CSCI 454/554 Computer and Network Security

Topic 4. Cryptographic Hash Functions

Outline

- 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 then 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 (preimage resistant): Given H(m) but not m, it's computationally infeasible to find m
  - Weak collision resistant (2-nd preimage resistant): Given H(m), it's computationally infeasible to find m' such that H(m') = H(m).
  - Strong collision resistant (collision resistant): 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,
  - 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
  - $P(n,k) \approx 1 - e^{-k*(k-1)/2n}$
  - 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
  - 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 $m \geq 128$

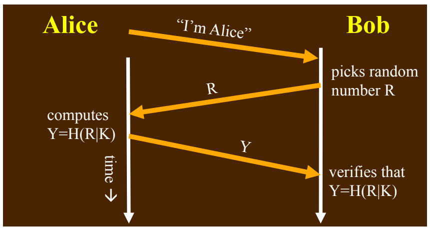
Hash Function Applications

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)
- 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)

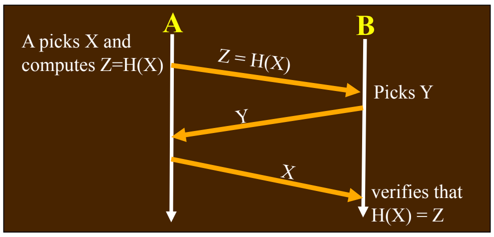
- Why not just send…
  - …K, in plaintext?
  - …H(K)? , i.e., what's the purpose of R?

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:



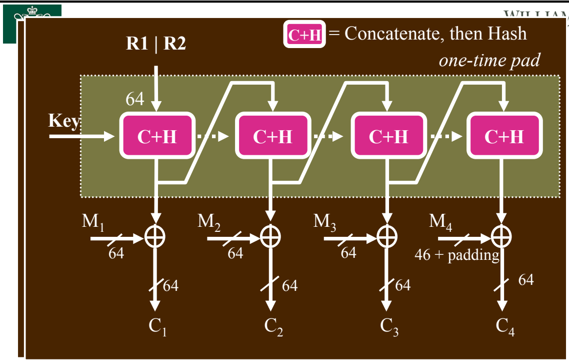
- Can either A or B successfully cheat now?

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

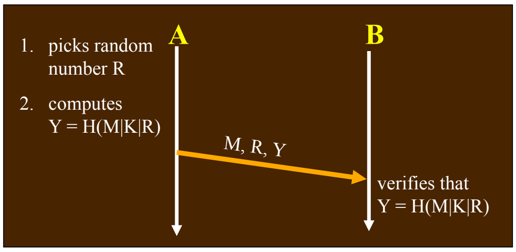
- Assume A and B share a secret key K
  - but don't want to just use encryption of the message with K
- A sends B the (encrypted) random number R1,
  B sends A the (encrypted) random number R2
- And then…



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

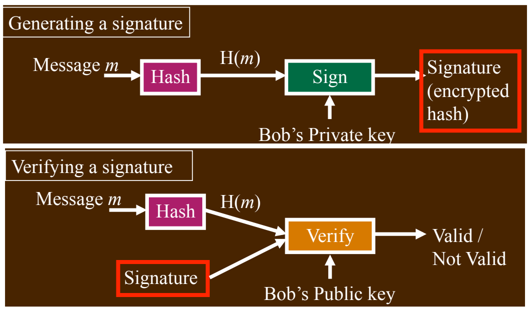
Application: Message Authentication

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



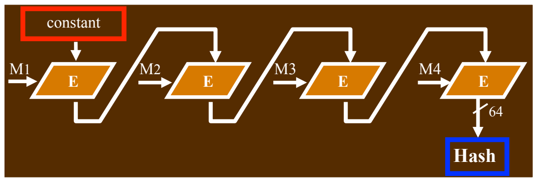
- Why is R needed? Why is K needed?

Application: Digital Signatures



- Only one party (Bob) knows the private key

Is Encryption a Good Hash Function?



- Building hash using block chaining techniques
  - Encryption block size may be too short (DES=64)
    - Birthday attack
  - Can construct a message with a particular hash fairly easily
  - Extension attacks

Hash Using Block Chaining Techniques

- Meet-in-the-middle attack
  - Get the correct hash value G
  - Construct any message in the form $Q_1, Q_2, \ldots, Q_{n-2}$
  - Compute $H_i = E_{Qi}(H_{i-1})$ for $1 \le i \le (n-2)$.
  - Generate $2^{m/2}$ random blocks; for each block X, compute $E_X(H_{n-2})$.
  - Generate $2^{m/2}$ random blocks; for each block Y, compute $D_Y(G)$.
  - With high probability there will be an X and Y such that $E_X(H_{n-2}) = D_Y(G)$.
  - Form the message $Q_1, Q_2, \ldots, Q_{n-2}$, X, Y. It has the hash value G.

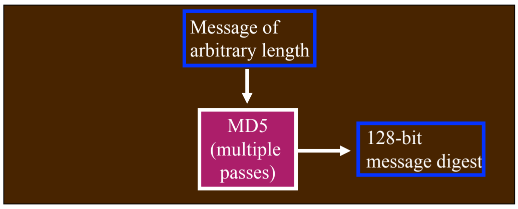
Modern Hash Functions

- 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-2 (SHA-256, SHA-384, …)

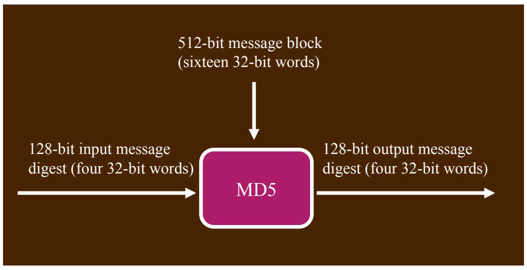
MD5 Hash Function

MD5: Message Digest Version 5

- MD5 at a glance

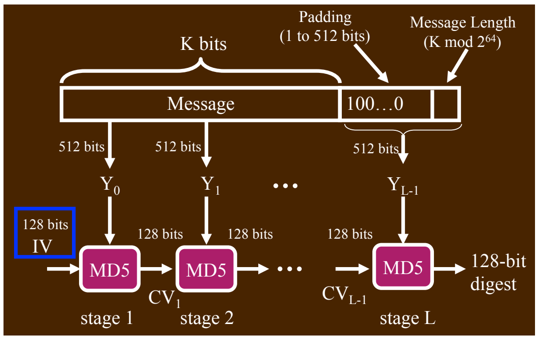


Processing of A Single Block



Called a compression function

MD5: A High-Level View



Padding

- There is always 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 of M =
- 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)

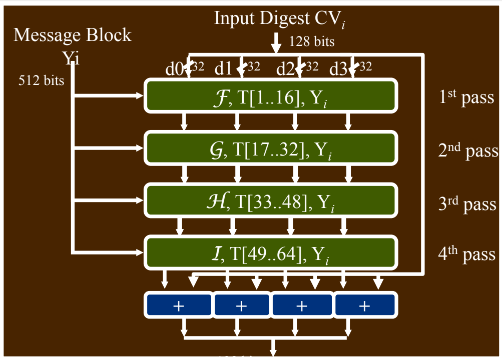
Notation

- $\sim x$ = bit-wise complement of x
- $x \wedge y$, $x \vee y$, $x \oplus y$ = bit-wise AND, OR, XOR of x and y
- $x << y$ = 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*:
  - $m_0\ m_1\ m_2\ \ldots\ m_{15}$
- A block is processed in 4 consecutive passes, each modifying the MD5 buffer $d_0, \ldots, d_3$.
  - 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] = $\lfloor 2^{32}*\text{abs}(\sin(i)) \rfloor$, represented in 32 bits
- Output digest = input digest + output of 4th pass

Overview (Cont'd)



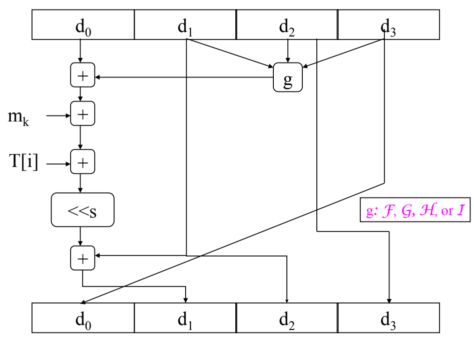
1st Pass of MD5

- $\mathcal{F}(x,y,z) \overset{\text{def}}{=} (x \wedge y) \vee (\sim x \wedge z)$
- 16 processing steps, producing $\mathbf{d_0..d_3}$ output:
  $\mathbf{d}_i = \mathbf{d}_j + (\mathbf{d}_k + \mathcal{F}(\mathbf{d}_l, \mathbf{d}_m, \mathbf{d}_n) + \mathbf{m}_o + \mathrm{T}_p) << s$
  - values of subscripts, in this order

i	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*s*
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

⋮

Logic of Each Step



Logic of Each Step (Cont'd)

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

2^{nd} Pass of MD5

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

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

i	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*s*
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

⋮

3^{rd} 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	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*s*
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

⋮

4^{th} Pass of MD5

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

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

i	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*s*
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 MD

(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

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

Big Endian vs. Little Endian

- A 32-bit word can be saved in 4 bytes
  - For instance, $90AB12CD_{16}$

- Big Endian

Address	Value
1000	90
1001	AB
1002	12
1003	CD

- Little Endian

Address	Value
1000	CD
1001	12
1002	AB
1003	90

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 $16 \le t \le 79$

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

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

Constants K_t

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

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) + 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

- SHA-1
  - output 160 bits
  - "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
  - Considered insecure for collision resistance
  - One-way property still holds
- SHA-2(SHA-224, SHA-256, SHA-384, SHA-512…)

SHA-3 is coming

- NIST is having an ongoing competition for SHA-3, the next generation of standard hash algorithms
  - 2007: Request for submissions of new hash functions
  - 2008: Submissions deadline. Received 64 entries. Announced first-round selections of 51 candidates.
  - 2009: After First SHA-3 candidate conference in Feb, announced 14 Second Round Candidates in July.
  - 2010: After one year public review of the algorithms, hold second SHA-3 candidate conference in Aug. Announced 5 Third-round candidates in Dec.
  - 2011: Public comment for final round
  - 2012: Held Final hash candidate conference on March 22-23. Draft standard, wait for comments, and submit recommendation.
- The winning algorithm, Keccak, was created by Guido Bertoni, Joan Daemen and Gilles Van Assche of STMicroelectronics and Michaël Peeters of NXP Semiconductors.

Hashed Message Authentication Code (HMAC)

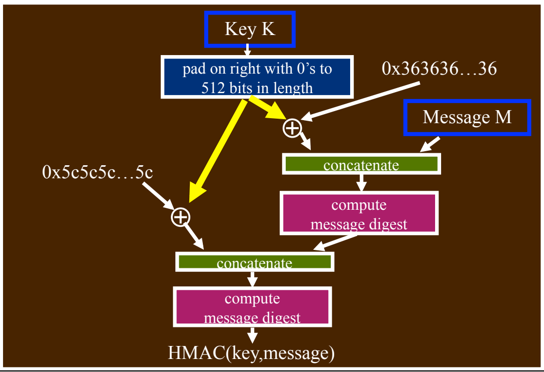
Extension Attacks

- 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



Security of HMAC

At high level, $HMAC_K[M] = H(K \| H(K \| M))$

- If used with a secure hash functions (e.g., SHA-256) and according to the specification (key size, and use correct output), no known practical attacks against HMAC

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