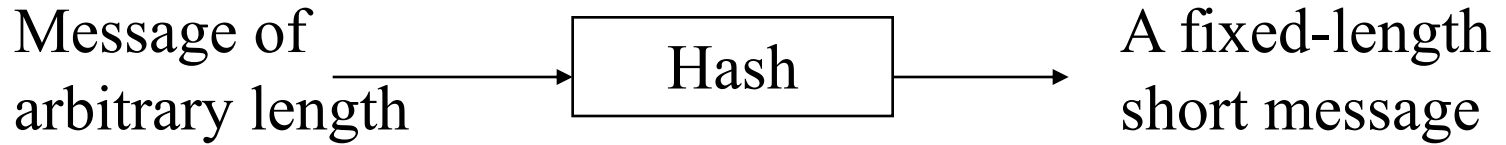


CS 5173/4173 Computer Security

Topic 4. Cryptographic Hash Functions

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
 - Weak collision resistance (free): Given $H(m)$, it's computationally infeasible to find m' such that $H(m') = H(m)$.
 - Strong collision resistance (free): 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
 - Loss of strong collision free property
 - Birthday paradox

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
 - $P(k \text{ people having } k \text{ different birthdays})$:
$$\begin{aligned} Q(365, k) &= (1 - 1/365) \times (1 - 2/365) \times (1 - 3/365) \times \dots \times \{1 - (k-1)/365\} \\ &= (364/365) \times (363/365) \times (362/365) \times \dots \times \{(365 - (k-1))/365\} \\ &= 365! / (365 - k)! 365^k \end{aligned}$$
 - $P(\text{at least two people have the same birthday})$:
$$P(365, k) = 1 - Q(365, k) \geq 0.5$$
 - 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,
 - What is the least value of k such that
 - There will be at least one duplicate
 - with probability $P(n,k) > 0.5$, ?

Birthday Paradox (Cont'd)

- Generalization of birthday paradox
 - $P(n,k) = 1 - \{n!/(n-k)!n^k\} \approx 1 - e^{-k*(k-1)/2n}$
 - For large n and k , to have $P(n,k) > 0.5$ with the smallest k , we have

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

- Example

- $1.18*(365)^{1/2} = 22.54$

Birthday Paradox (Cont'd)

- Implication for hash function H of length m
 - The hash value of an arbitrary input message is randomly distributed between 1 and 2^m
 - What is the least value of k such that
 - If we hash k messages, the probability that at least two of them have the same hash is larger than 0.5?

$$k \approx \sqrt{n} = \sqrt{2^m} = 2^{m/2}$$

– Birthday attack

- Choose $m \geq 128$

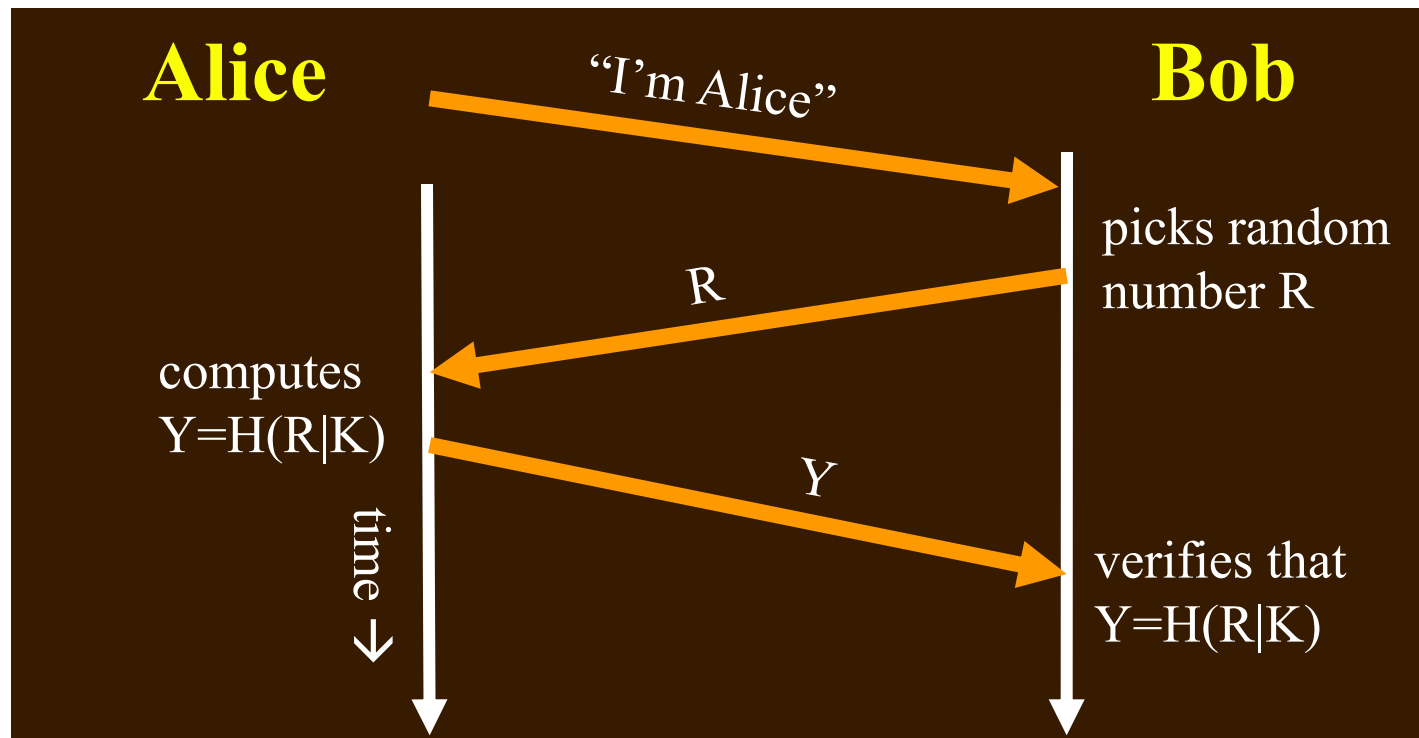
Hash Function Applications

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 $H(F)$
- 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)

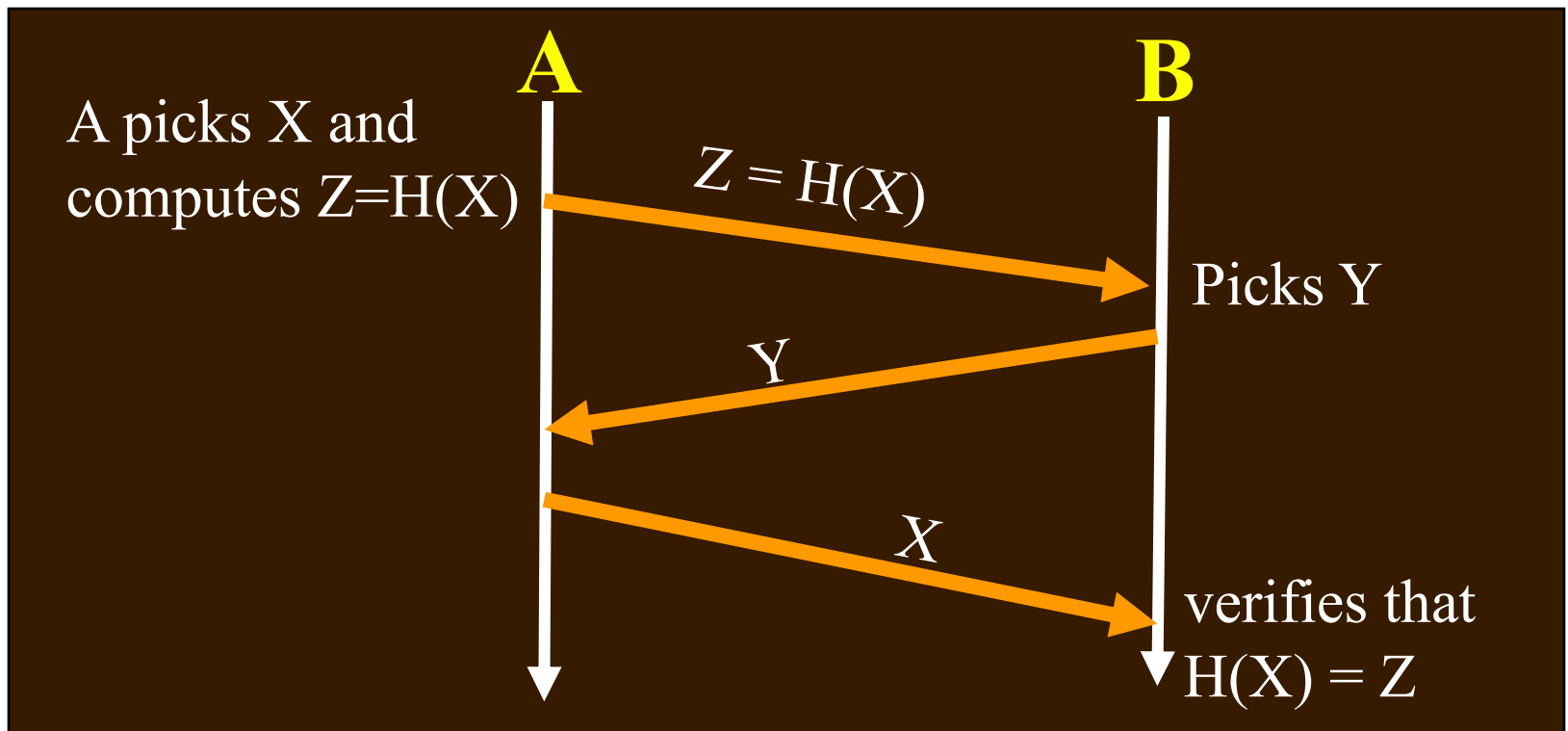
- Why not just send...
 - ...K, in plaintext?
 - ...H(K)? , i.e., what's the purpose of R?

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:



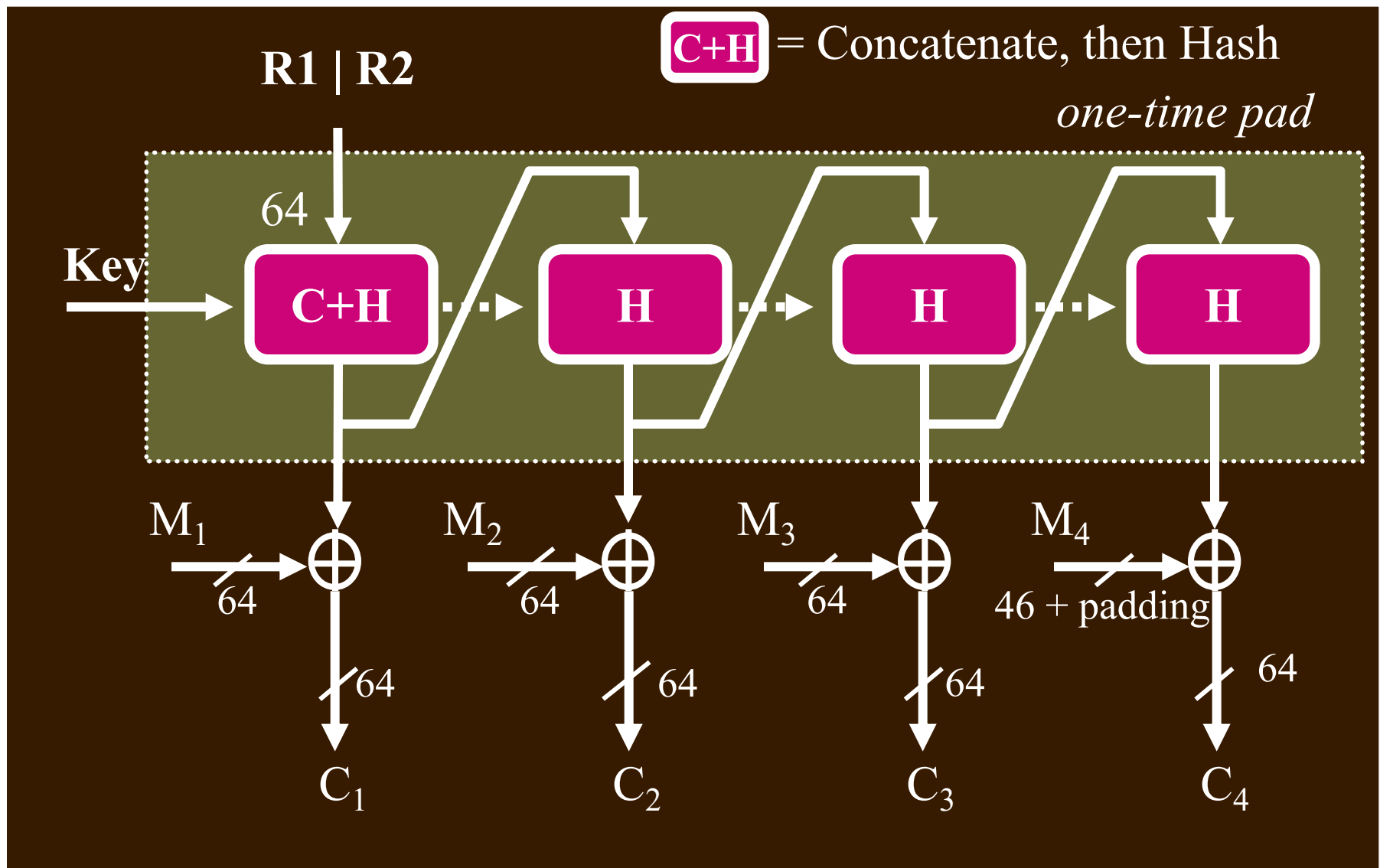
- Can either A or B successfully cheat now?

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:
 The set of possible values for X is **small**?

Application: Message Encryption

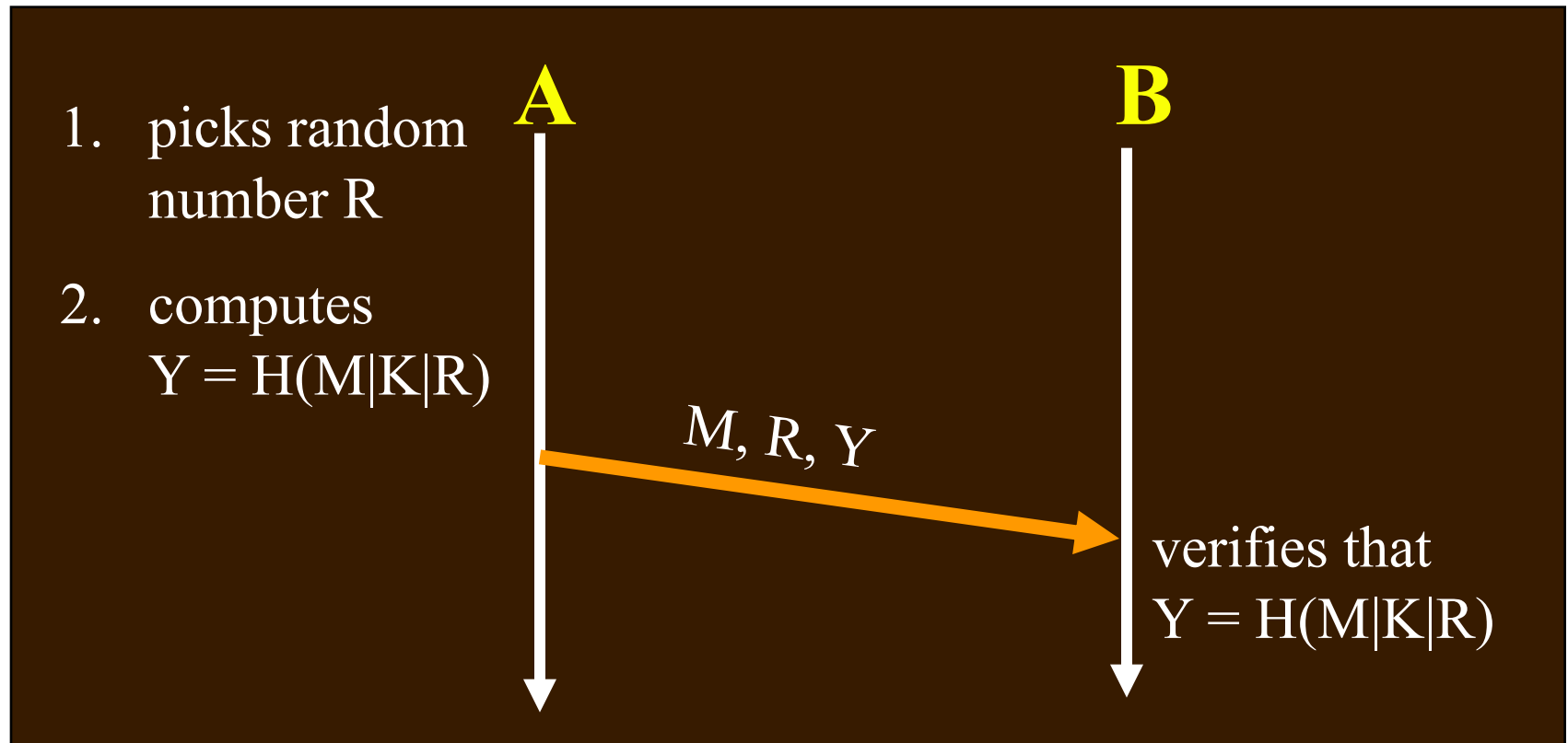
- Assume A and B share a secret key K
 - but don't want to just use encryption of the message with K
- A sends B the (encrypted) random number R_1 ,
B sends A the (encrypted) random number R_2
- And then...



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

Application: Message Integrity

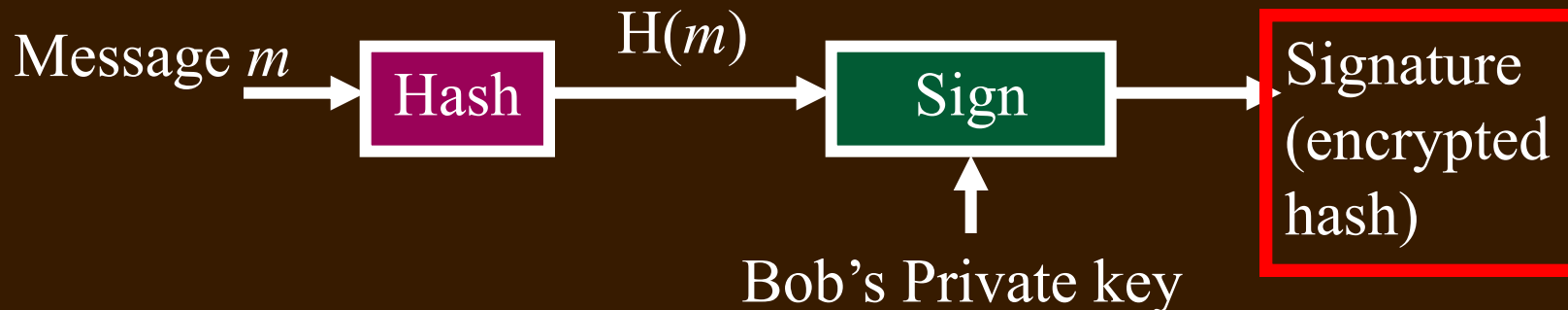
- A wishes to authenticate (but not encrypt) a message M (and A, B share secret key K)



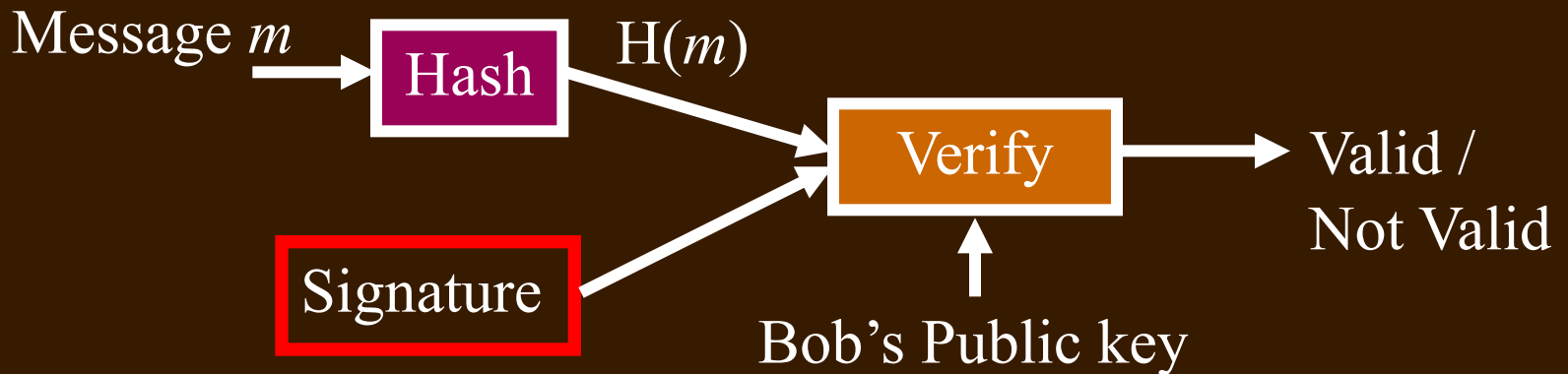
- Why is R needed?
- Why is K needed?

Application: Digital Signatures

Generating a signature

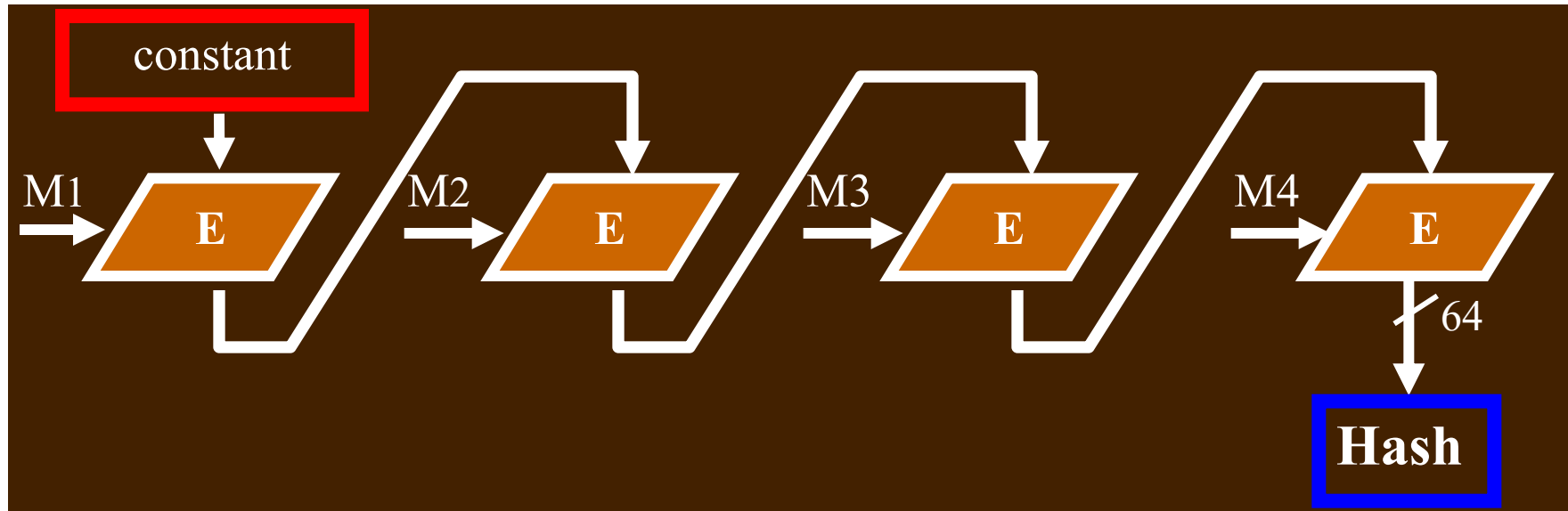


Verifying a signature



- Only **one party** (Bob) knows the **private** key

Is Encryption a Good Hash Function?



- Building hash using block chaining techniques
 - Encryption block size may be too short (DES=64)
 - Birthday attack
 - Expensive in terms of computation time

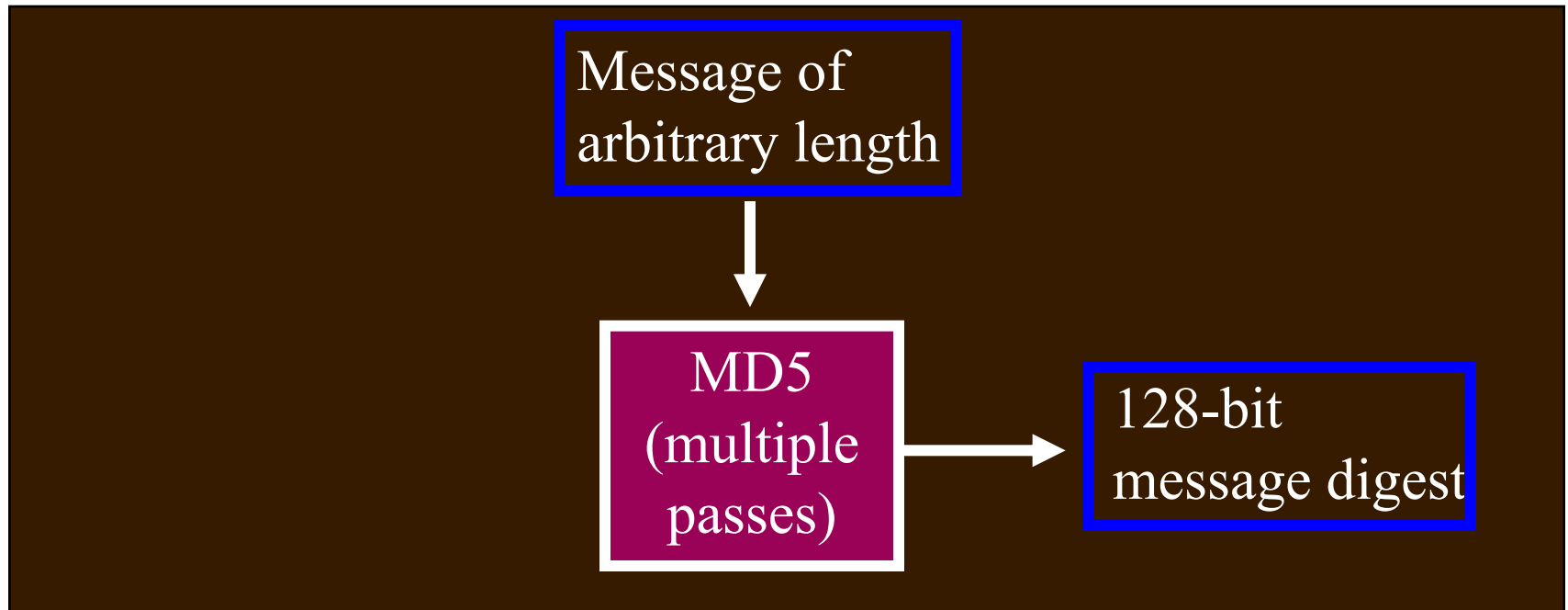
Modern Hash Functions

- MD5
 - Previous versions (i.e., MD2, MD4) have weaknesses.
 - Broken; collisions published in August 2004
 - Previous versions are too weak to be used for serious applications
 - As of 2019, **one quarter** of widely used content management systems were reported to still use MD5 for password hashing
(Cimpanu, Catalin. "A quarter of major CMSs use outdated MD5 as the default password hashing scheme". ZDNet. 2019.)
- SHA (Secure Hash Algorithm)
 - Weaknesses were found
- SHA-1
 - Cracked in 2017 (*CWI Amsterdam* and *Google*)
 - Collisions in 2^{69} hash operations, much less than the birthday attack of 2^{80} operations
- SHA-256, SHA-384, ...

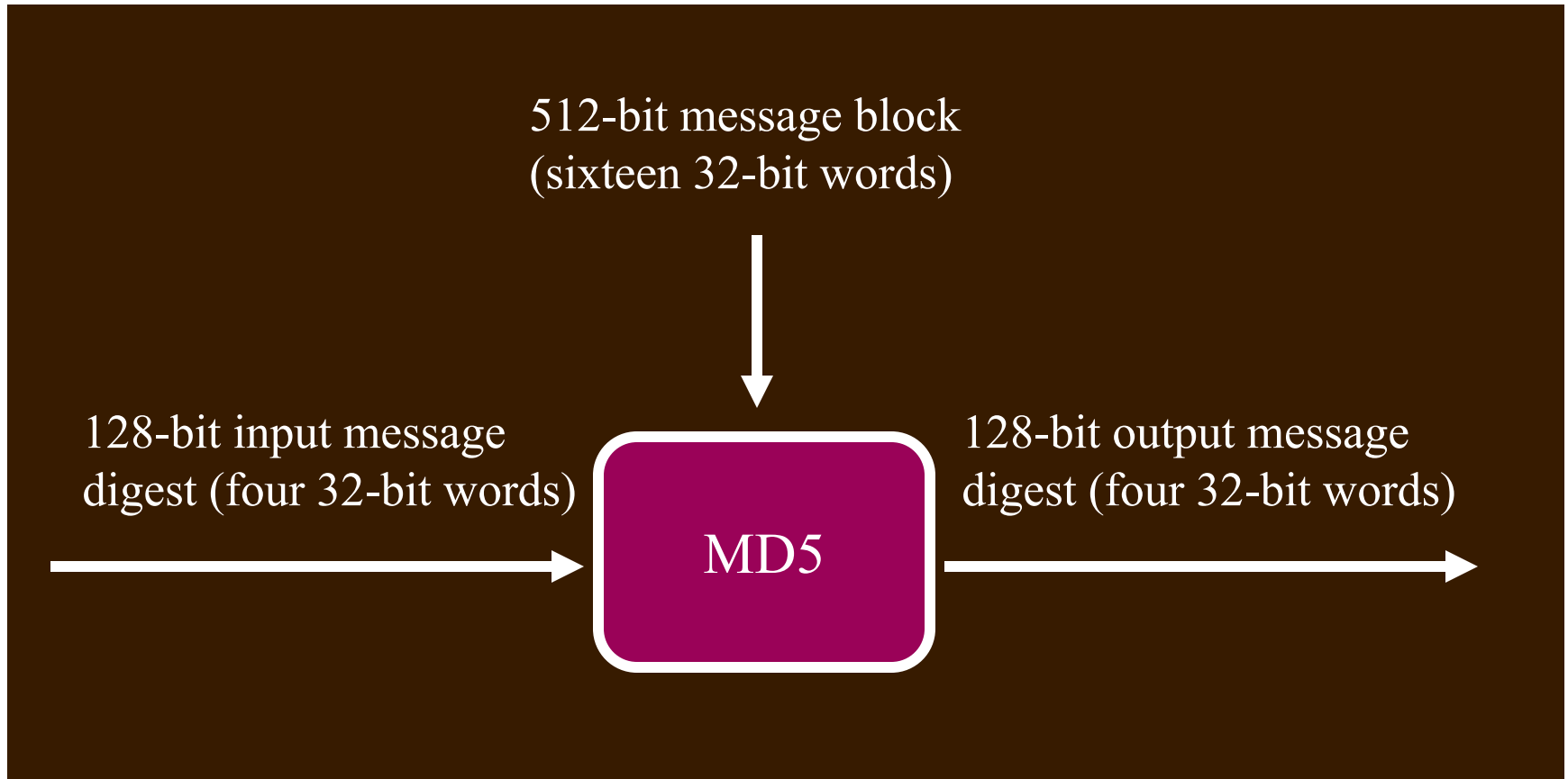
The MD5 Hash Function

MD5: Message Digest Version 5

- MD5 at a glance

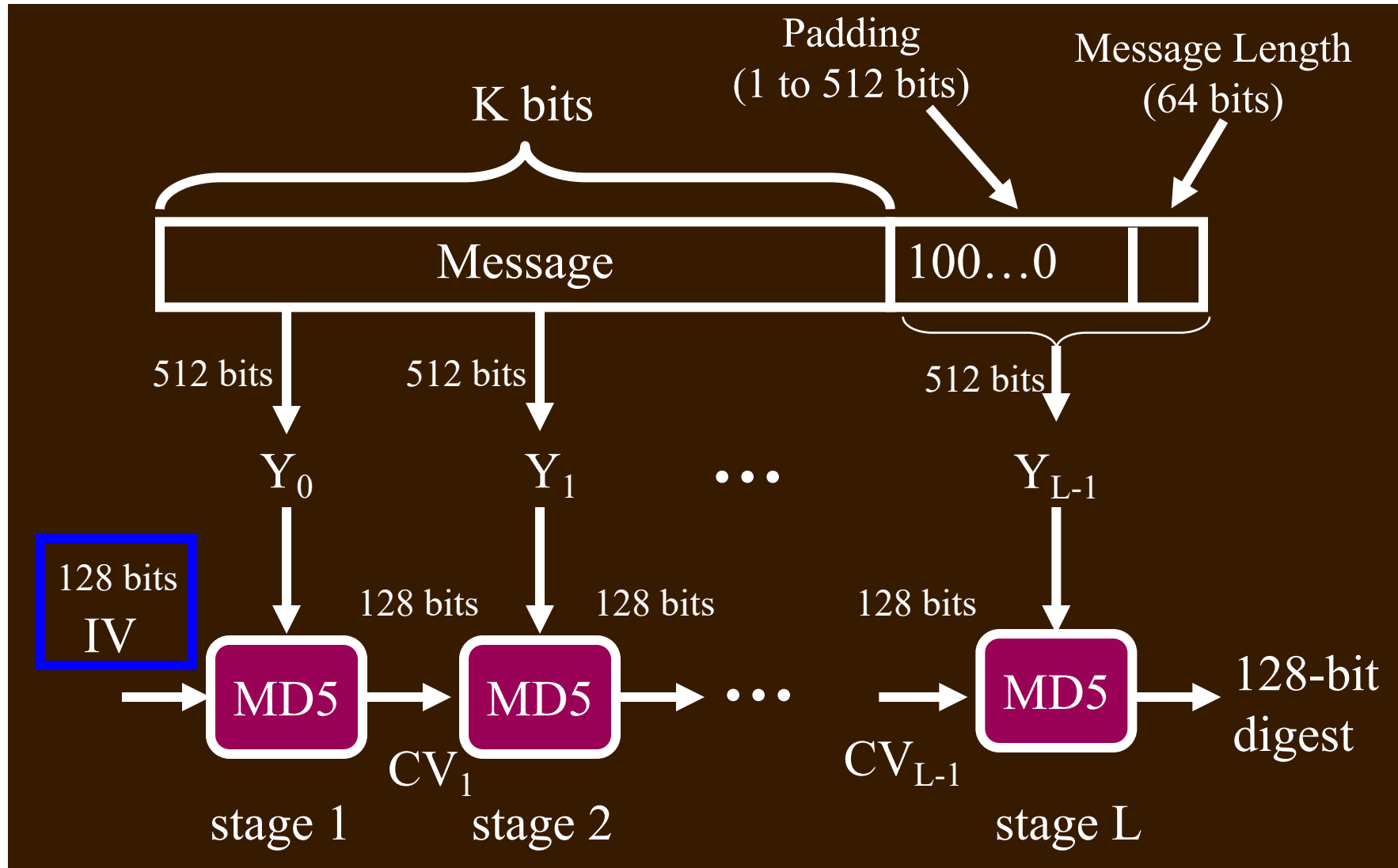


Processing of A Single Block



Called a compression function

MD5: A High-Level View



Padding

- There is padding for MD5, and padded messages must be **multiples of 512 bits**
- To original message M, add padding bits **“10...0”**
 - enough 0’s so that resulting total length is 64 bits less than a multiple of 512 bits
- Append L (original length of M), represented in 64 bits, to the padded message
- Footnote: the bytes of each 32-bit word are stored in **little-endian order** (LSB to MSB)

Padding... (cont'd)

- How many 0's if length K of Message =
- $n * 512$?
- $n * 512 - 64$?
- $n * 512 - 65$?

Preliminaries

- The four 32-bit words of the output (the *digest*) are referred to as **d0, d1, d2, d3**
- Initial values (in little-endian order)
 - **d0** = 0x67452301
 - **d1** = 0xEFCDAB89
 - **d2** = 0x98BADCFE
 - **d3** = 0x10325476
- The sixteen 32-bit words of each message block are referred to as **m0, ..., m15**
 - (16*32 = 512 bits in each block)

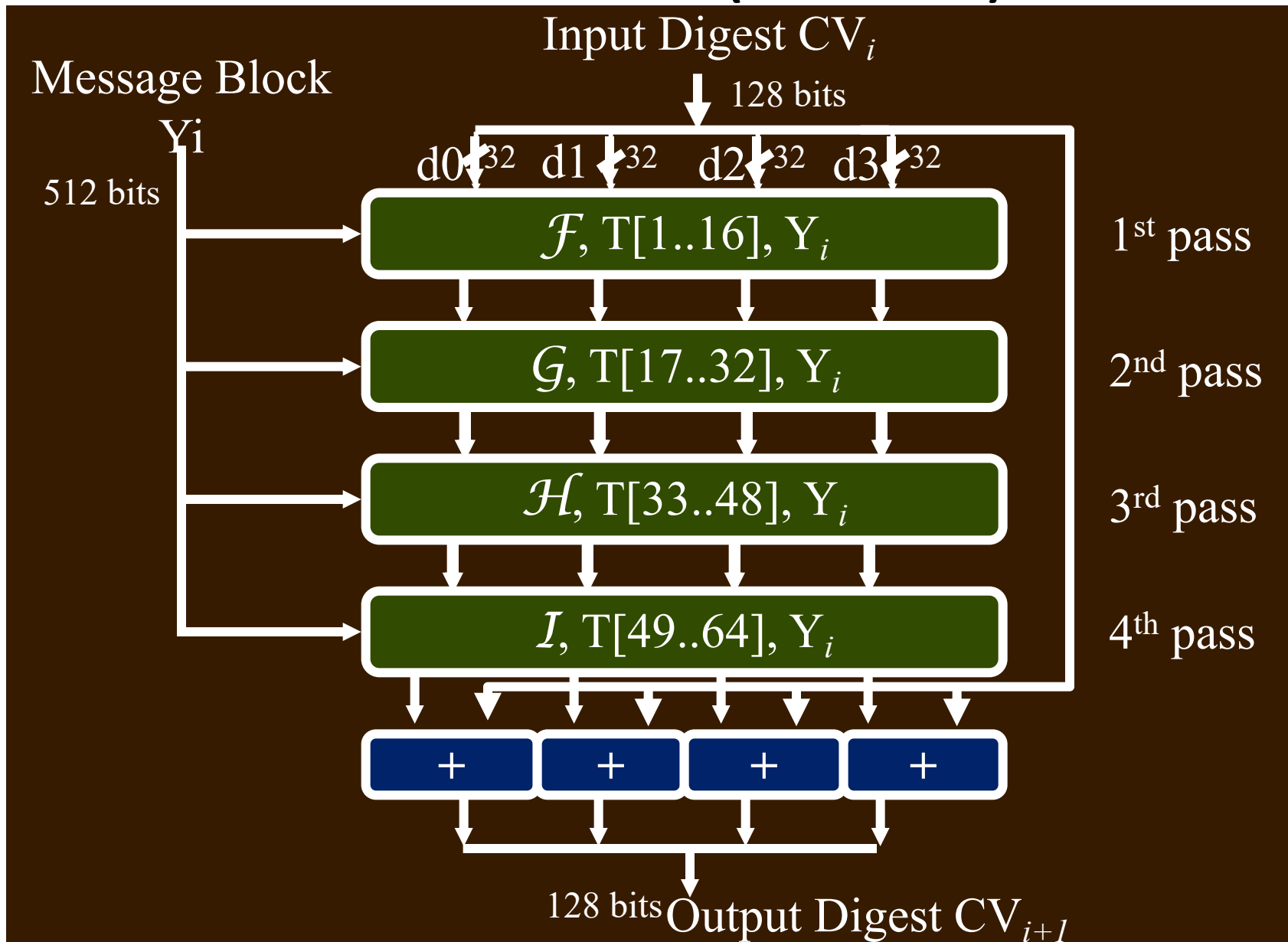
Notation

- $\sim x$ = bit-wise complement of x
- $x \wedge y, x \vee y, x \oplus y$ = bit-wise AND, OR, XOR of x and y
- $x \ll y$ = 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 Y_i contains **16 32-bit words**:
 - $m_0 m_1 m_2 \dots m_{15}$
- A block is processed in **4** consecutive passes, each modifying the MD5 buffer d_0, \dots, d_3 .
 - Called $\mathcal{F}, \mathcal{G}, \mathcal{H}, \mathcal{I}$
- Each pass uses one-fourth of a 64-element table of constants, $T[1\dots 64]$
 - $T[i] = \lfloor 2^{32} * \text{abs}(\sin(i)) \rfloor$, represented in 32 bits
- Output digest = input digest + output of 4th pass

Overview (Cont'd)



1st Pass of MD5

- $\mathcal{F}(x,y,z) \stackrel{\text{def}}{=} (x \wedge y) \vee (\sim x \wedge z)$
- 16 processing steps (the table below only shows the first 5 steps), producing $\mathbf{d}_0 \dots \mathbf{d}_3$ output:

$$\mathbf{d}_i = \mathbf{d}_j + (\mathbf{d}_k + \mathcal{F}(\mathbf{d}_l, \mathbf{d}_m, \mathbf{d}_n) + \mathbf{m}_o + T_p) \ll s$$

– values of subscripts, in this order

<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>	<i>p</i>	<i>s</i>
0	1	0	1	2	3	0	1	7
3	0	3	0	1	2	1	2	12
2	3	2	3	0	1	2	3	17
1	2	1	2	3	0	3	4	22
0	1	0	1	2	3	4	5	7

2nd Pass of MD5

- $G(x,y,z) \stackrel{\text{def}}{=} (x \wedge z) \vee (y \wedge \sim z)$
- Form of processing (16 steps):

$$\mathbf{d}_i = \mathbf{d}_j + (\mathbf{d}_k + G(\mathbf{d}_l, \mathbf{d}_m, \mathbf{d}_n) + \mathbf{m}_o + \mathbf{T}_p) \ll s$$

<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>	<i>p</i>	<i>s</i>
0	1	0	1	2	3	1	17	5
3	0	3	0	1	2	6	18	9
2	3	2	3	0	1	11	19	14
1	2	1	2	3	0	0	20	20
0	1	0	1	2	3	5	21	5

3rd Pass of MD5

- $\mathcal{H}(x,y,z) \stackrel{\text{def}}{=} (x \oplus y \oplus z)$
- Form of processing (16 steps):

$$\mathbf{d}_i = \mathbf{d}_j + (\mathbf{d}_k + \mathcal{H}(\mathbf{d}_l, \mathbf{d}_m, \mathbf{d}_n) + \mathbf{m}_o + T_p) \lll s$$

<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>	<i>p</i>	<i>s</i>
0	1	0	1	2	3	5	33	4
3	0	3	0	1	2	8	34	11
2	3	2	3	0	1	11	35	16
1	2	1	2	3	0	14	36	23
0	1	0	1	2	3	1	37	4

4th Pass of MD5

- $I(x,y,z) \stackrel{\text{def}}{=} y \oplus (x \vee \sim z)$

- Form of processing (16 steps):

$$\mathbf{d}_i = \mathbf{d}_j + (\mathbf{d}_k + I(\mathbf{d}_l, \mathbf{d}_m, \mathbf{d}_n) + \mathbf{m}_o + T_p) \ll s$$

<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>	<i>p</i>	<i>s</i>
0	1	0	1	2	3	0	49	6
3	0	3	0	1	2	7	50	10
2	3	2	3	0	1	14	51	15
1	2	1	2	3	0	5	52	21
0	1	0	1	2	3	12	53	6

- Output of this pass added to input CV

Logic of Each Step

- Within each pass, each of the 16 words of the message block is used exactly once
 - Pass 1, m_i are used in the order of i
 - Pass 2, in the order of $\rho_2(i)$, where $\rho_2(i) = (1+5i) \wedge 15$
 - Pass 3, in the order of $\rho_3(i)$, where $\rho_3(i) = (5+3i) \wedge 15$
 - Pass 4, in the order of $\rho_4(i)$, where $\rho_4(i) = 7i \wedge 15$
- Each word of $T[i]$ is used exactly once throughout all passes
- Number of bits s to rotate to get d_i
 - Pass 1, $s(d_0)=7, s(d_1)=22, s(d_2)=17, s(d_3)=12$
 - Pass 2, $s(d_0)=5, s(d_1)=20, s(d_2)=14, s(d_3)=9$
 - Pass 3, $s(d_0)=4, s(d_1)=23, s(d_2)=16, s(d_3)=11$
 - Pass 4, $s(d_0)=6, s(d_1)=21, s(d_2)=15, s(d_3)=10$

(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
 - 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

Preprocessing of a Block

- Let 512-bit block be denoted as sixteen 32-bit words $\mathbf{W}_0.. \mathbf{W}_{15}$
- Preprocess $\mathbf{W}_0.. \mathbf{W}_{15}$ to derive an additional sixty-four 32-bit words $\mathbf{W}_{16}.. \mathbf{W}_{79}$, as follows:

for $16 \leq t \leq 79$

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

Block Processing

- Consists of **80 steps!** (vs. 64 for MD5)
- Inputs for each step $0 \leq t \leq 79$:
 - 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

Constants K_t

- Only 4 values (represented in 32 bits), derived from $2^{30} * i^{1/2}$, for $i = 2, 3, 5, 10$
 - for $0 \leq t \leq 19$: $K_t = 0x5A827999$
 - for $20 \leq t \leq 39$: $K_t = 0x6ED9EBA1$
 - for $40 \leq t \leq 59$: $K_t = 0x8F1BBCDC$
 - for $60 \leq t \leq 79$: $K_t = 0xCA62C1D6$

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
$0 \leq t \leq 19$	$(B \wedge C) \vee (\sim B \wedge D)$	$\mathcal{F} = (x \wedge y) \vee (\sim x \wedge z)$
$20 \leq t \leq 39$	$B \oplus C \oplus D$	$\mathcal{H} = x \oplus y \oplus z$
$40 \leq t \leq 59$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	
$60 \leq t \leq 79$	$B \oplus C \oplus D$	$\mathcal{H} = x \oplus y \oplus z$

- 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) + Wt + Kt + f(t,B,C,D)
  B = A
  C = B << 30
  D = C
  E = D
endfor
```

Comparison: SHA-1 vs. MD5

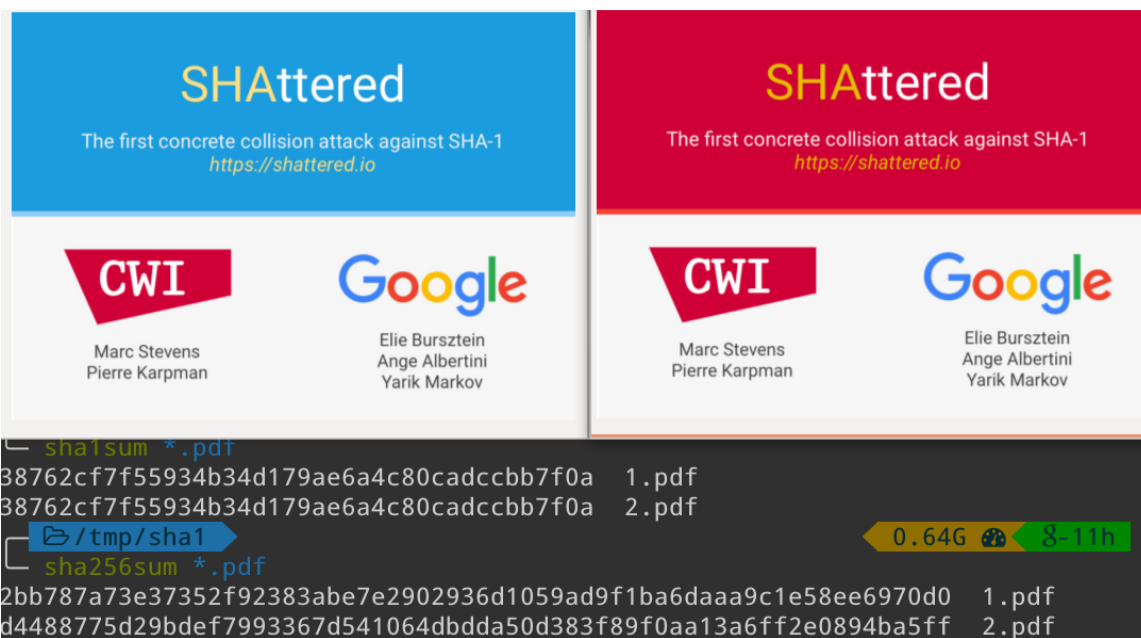
- SHA-1 is a stronger algorithm
 - brute-force attacks require on the order of 2^{80} operations vs. 2^{64} for MD5
- SHA-1 is about twice as expensive to compute
- Both MD-5 and SHA-1 are **much** faster to compute than DES

Security of SHA-1

- Theoretical attacks have been known since 2005
 - Brute-force attack (Birthday attack) needs 2^{80} hash operations
 - Cryptanalysis attacks can find collisions by using 2^{69} hash operations.
 - Results were circulated in February 2005, and published in CRYPTO '05 in August 2005
 - 2^{69} is still too huge to make it a practical attack
 - The time complexity of this attack is further reduced to 2^{63} hash operations

Security of SHA-1 (Cont'd)

- In 2017, the SHAttered attack was proposed by CWI Institute in Amsterdam and Google.
 - This is the first concrete collision attack against SHA1 : two PDF files have the same hash value.



The image displays the SHAttered attack announcement graphic and a terminal window showing the results of the attack.

SHAttered
The first concrete collision attack against SHA-1
<https://shattered.io>

CWI
Marc Stevens
Pierre Karpman

Google
Elie Bursztein
Ange Albertini
Yarik Markov

SHAttered
The first concrete collision attack against SHA-1
<https://shattered.io>

CWI
Marc Stevens
Pierre Karpman

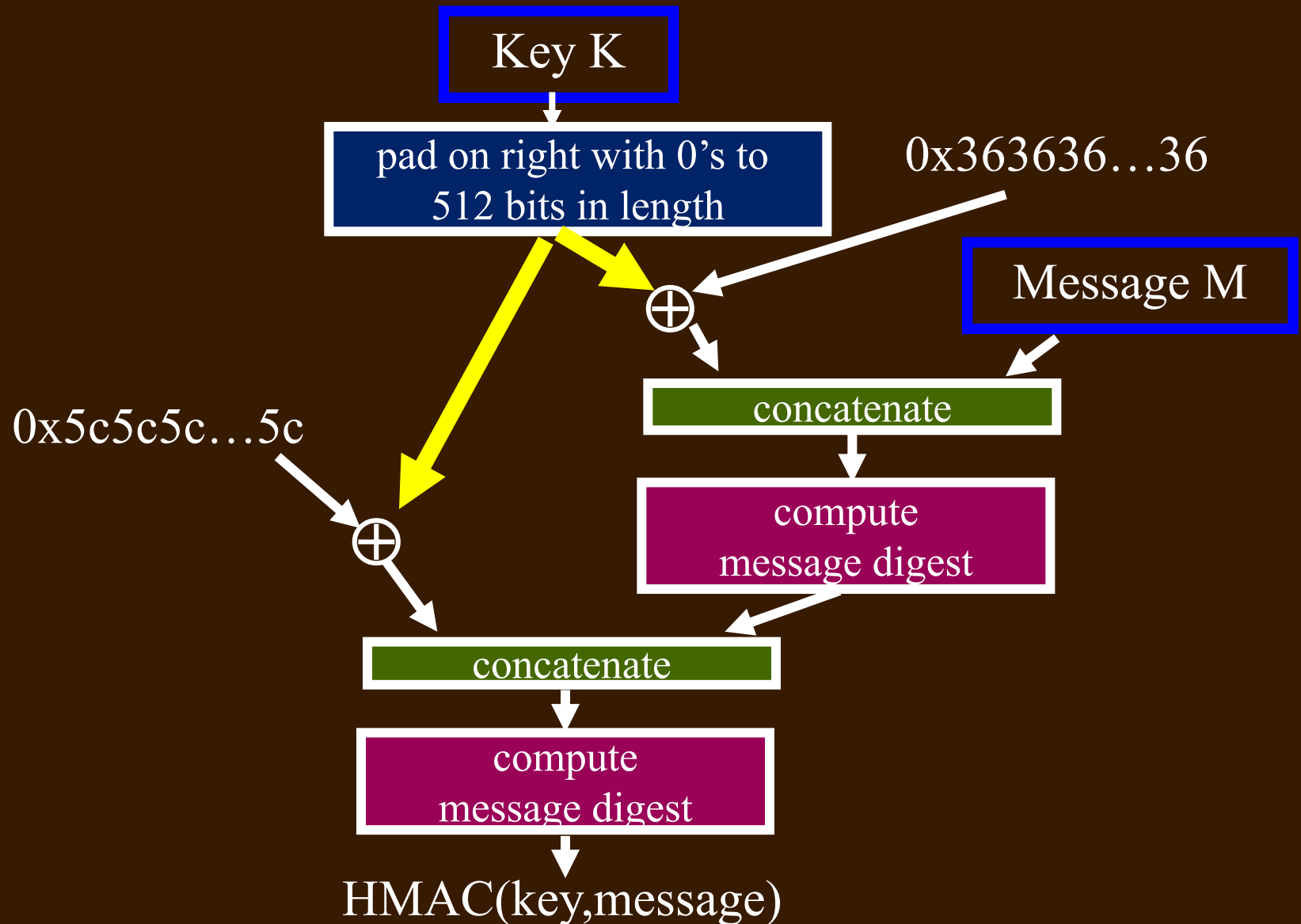
Google
Elie Bursztein
Ange Albertini
Yarik Markov

```
sha1sum *.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a 1.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a 2.pdf
/tmp/sha1 0.64G 8-11h
sha256sum *.pdf
2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0 1.pdf
d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff 2.pdf
```


The Hashed Message Authentication Code (HMAC)


- HMAC generates the message digest of both a message and a key
- Essence: digest-inside-a-digest, with the secret used at both levels
- The particular hash function used determines the 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 generates the message digest of both a message and a key

 Please read the slide
[lecture6.pdf](#) before next class.