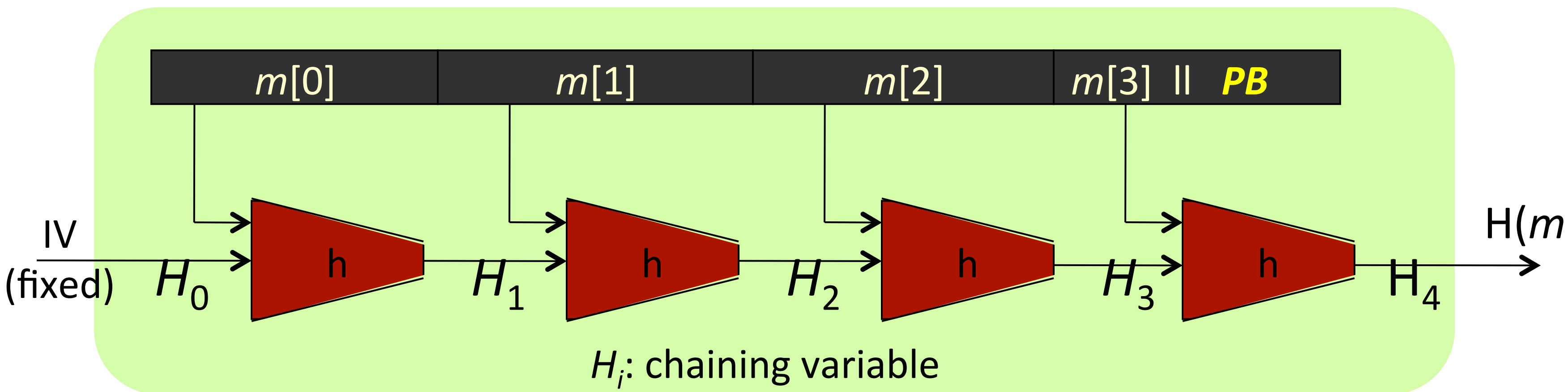


If no space, add another block

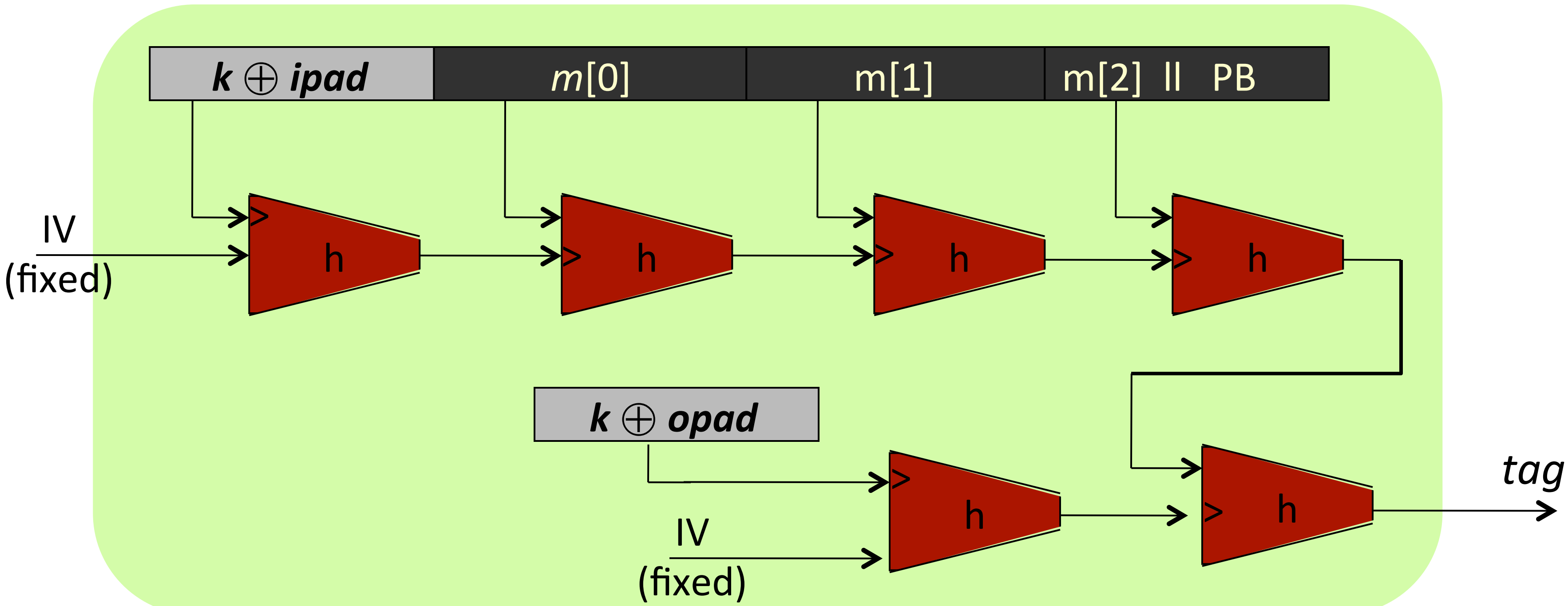
[from Dan Boneh]

Message Extension Attack

- Can we use H( ) to directly build a MAC?
- Yes, but with caution, *e.g.*, don't use **MAC(*k*, *m*) := H(*k* || *m*)**



HMAC (Hash MAC)



- Similar to CBC-MAC, derive two keys from one key
- Two constants: inner padding (*ipad*) and outer padding (*opad*)

HMAC

- Effort to develop a MAC derived from a crypto. hash code
- Executes faster in software
- No export restrictions
- Relies on a secret key
- RFC 2104 list design objectives
- Provable security properties
- Used in IPsec, TLS
- Can use diff. digest hash as a component say HMAC-SHA1/-MD5

Authenticated Encryption

- What if you want both confidentiality and authenticity?
- (Generic) Composition of 2 cryptographic primitives
  - Encrypt (*e.g.*, CBC mode) **then** MAC
  - 2 different keys for 2 crypto primitives (1 for enc., 1 for auth.)
- Authenticated mode of operation [**]
  - Single key, typically uses only one primitive (*e.g.*, block cipher)