

Computer Security

Hashes and Message Digests

Hash

- hash (or message digest) is a one-way function
- one-way as impractical, given output, to determine, input
- at least, that's the intent
- takes arbitrary input and gives fixed length output
- does not use a key

Cryptographic Security

- for a given hash function must be computationally infeasible
 - to find message that hashes to given output
 - to find two messages that hash to same output

Randomness

- eg., for 1000 different hash results
 - any given bit should be on about half the time
 - each output should have about half the bits on
 - similar input should not lead to similar output

Same Digest

- given arbitrary message length many messages will have same hash
- eg., 1000 bit message, 128 bit digest, 2^{872} messages for every digest
- would need to try about 2^{128} messages before finding one that maps to given digest

5

How Many Bits

- how long should a digest be?
- if digest has m bits then $2^{(m/2)}$ messages chosen at random will probably yield two with same digest
- it is not considered feasible to search 2^{64} messages, given current capabilities
- so 128 bits enough for digest

6

Why Two Messages?

- if only worried about finding message with same digest, then 64 bits enough
- why worry about being able to find pair?

7

Example

- Alice wants to fire Fred
- Alice's secretary Bob writes letter
- Alice generates hash
- Bob (friend of Fred) substitutes other message, with same digest

8

Algorithm Construction

- hash algorithms hard to understand
- apparently unrelated steps
- why this particular shuffling?
- why not something simpler and just as secure?

Algorithm Construction

- combine lots of perverse operations
- if any patterns are detected in output, try again

Safety

- digest should be easy to compute
- what then is the minimal digest function?
- who cares, better to overdo it then leave holes
- designers probably find the safe point, then do a bit more to be sure

Uses of Hash Function

- authentication
- MIC
- encryption

Authentication

- parties still need to share a secret, as in secret key authentication
- remember message digests are not reversible, so no decryption
- Alice sends a challenge to Bob
- Bob takes challenge, concatenates secret, transmits digest of result
- Alice redoes operation to check

13

Protocol

- Alice -> Bob : ra
- Bob-> Alice : MD(Kab | ra)

14

MIC

- for message m what about MD(m) as MIC?
- no good, as anyone can calculate, so can replace message and MIC
- use similar approach to authentication - concatenation
- concatenate shared secret Kab with message and use MD(Kab | m) as MIC

15

Does it Work?

- almost
- popular digest algorithms have some idiosyncracies
- allow attacker to compute MIC of different message, given m and its MIC

16

Example: MD4, MD5, SHS

- message is padded to multiple of 512 bits
- pad includes message length
- message digested from left to right in 512 bit chunks
- to compute digest at chunk n need chunk and digest at chunk n-1

17

Attack

- Alice sends message to Bob
- Carol intercepts and wants to add "Give Carol a pay rise"
- Alice has transmitted m and $MD(K_{ab} || m)$
- Carol can see both
- Carol concatenates her addition to m
- then calculates new MD

18

Solution 1

- put secret at end rather than beginning
- sometimes criticised as if intermediate block cracked, easier to find final block
- but if algorithm secure this shouldn't be a problem

19

Solution 2

- use half the bits of digest as MIC
- attacker cannot then continue digest (1 in 2^{64} chance of guessing)

20

Encryption

- encryption with digest algorithm is easy
- but what about decryption?
- message digest algorithms are not reversible
- so need scheme that does encryption and decryption with MD algorithm running in same direction

21

One-Time Pad

- message digest algorithm can be used to generate pseudorandom bit stream
- parties need to share a secret key K_{ab}
- Alice computes $MD(K_{ab})$
- result is first block of bit stream, b_1
- $MD(K_{ab} \parallel b_1)$ is used as b_2
- $b_i = MD(K_{ab} \parallel b_{i-1})$

22

Cont.

- Alice and Bob can calculate bit stream in advance
- when Alice wishes to send message bitwise exclusive or it with bit stream
- Bob decrypts by exclusive oring ciphertext with bit stream

23

Initialisation

- not secure to use same bit stream twice
- so need IV
- IV must be transmitted to other party
- first block, b_1 , is actually $MD(K_{ab} \parallel IV)$

24

Discussion

- one time pads have the problem that if an attacker may guess plaintext
 - can then bitwise exclusive or guess with ciphertext to get bitstream
 - can then bitwise exclusive or bitstream with any ciphertext using that bitstream attacker wants
- chance of correct guess very low
- one-time pad gives privacy only, not integrity

Mixing in the Plaintext

- similarly to CFB can mix plaintext into bit stream generation
- break message into MD length chunks p_1, p_2, \dots
- call ciphertext blocks c_1, c_2, \dots
- need intermediate values b_1, b_2, \dots

CFB Equivalent Equations

$b_1 = MD(K_{AB} IV)$	$c_1 = p_1 \oplus b_1$
$b_2 = MD(K_{AB} c_1)$	$c_2 = p_2 \oplus b_2$
$b_i = MD(K_{AB} c_{i-1})$	$c_i = p_i \oplus b_i$

Secret Key for Hash

- secret key cryptographic algorithms can be used for message digest
- required properties
 - not require a secret (key)
 - be publishable
 - noninvertible

Unix Password Hash

- uses secret key algorithm
- modified DES algorithm

Method

- text string -> secret key
- pack the 7-bit ascii of the first 8 characters of password into a 56-bit quantity
- insert DES parity
- 12-bit random number, called **salt**, is stored with hashed password

Salt and Modified DES

- salt is used in modifying DES
- modified DES used to avoid use of DES hardware accelerators to reverse password hash
- modified DES is used with password as secret key to encrypt 0
- result stored along with 12-bit number as user's hashed password

Hashing Large Messages

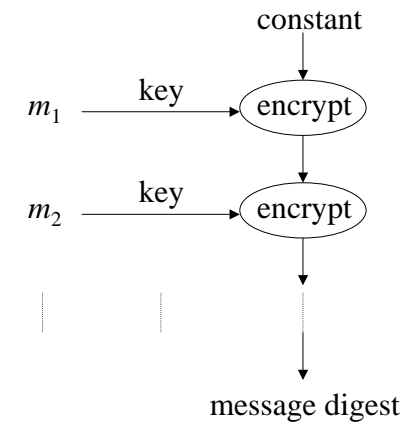
- secret key algorithm has key length, k bits, and message block length, b bits
- divide message into k -bit chunks

Method

- use first message chunk as a key to encrypt a b-bit constant
- result is b bits
- use second k bit chunk of message to encrypt this to new b-bit quantity
- keep going
- final result is message digest

33

Digest using Secret Key Cryptography



34

Discussion

- typical block length is 64 bits
- we decided that 128 bits needed
- need two 64 bit chunks
- generate first as already described
- generate second by using chunks in reverse order
- or do it twice in forward direction, using two different constants

35

Popular Message Digest Algorithms

- MD2
- MD4
- MD5
- SHS

36

MD2

- takes as input arbitrary number of 8-bit bytes
- need to pad if input not correct size
- output is 128 bit message digest

37

MD2

- input is a message whose length is an arbitrary number of bytes
- message is padded to be a multiple of 16 bytes
- 16-byte quantity, called checksum, is appended to end
- checksum is function of padded message

38

Message Processing

- message is processed, 16 bytes at a time
 - each time an intermediate result for the message digest is produced
 - each intermediate value depends on previous intermediate value and 16 bytes being processed
- whole processing can be done in one pass

39

MD2 Padding

- always need padding
- even if message multiple of 16 bytes, add another 16 bytes
- each pad byte specifies number of bytes added

40

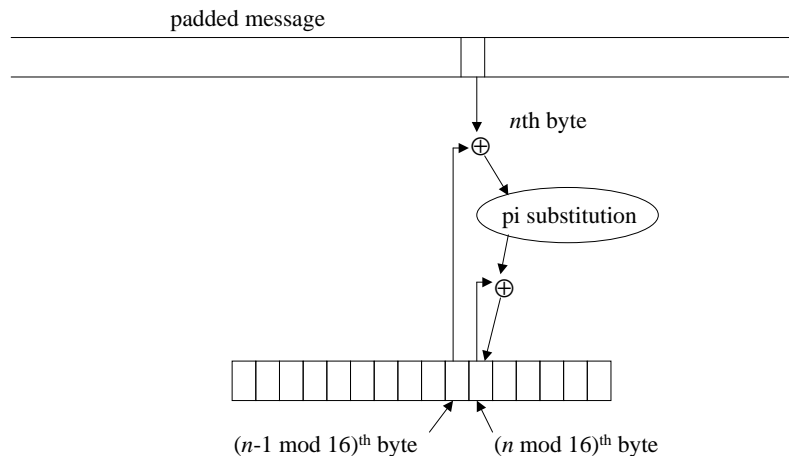
MD2 Checksum Computation

- checksum is 16 bytes
- almost message digest but not cryptographically secure

Checksum Calculation

- checksum initialised to zero
- message is $k \times 16$ bytes
- checksum calculation processes padded message a byte at a time
- total calculation requires $k \times 16$ steps
- each step looks at one byte of message and updates one byte of checksum
- each byte updated k times by end

MD2 Checksum Calculation



MD2 pi Substitution Table

41	46	67	201	162	216	124	1	61	54	84	161	236	240	6	19
98	167	5	243	192	199	115	140	152	147	43	217	188	76	130	202
30	155	87	60	253	212	224	22	103	66	111	24	138	23	229	18
190	78	196	214	218	158	222	73	160	251	245	142	187	47	238	122
169	104	121	145	21	78	7	63	148	194	16	137	11	32	95	33
128	127	93	154	90	144	50	39	53	62	204	231	191	247	151	3
255	25	48	179	72	165	181	209	215	94	146	42	172	86	170	198
79	184	56	210	150	164	125	182	118	252	107	226	156	116	4	241
69	157	112	89	100	113	135	32	134	91	207	101	230	45	168	2
27	96	37	173	174	176	185	246	28	70	97	105	52	64	126	15
85	71	163	35	221	81	175	58	195	92	249	206	186	197	234	38
44	83	13	110	133	40	132	9	211	223	205	244	65	129	77	82
106	220	55	200	108	193	171	250	36	225	123	8	12	189	177	74
120	136	149	139	227	99	232	109	233	203	213	254	59	0	29	57
242	239	183	14	102	88	208	228	119	119	114	248	235	117	75	10
49	68	80	180	143	237	31	26	153	153	141	51	159	17	131	20

Substitution

- claimed to be based on pi
- done this way to avoid charges of sneaky back doors

45

MD2 Final Pass

- message with padding and checksum appended is processed in chunks of 16 bytes
- for each new 16-byte chunk of message taken construct 48-byte quantity
 - 16 byte value of message digest
 - 16 byte chunk of message
 - bitwise exclusive or of first two

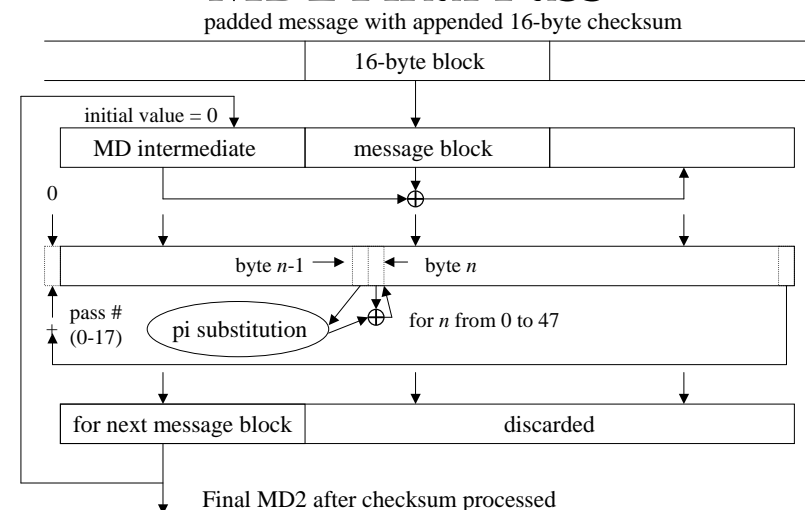
46

Method

- 48-byte quantity massaged byte by byte
- after 18 passes over 48 byte quantity first 16-byte of the final 48-byte result is used as next value of message digest
- during a pass each step processes one byte
- 18x48 steps
- pass number used in computation

47

MD2 Final Pass



48

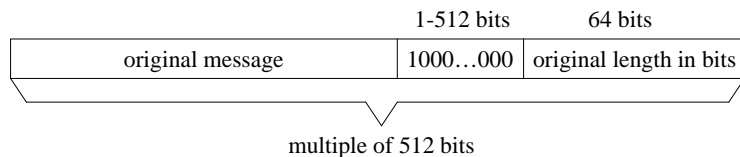
MD4

- designed to be 32-bit word oriented
- so could be computed faster on 32 bit machine
- can handle messages with arbitrary number of bits
- can be done in one pass
- digest is 128 bits

MD4 Message Padding

- actual digest computation requires message multiple of 512 bits (16x32)
- message is padded by adding
 - a 1
 - enough 0's to leave the message 64 bits short
 - a 64 bit quantity representing number of bits in unpadded message, mod 2^{64}

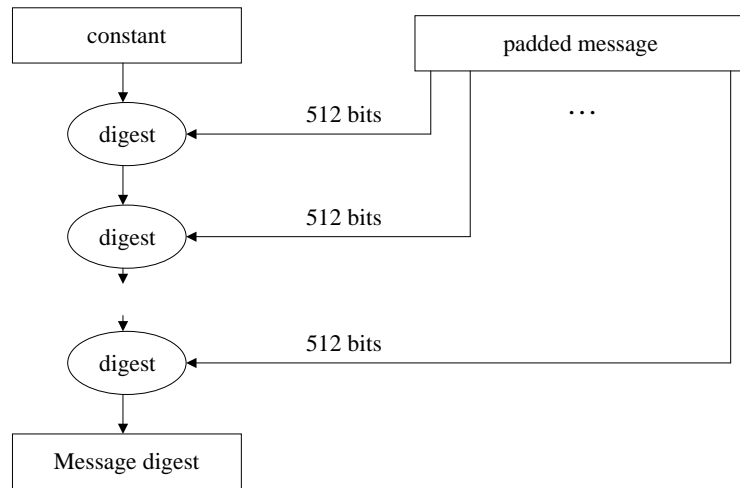
Padding for MD4, MD5, SHS



Overview

- message digest is 128 bits
- message is processed in 512 bit blocks
- digest initialised to fixed value
- each stage (processing a 512 bit block)
 - takes current value of message digest
 - modifies it using next block of message

Overview MD4, MD5, SHA



53

Each 512 Bit Block

- makes three passes over message block
- each pass has a slightly different method of mangling message digest
- at end of stage each word of mangled message is added to pre-stage value to give post-stage value

54

Detail of Stage

- each stage begins with 16-word message block and 4-word digest value
- message words are $m_0, m_1, m_2, \dots, m_{15}$
- digest words are d_0, d_1, d_2, d_3
- each pass modifies d_0, d_1, d_2, d_3 using $m_0, m_1, m_2, \dots, m_{15}$
see text for details

55

Initialisation

- intialisation (base 16 values)
 - d_0 01234567
 - d_1 89abcdef
 - d_2 fedcba98
 - d_3 76543210

56

MD5

- designed to be slower than MD4 but more secure
- makes four passes rather than three over each 16 bit chunk
- slightly different functions
- more varied constants

57

MD5 Message Padding

- same as MD4

58

Overview of Digest Computation

- as in MD4 message is processed in 512 bit blocks
- message digest is 128-bit quantity
- each stage consists of computing a function based on the 512 bit message chunk and the intermediate message digest value to compute new intermediate value

59

Overview Cont

- message digest is result of calculation on final block
- each stage makes four passes over message block

60

Overview Cont.

- as with MD4 at end of stage each word of modified digest is added to pre-stage digest value
- d0, d1, d2 , d3 initialised as in MD4
- see text for details of each pass

61

SHS

- Secure Hash Standard
- takes a message of at most 2^{64} bits
- produces 160 bit output
- similar to MD5
- makes 5 passes over data rather than 4

62

Length

- 2^{64} bits is not a real problem
- would take several hundred years to transmit at 10 Gigabits per second

63

SHS Message Padding

- same as MD4/MD5

64

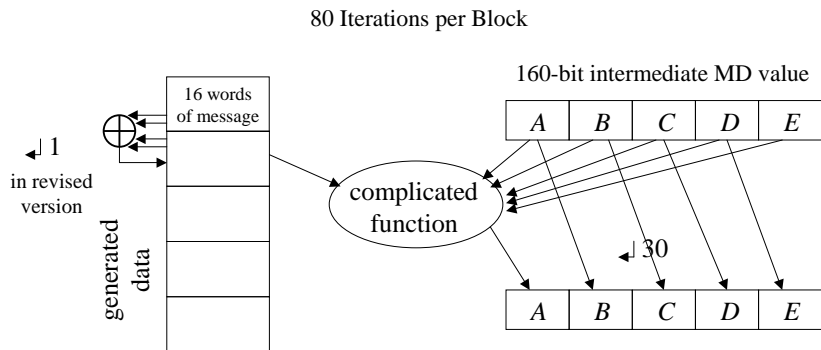
Overview

- operates in stages
- each stage mangles pre-stage message by sequence of operations on current message block
- at end of each stage each word of mangled message digest is added to pre-stage value to produce post-stage value

Overview

- 160-bit digest consists of five 32-bit words
- A,B,C,D,E
- initialised (base 16)
 - A 67452301
 - B efc dab89
 - C 98badcfe
 - D 10325476
 - E c3d2e1f0

Inner Loop of SHS



Stage

- at start of each stage 512-bit message block is used to create 5x512 bit chunk
- first is message block
- rest filled in 32 bit word at a time

512 Bit Block

- nth word (starting from word 16, 0 to 15 being message block) is bitwise exclusive or of words n-3, n-8, n-14 and n-16
- in revised SHS result rotated one bit left

69

Stage

- 5x512 bits gives us eighty 32-bit words
- W0, W1, W2, ..., W79
- used to modify A, B, C, D, E

70

SHS Equations

$B = \text{old } A$ $C = \text{old } B \ll 30$ $D = \text{old } C$ $E = \text{old } D$

$$A = E + (A \ll 5) + W_t + K_t + f(t, B, C, D)$$

$$K_t = \lfloor 2^{30} \sqrt{2} \rfloor = 5a827999_{16}$$

$$K_t = \lfloor 2^{30} \sqrt{3} \rfloor = 6ed9eba1_{16}$$

$$K_t = \lfloor 2^{30} \sqrt{5} \rfloor = 8f1bbcdc_{16}$$

$$K_t = \lfloor 2^{30} \sqrt{10} \rfloor = ca62c1d6_{16}$$

$$f(t, B, C, D) = (B \wedge C) \vee (\sim B \wedge D) \quad (0 \leq t \leq 19)$$

$$f(t, B, C, D) = B \oplus C \oplus D \quad (20 \leq t \leq 39)$$

$$f(t, B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \quad (40 \leq t \leq 59)$$

$$f(t, B, C, D) = B \oplus C \oplus D \quad (60 \leq t \leq 79)$$

71

Summary

- Message Digest Properties
 - Every bit in message affects final result
 - 1 bit changed in message produces arbitrary number changed in result
 - Time consuming to calculate
 - Cannot be reversed

72