CSCE 465 Computer & Network Security

Instructor: Abner Mendoza

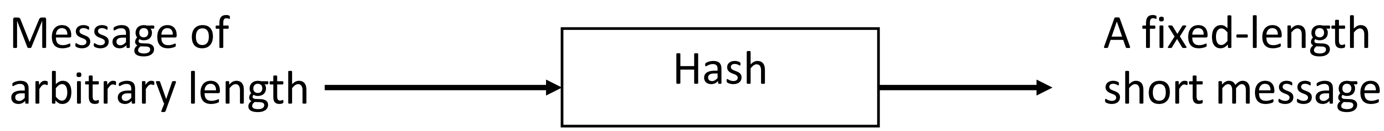
Hash Functions

Roadmap

- Hash function lengths
- Hash function applications
- MD5 standard
- SHA-1 standard
- Hashed Message Authentication Code (HMAC)

Hash Function Properties

Hash Function



- Also known as
  - Message digest
  - One-way transformation
  - One-way function
  - Hash
- Length of $H(m)$ much shorter than length of m
- Usually fixed lengths: 128 or 160 bits

Desirable Properties of Hash Functions

- Consider a hash function H
  - Performance: Easy to compute H(*m*)
  - One-way property: Given H(*m*) but not *m*, it's computationally infeasible to find *m*
  - Collision Resistance:
    - Given H(*m*), it's computationally infeasible to find *m'* such that H(*m'*) = H(*m*).
    - Computationally infeasible to find m_1, m_2 such that H(m_1) = H(m_2)

Length of Hash Image

- Question
  - Why do we have 128 bits or 160 bits in the output of a hash function?
  - If it is too long
    - Unnecessary overhead
  - If it is too short
    - Birthday paradox
    - Loss of strong collision property

Birthday Paradox

- Question:
  - What is the smallest group size k such that
    - The probability that at least two people in the group have the same birthday is greater than 0.5?
    - Assume 365 days a year, and all birthdays are equally likely
  - P(k people having k different birthdays):

    $Q(365,k) = 365!/(365-k)!365^k$
  - P(at least two people have the same birthday):

    $P(365,k) = 1-Q(365,k) \geq 0.5$
  - k is about 23

Birthday Paradox (Cont'd)

- Generalization of birthday paradox
  - Given
    - a random integer with uniform distribution between 1 and *n*, and
    - a selection of *k* instances of the random variables
  - For large *n* and *k*, to have at least one duplicate with P(*n*,*k*) > 0.5 with the smallest *k*, we have

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

  - Example in the previous case
    - $1.18*(365)^{1/2} = 22.54$

Birthday Paradox (Cont’d)

- Implication for hash function H of length m
  - With probability at least 0.5
  - If we hash about $2^{m/2}$ random inputs,
  - Two messages will have the same hash image
  - Birthday attack

- Conclusion
  - Choose $m \geq 128$

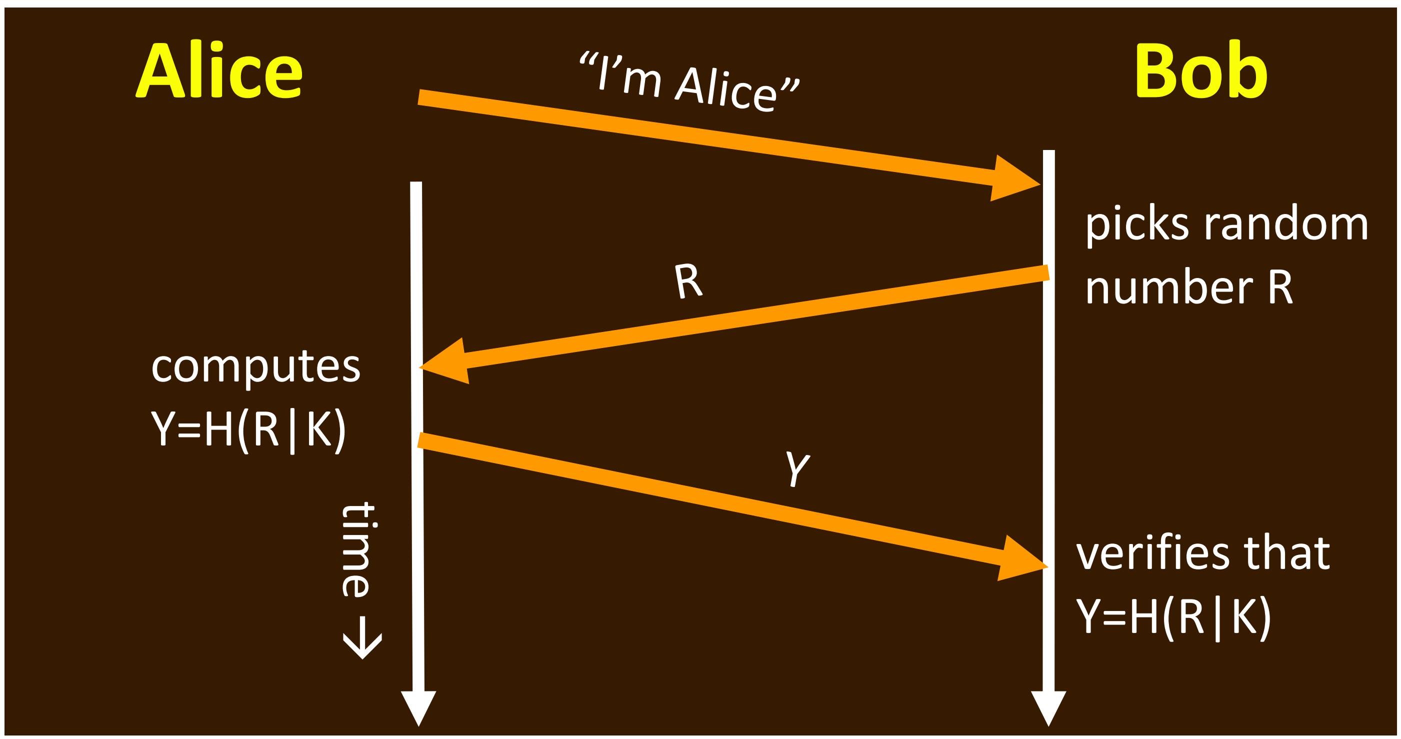
Hash Function Applications

Application: File Authentication

- Want to detect if a file has been changed by someone after it was stored
- Method
  - Compute a hash H(F) of file F
  - Store H(F) separately from F
  - Can tell at any later time if F has been changed by computing H(F’) and comparing to stored H(F)
- Example tool: Tripwire
- Why not just store a duplicate copy of F???

Application: User Authentication

- Alice wants to authenticate herself to Bob
  – assuming they already share a secret key K
- Protocol:



User Authentication… (cont’d)

- Why not just send…
  - …K, in plaintext?
  - …H(K)? , i.e., what’s the purpose of R?

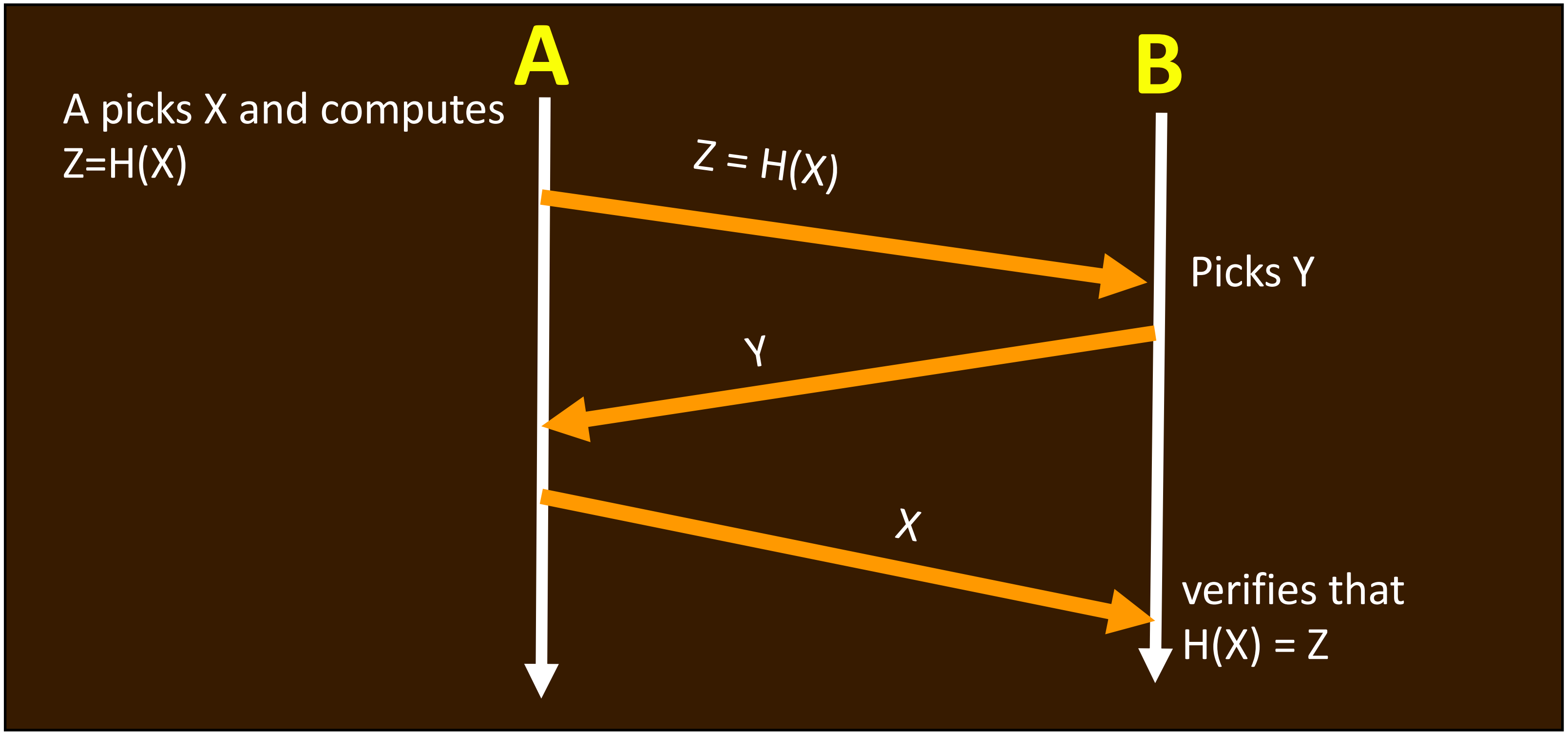
Application: Commitment Protocols

- Ex.: A and B wish to play the game of "odd or even" over the network
  1. A picks a number X
  2. B picks another number Y
  3. A and B "simultaneously" exchange X and Y
  4. A wins if X+Y is odd, otherwise B wins

- If A gets Y before deciding X, A can easily cheat (and vice versa for B)
  - How to prevent this?

Commitment… (Cont'd)

- Proposal: A must commit to X before B will send Y
- Protocol:



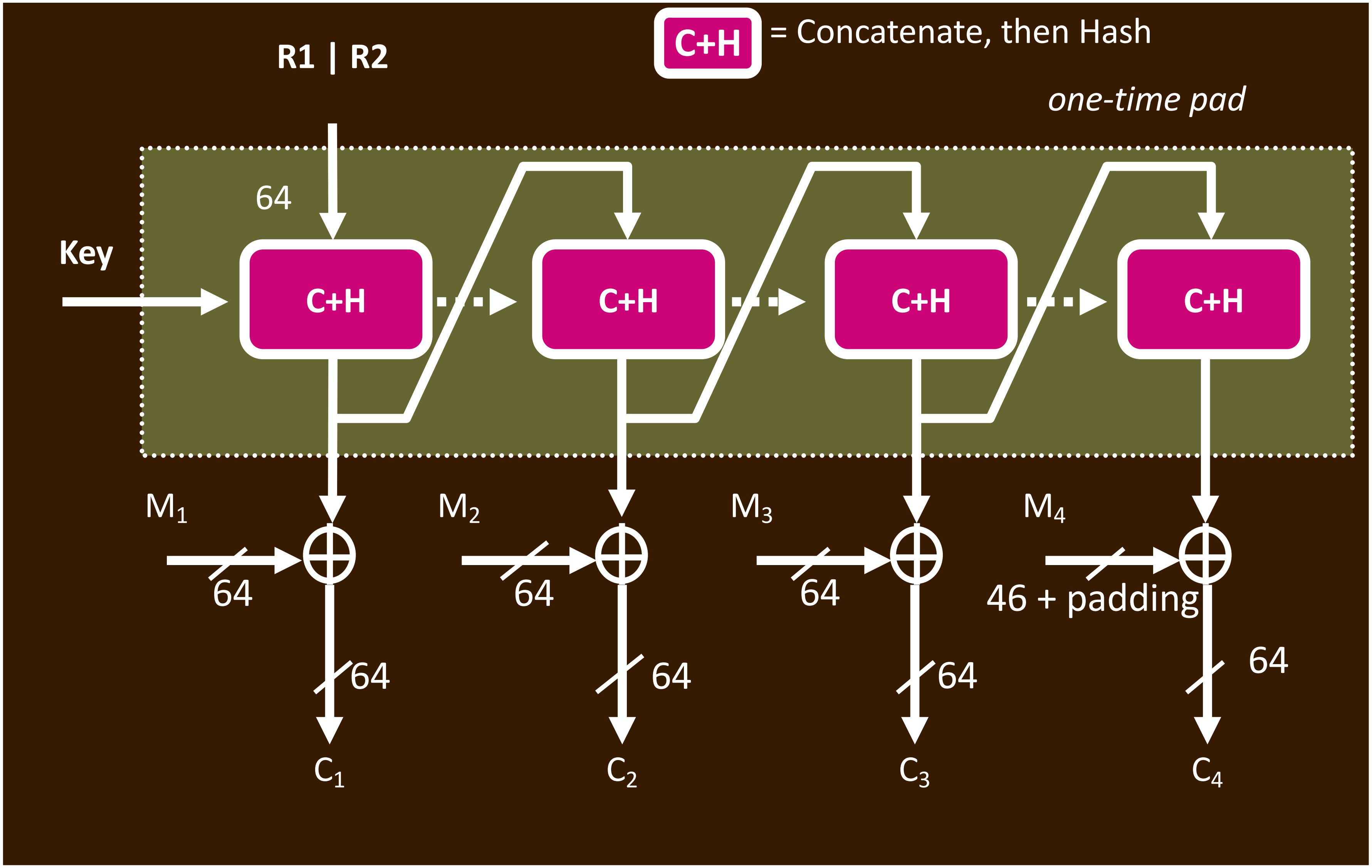
- Can either A or B successfully cheat now?

Commitment… (Cont’d)

- Why is sending H(X) better than sending X?
- Why is sending H(X) good enough to prevent A from cheating?
- Why is it not necessary for B to send H(Y) (instead of Y)?
- What problems are there if:
  1. The set of possible values for X is small?
  2. B can predict the next value X that A will pick?

Application: Message Encryption

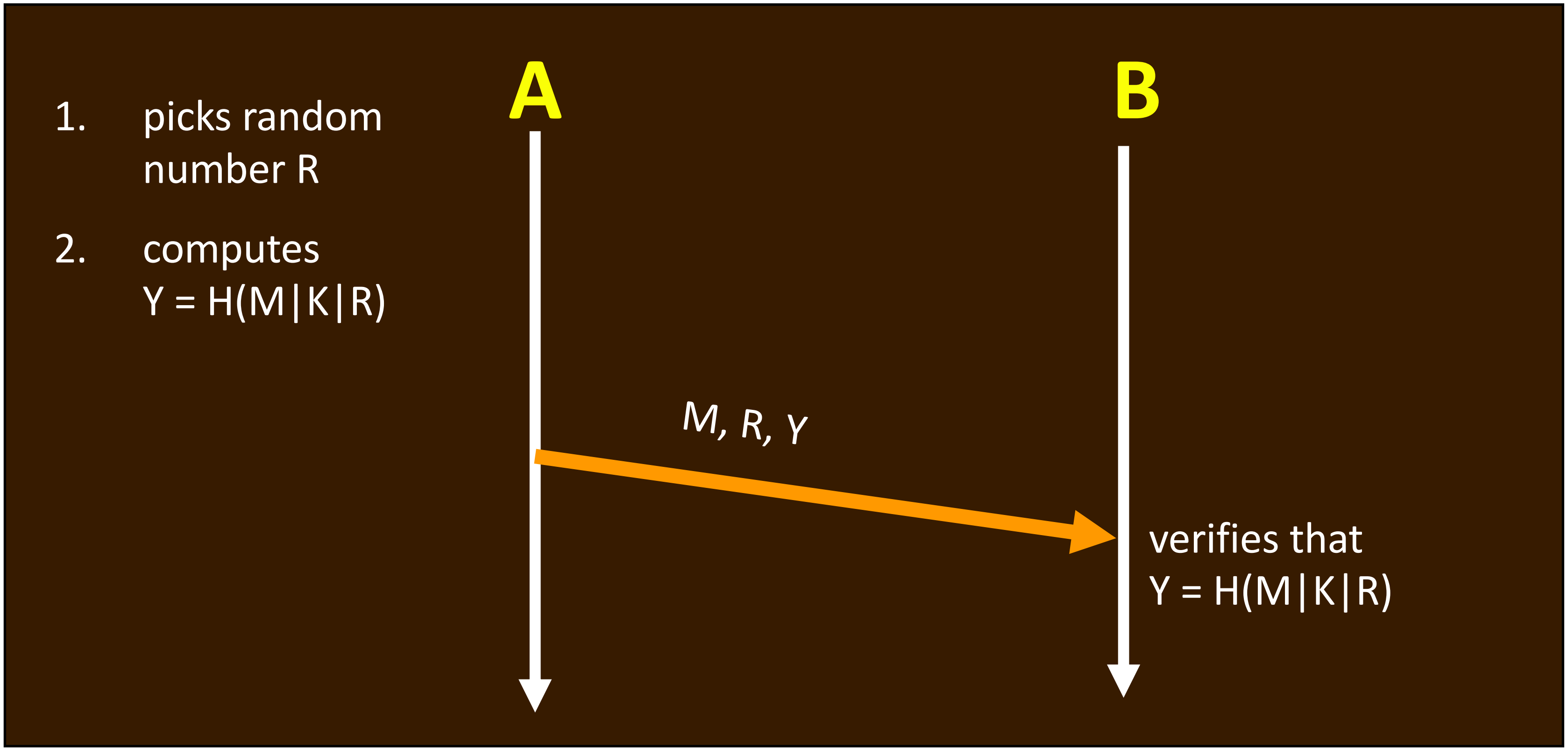
- Assume A and B share a secret key K
  - but don't want to just use encryption of the message with K
- A sends B the (encrypted) random number R1,
  B sends A the (encrypted) random number R2
- And then…



- R1 | R2 is used like the IV of OFB mode, but C+H replaces encryption;

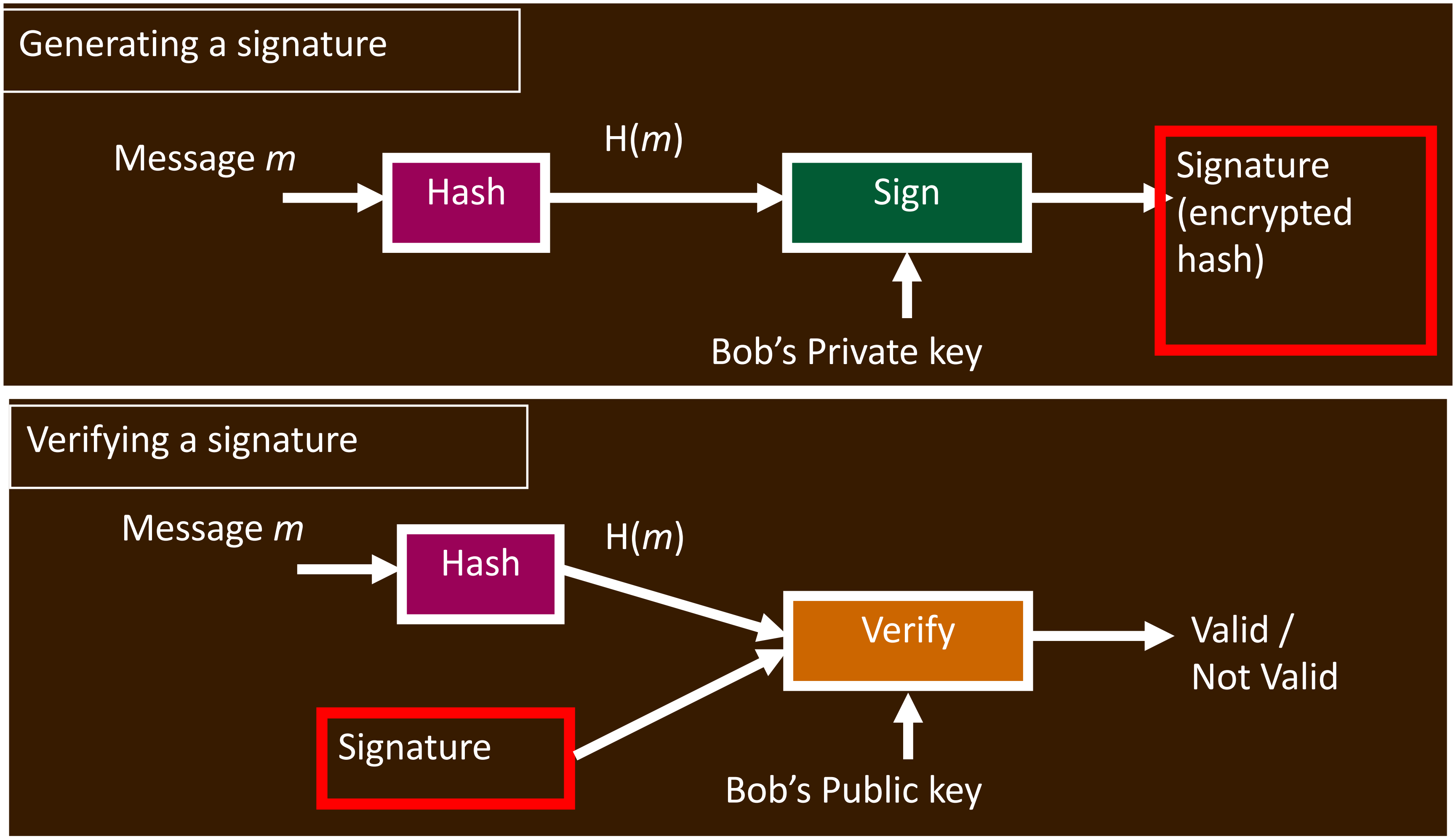
Application: Message Authentication

- A wishes to authenticate (but not encrypt) a message M (and A, B share secret key K)



- Why is R needed? Why is K needed?

Application: Digital Signatures



- Only one party (Bob) knows the private key

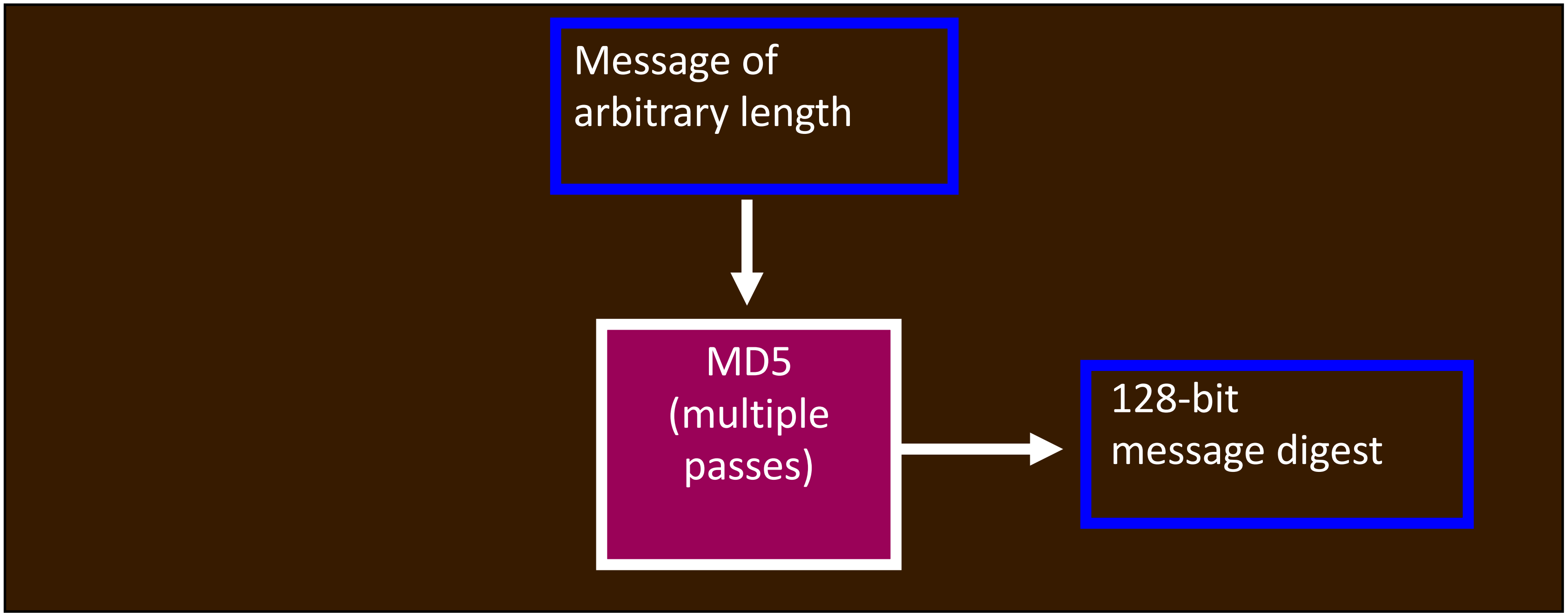
Modern Hash Functions

- MD5
  - Previous versions (i.e., MD2, MD4) have weaknesses.
  - Broken; collisions published in August 2004
  - Too weak to be used for serious applications
- SHA (Secure Hash Algorithm)
  - Weaknesses were found
- SHA-1
  - Broken, but not yet cracked
  - Collisions in 2^{69} hash operations, much less than the brute-force attack of 2^{80} operations
  - Results were circulated in February 2005, and published in CRYPTO '05 in August 2005
- SHA-256, SHA-384, …

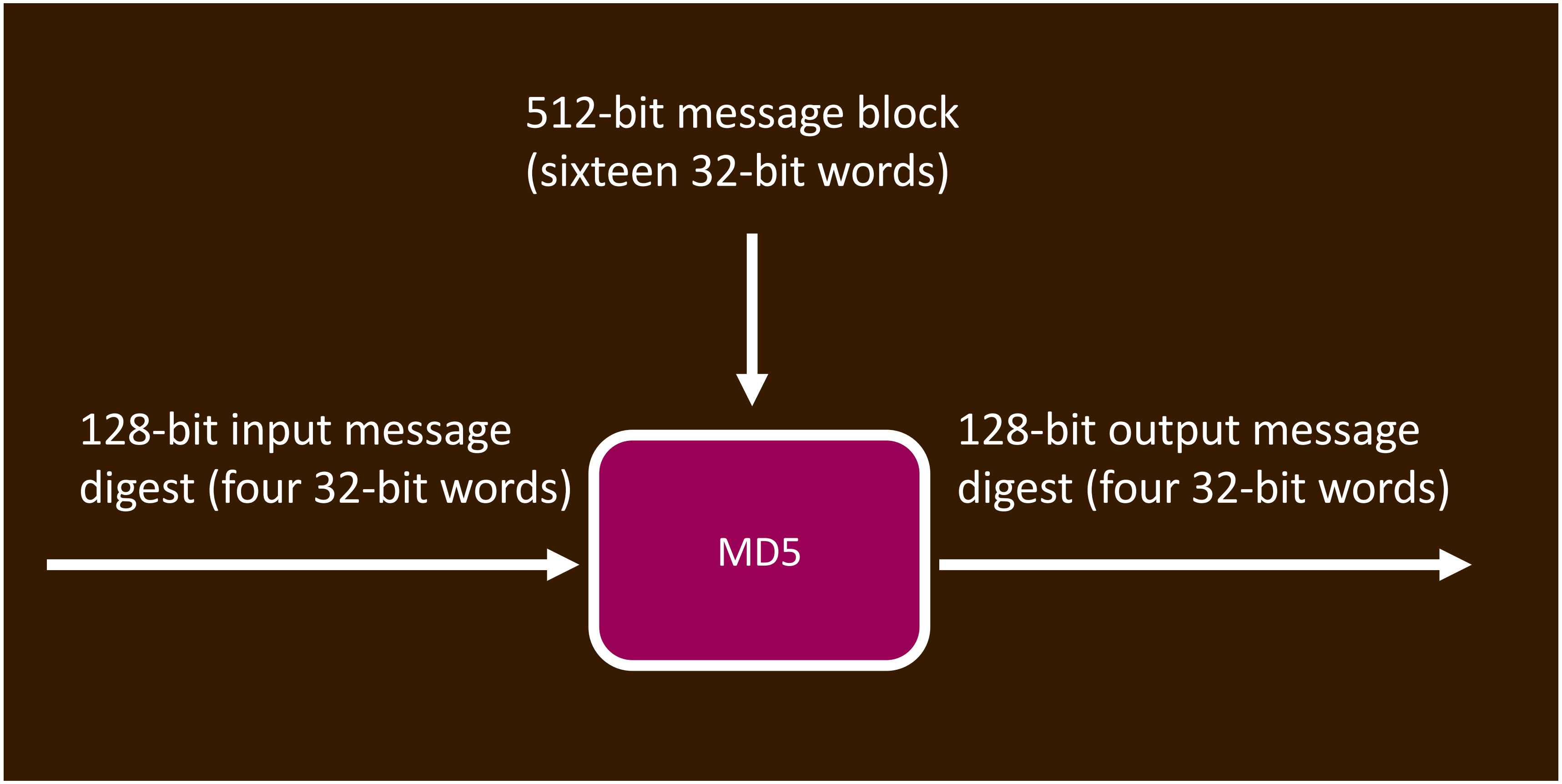
The MD5 Hash Function

MD5: Message Digest Version 5

- MD5 at a glance

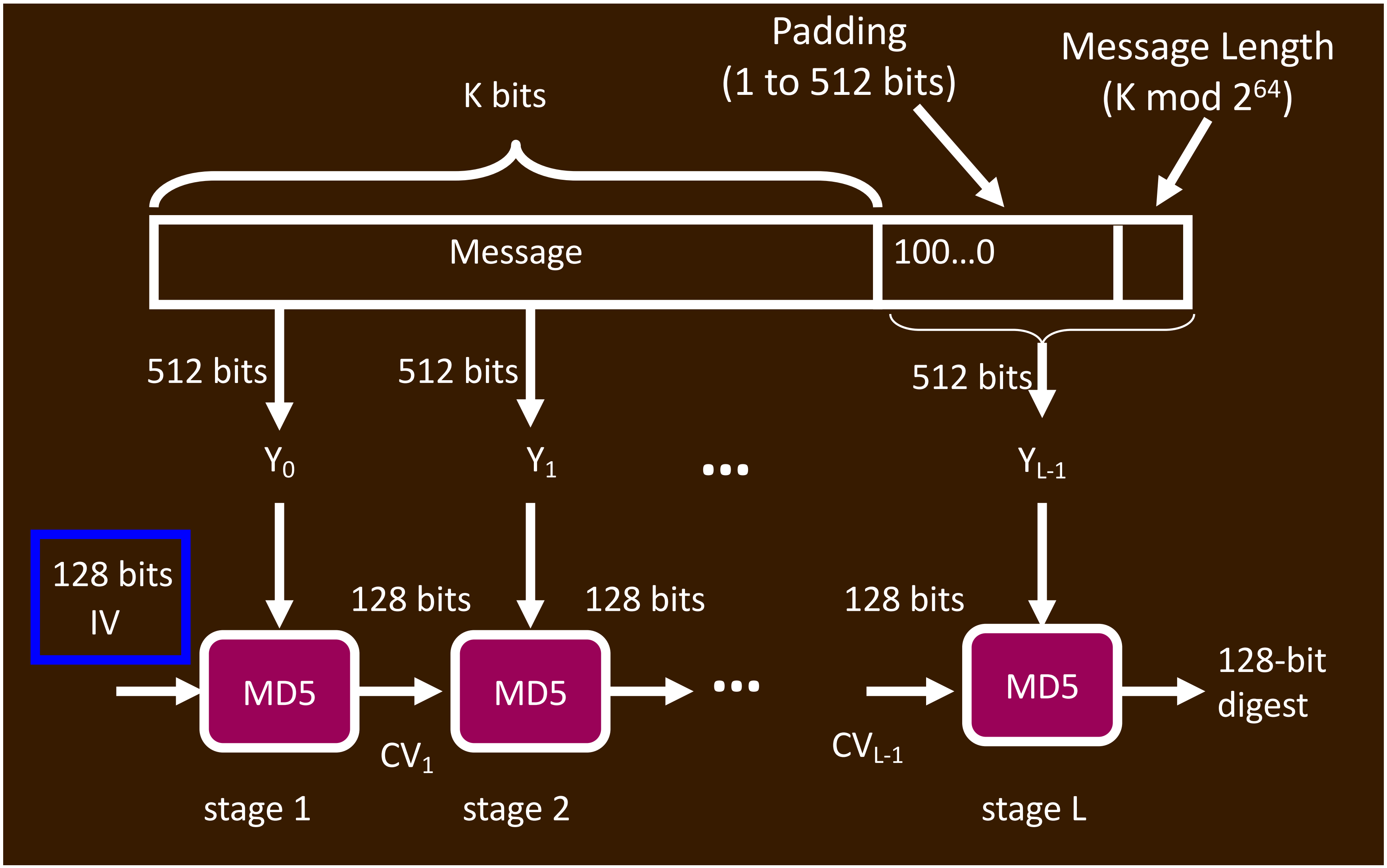


Processing of A Single Block



Called a compression function

MD5: A High-Level View



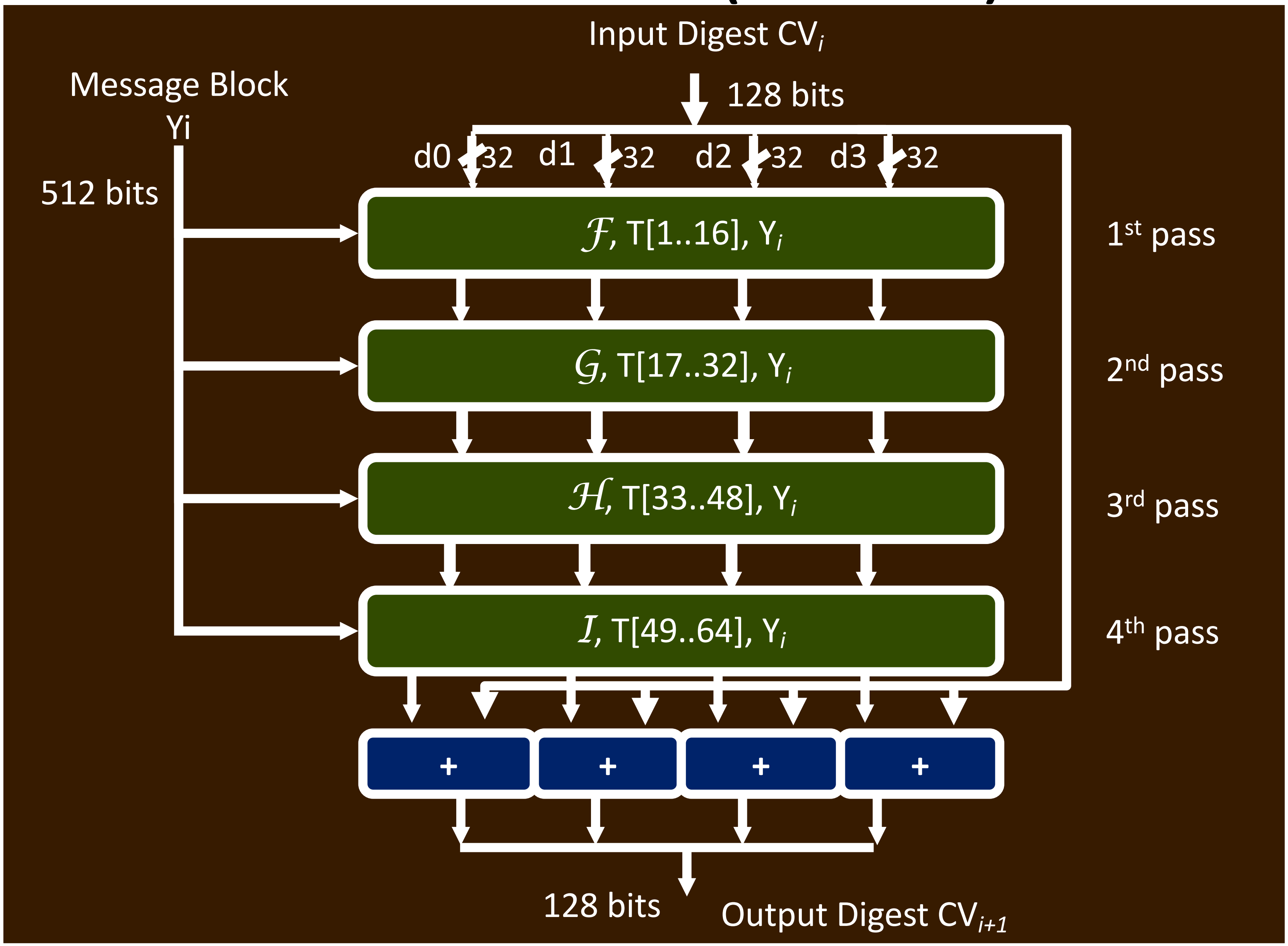
Notation

- $\sim x$ = bit-wise complement of x
- $x \wedge y$, $x \vee y$, $x \oplus y$ = bit-wise AND, OR, XOR of x and y
- $x \lll y$ = left circular shift of x by y bits
- $x+y$ = arithmetic sum of x and y (discarding carry-out from the msb)
- $\lfloor x \rfloor$ = largest integer less than or equal to x

Processing a Block -- Overview

- Every message block Yi contains 16 *32-bit words*:
  - m_0 m_1 m_2 … m_{15}
- A block is processed in 4 consecutive passes, each modifying the MD5 buffer (the *digest*) d_0, …, d_3.
  - Called $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}$
- Each pass uses one-fourth of a 64-element table of constants, T[1…64]
  - $T[i] = \lfloor 2^{32} * \mathrm{abs}(\sin(i)) \rfloor$, represented in 32 bits
- Output digest = input digest + output of 4th pass

Overview (Cont'd)

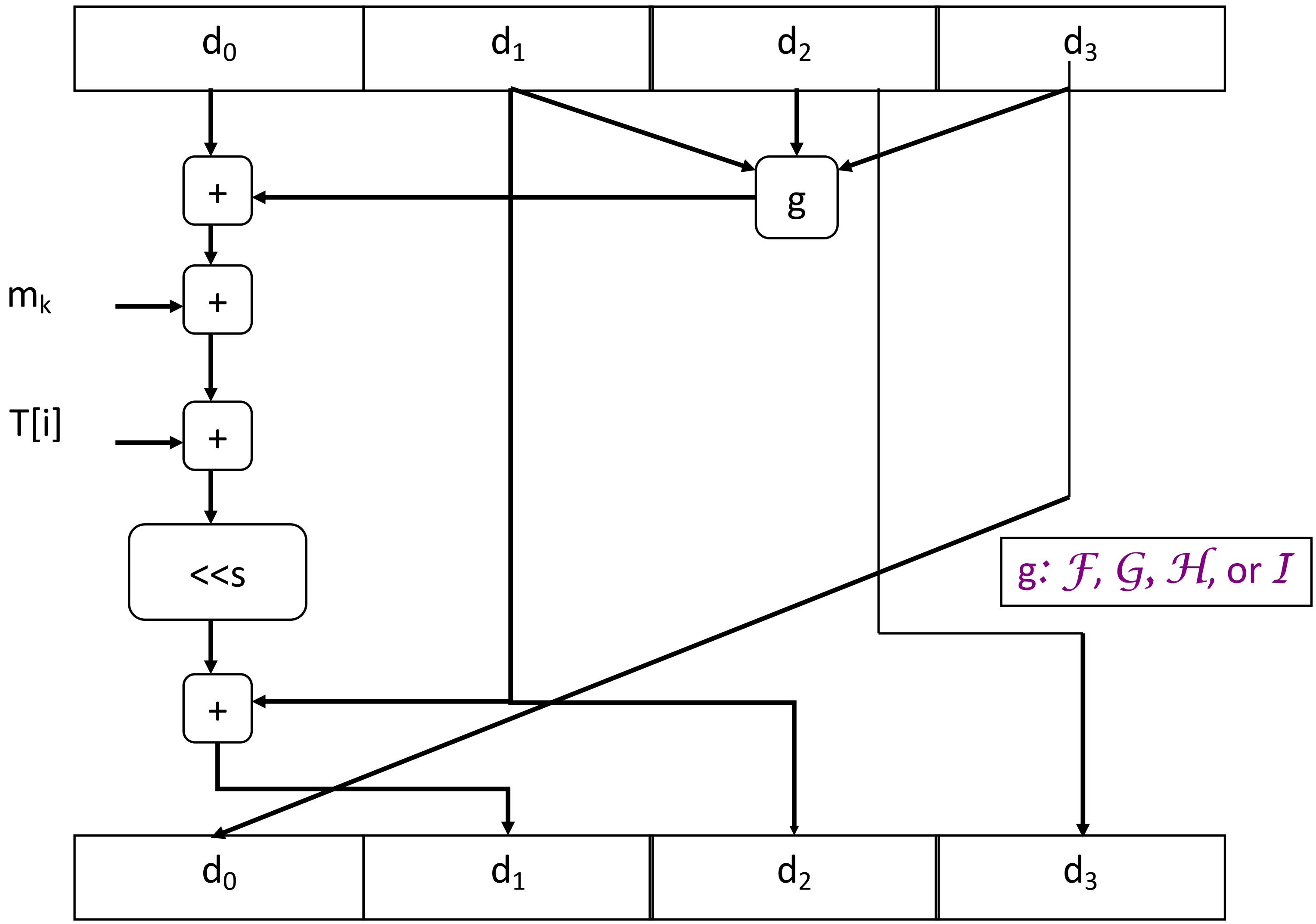


Four Passes of MD5

- $\mathcal{F}(x,y,z) \stackrel{\text{def}}{=} (x \wedge y) \vee (\sim x \wedge z)$
- $\mathcal{G}(x,y,z) \stackrel{\text{def}}{=} (x \wedge z) \vee (y \wedge \sim z)$
- $\mathcal{H}(x,y,z) \stackrel{\text{def}}{=} (x \oplus y \oplus z)$
- $\mathcal{I}(x,y,z) \stackrel{\text{def}}{=} y \oplus (x \vee \sim z)$

- Every pass has 16 processing steps (each step involves calculation using above functions and circular shift)

Logic of Each Step



(In)security of MD5

- A few recently discovered methods can find collisions in a few hours
  - A few collisions were published in 2004
  - Can find many collisions for 1024-bit messages
  - More discoveries afterwards
  - In 2005, two X.509 certificates with different public keys and the same MD5 hash were constructed
    - This method is based on differential analysis
    - 8 hours on a 1.6GHz computer
    - Much faster than birthday attack

The SHA-1 Hash Function

Secure Hash Algorithm (SHA)

- Developed by NIST, specified in the Secure Hash Standard, 1993
- SHA is specified as the hash algorithm in the Digital Signature Standard (DSS)
- SHA-1: revised (1995) version of SHA

SHA-1 Parameters

- Input message must be < 2^{64} bits
- Input message is processed in 512-bit blocks, with the same padding as MD5
- Message digest output is 160 bits long
  - Referred to as five 32-bit words **A, B, C, D, E**
  - IV: **A** = 0x67452301, **B** = 0xEFCDAB89, **C** = 0x98BADCFE, **D** = 0x10325476, **E** = 0xC3D2E1F0

- Footnote: bytes of words are stored in big-endian order

Preprocessing of a Block

- Let 512-bit block be denoted as sixteen 32-bit words $\mathbf{W_0..W_{15}}$
- Preprocess $\mathbf{W_0..W_{15}}$ to derive an additional sixty-four 32-bit words $\mathbf{W_{16}..W_{79}}$, as follows:

for $16 \leq t \leq 79$

$$\mathbf{W}_t = (\mathbf{W}_{t-16} \oplus \mathbf{W}_{t-14} \oplus \mathbf{W}_{t-8} \oplus \mathbf{W}_{t-3}) << 1$$

Block Processing

- Consists of 80 steps! (vs. 64 for MD5)
- Inputs for each step $0 \le t \le 79$:
  - $\mathbf{W}_t$
  - K_t – a constant
  - **A**,**B**,**C**,**D**,**E**: current values to this point
- Outputs for each step:
  - **A**,**B**,**C**,**D**,**E** : new values
- Output of last step is added to input of first step to produce 160-bit Message Digest

Function f(t,B,C,D)

- 3 different functions are used in SHA-1 processing

Round	Function f(t,B,C,D)	Compare with MD-5
$0 \le t \le 19$	$(B \wedge C) \vee (\sim B \wedge D)$	$\mathcal{F} = (x \wedge y) \vee (\sim x \wedge z)$
$20 \le t \le 39$	$B \oplus C \oplus D$	$\mathcal{H} = x \oplus y \oplus z$
$40 \le t \le 59$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	
$60 \le t \le 79$	$B \oplus C \oplus D$	$\mathcal{H} = x \oplus y \oplus z$

- No use of MD5's $\mathcal{G}$ $((x \wedge z) \vee (y \wedge \sim z))$ or $\mathcal{I}$ $(y \oplus (x \vee \sim z))$

Processing Per Step

- Everything to right of "=" is input value to this step

```
for t = 0 upto 79
     A = E + (A << 5) + W_t + K_t + f(t,B,C,D)
     B = A
     C = B << 30
     D = C
     E = D
endfor
```

Comparison: SHA-1 vs. MD5

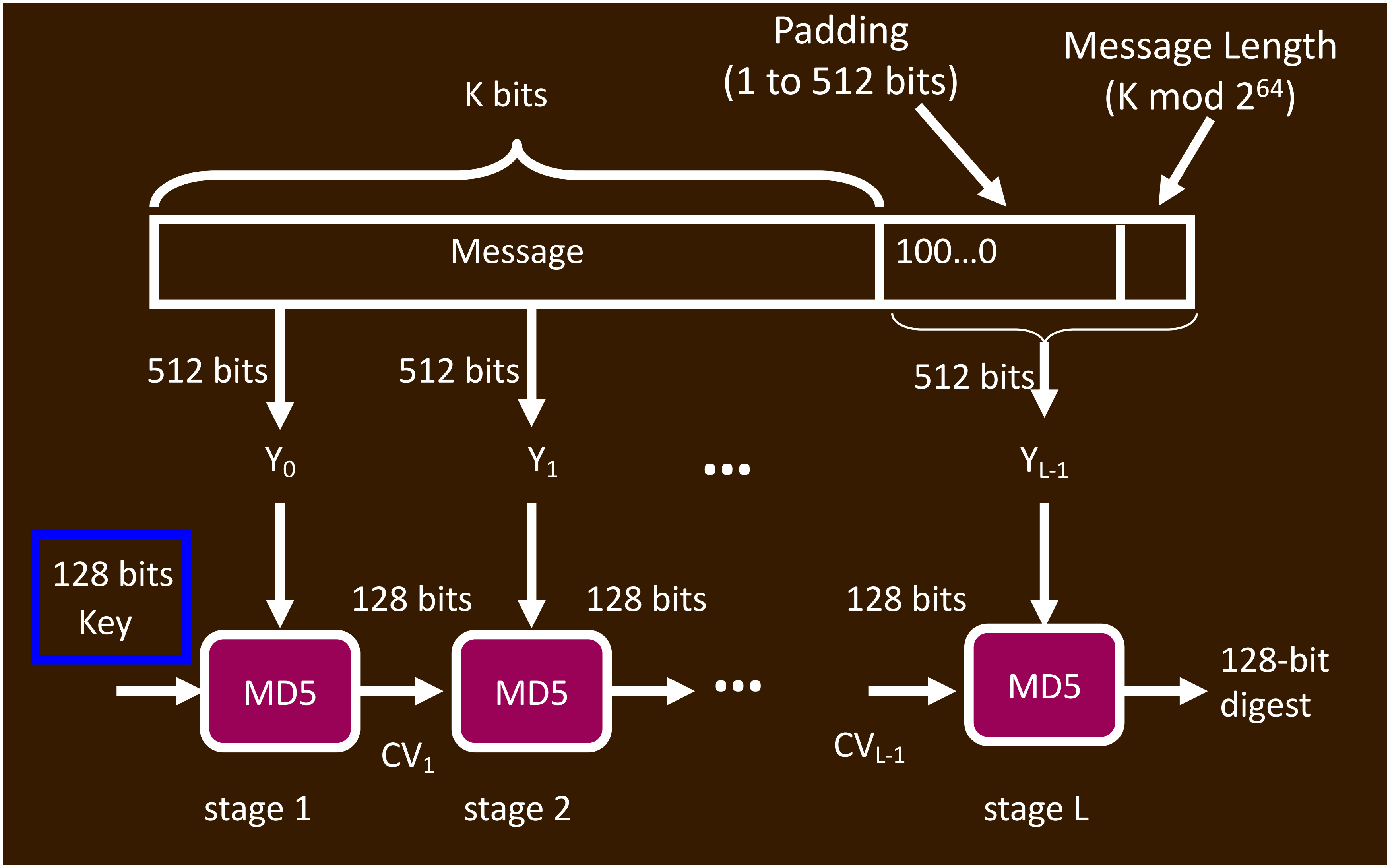
- SHA-1 is a stronger algorithm
  - brute-force attacks require on the order of 2^{80} operations vs. 2^{64} for MD5
- SHA-1 is about twice as expensive to compute
- Both MD-5 and SHA-1 are much faster to compute than DES

Security of SHA-1

- SHA-1
  - "Broken", but not yet cracked
  - Collisions in 2^{69} hash operations, much less than the brute-force attack of 2^{80} operations
  - Results were circulated in February 2005, and published in CRYPTO '05 in August 2005
- SHA-256, SHA-384, …

The Hashed Message Authentication Code (HMAC)

MD5 Revisited



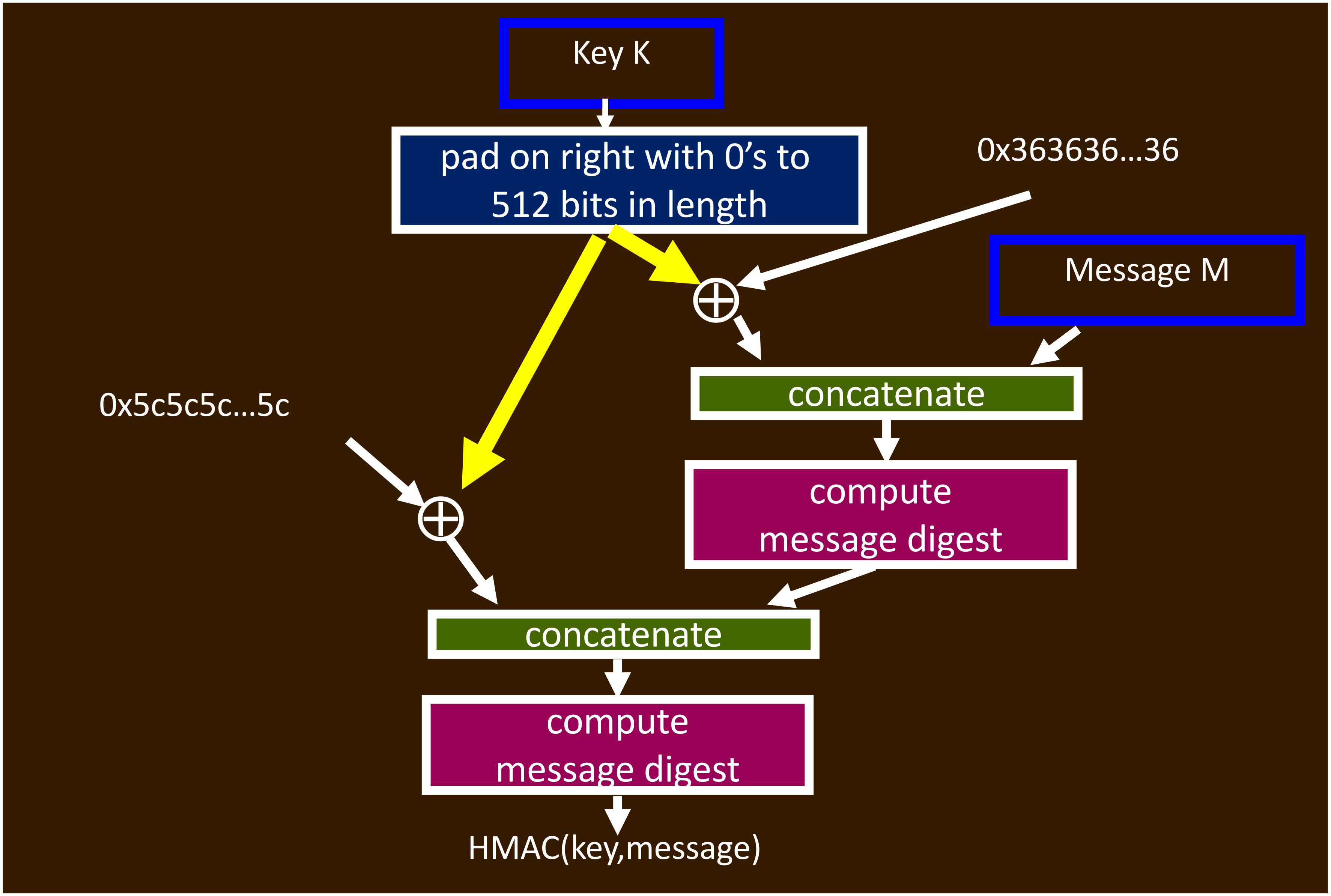
Extension Attacks

- Given M1, and secret key K, can easily concatenate and compute the hash:
  H(K|M1|padding)
- Given M1, M2, and H(K|M1|padding) easy to compute H(K|M1|padding|M2|newpadding) for some new message M2
- Simply use H(K|M1|padding) as the IV for computing the hash of M2|newpadding
  - does not require knowing the value of the secret key K

Extension Attacks (Cont'd)

- Many proposed solutions to the extension attack, but HMAC is the standard
- Essence: digest-inside-a-digest, with the secret used at both levels
- The particular hash function used determines the length of the message digest = length of HMAC output

HMAC Processing



Summary

- Hashing is fast to compute
- Has many applications (some making use of a secret key)
- Hash images must be at least 128 bits long
  - but longer is better
- Hash function details are tedious ☹
- HMAC protects message digests from extension attacks