Security Engineering

Lesson 11

One-way Hash Functions and Public-Key Algorithms

Spring 2010

Dr. Marenglen Biba

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

Background

- A one-way hash function, $H(M)$, operates on an arbitrary-length pre-image message, M. It returns a fixed-length hash value, h.
  - $h = H(M)$, where h is of length m
- Many functions can take an arbitrary-length input and return an output of fixed length, but one-way hash functions have additional characteristics that make them one-way:
  - Given M, it is easy to compute h.
  - Given h, it is hard to compute M such that $H(M) = h$.
  - Given M, it is hard to find another message, M', such that $H(M) = H(M')$.

Collision-resistance

- In some applications, **one-wayness** is insufficient.
- We need an additional requirement called **collision-resistance**.
- It is hard to find two random messages, *M* and *M'*, such that $H(M) = H(M')$.
- Remember the birthday attack.
  - It is not based on finding another message *M'*, such that $H(M) = H(M')$, but based on finding two random messages, *M* and *M'*, such that $H(M) = H(M')$.

Birthday attack in action

(1) Alice prepares two versions of a contract: one is favorable to Bob; the other bankrupts him.

(2) Alice makes several subtle changes to each document and calculates the hash value for each. (These changes could be things like: replacing SPACE with SPACE-BACKSPACE-SPACE, putting a space or two before a carriage return, and so on. By either making or not making a single change on each of 32 lines, Alice can easily generate 232 different documents.)

(3) Alice compares the hash values for each change in each of the two documents, looking for a pair that matches. She reconstructs the two documents that hash to the same value.

(4) Alice has Bob sign the version of the contract that is favorable to him, using a protocol in which he only signs the hash value.

(5) At some time in the future, Alice substitutes the contract Bob signed with the one that he didn't. Now she can convince an adjudicator that Bob signed the other contract.

Length of One-Way Hash Functions

- Hash functions of 64 bits are just too small to survive a birthday attack.
- Most practical one-way hash functions produce 128-bit hashes.
- This forces anyone attempting the birthday attack to hash 2^{64} random documents to find two that hash to the same value,
  - Not enough for lasting security!
- NIST, in its Secure Hash Standard (SHS), uses a 160-bit hash value. This makes the birthday attack even harder, requiring 2^{80} random hashes.

Generating longer hash values

(1) Generate the hash value of a message, using a one-way hash function.

(2) Prepend the hash value to the message.

(3) Generate the hash value of the concatenation of the message and the hash value.

(4) Create a larger hash value consisting of the hash value generated in step (1) concatenated with the hash value generated in step (3).

(5) Repeat steps (1) through (3) as many times as you wish, *concatenating* as you go.

Overview of One-Way Hash Functions

- It's not easy to design a function that accepts an arbitrary-length input, let alone make it one-way.
- In the real world, one-way hash functions are built on the idea of a compression function.
- This one-way function outputs a hash value of length *n* given an input of some larger length *m*.
- The inputs to the compression function are a message block and the output of the previous blocks of text. The output is the hash of all blocks up to that point.
- That is, the hash of block M_i is:

$$h_i = f(M_i, h_i - 1)$$

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

Snefru

- Snefru is a one-way hash function designed by Ralph Merkle.
- Snefru hashes arbitrary-length messages into either 128-bit or 256-bit values.
- First the message is broken into chunks, each 512-*m* in length. (The variable *m* is the length of the hash value.)
- If the output is a 128-bit hash value, then the chunks are each 384 bits long; if the output is a 256-bit hash value, then the chunks are each 256 bits long.

Snefru working

- The heart of the algorithm is function H, which hashes a 512-bit value into an *m*-bit value.
- The first *m* bits of H's output are the hash of the block; the rest are discarded.
- The next block is appended to the hash of the previous block and hashed again. (The initial block is appended to a string of zeros.)
- Function H is based on E, which is a block-cipher function that operates on 512-bit blocks.
- H is the last *m* bits of the output of E. XORed with the first *m* bits of the input of E.

Security of Snefru

- The security of Snefru resides in function E, which randomizes data in several passes. Each pass is composed of 64 randomizing rounds.
- In each round a different byte of the data is used as an input to an S-box; the output word of the S-box is XORed with two neighboring words of the message.

Cryptanalysis of Snefru

- Using differential cryptanalysis, Biham and Shamir demonstrated the insecurity of two-pass Snefru (128-bit hash value). Their attack finds pairs of messages that hash to the same value within minutes.
- On 128-bit Snefru, their attacks work better than brute force for four passes or less.
- A birthday attack against Snefru takes 2^{64} operations;
- Differential cryptanalysis can find a pair of messages that hash to the same value in $2^{28.5}$ operations for three-pass Snefru and $2^{44.5}$ operations for four-pass Snefru.
- Finding a message that hashes to a given value by brute force requires 2^{128} operations; differential cryptanalysis takes 2^{56} operations for three-pass Snefru and 2^{88} operations for four-pass Snefru.

Number of passes for Snefru

- Although Biham and Shamir didn't analyze 256-bit hash values, they extended their analysis to 224-bit hash values.
- Compared to a birthday attack that requires 2^{112} operations, they can find messages that hash to the same value in $2^{12.5}$ operations for two-pass Snefru, 2^{33} operations for three-pass Snefru, and 2^{81} operations for four-pass Snefru.
- Merkle recommended using Snefru with at least eight passes.
- However, with this many passes the algorithm is significantly slower than either MD5 or SHA.

Outline

One-Way Hash Functions

- Background
- Snefru
- N-Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

N-Hash

- *N*-Hash is an algorithm invented by researchers at Nippon Telephone and Telegraph.
- The hash of each 128-bit block is a function of the block and the hash of the previous block.
  - $H_0 = I$, where I is a random initial value
  - $H_i = g(M_i, H_i\text{- }1) \oplus M_i \oplus H_i\text{- }1$
- The hash of the entire message is the hash of the last message block.
- The random initial value, I, can be any value determined by the user.
- g is a very complicated function.
- The processing stages are N.

Cryptanalysis of N-Hash

- Bert den Boer discovered a way to produce collisions in the round function of *N*-Hash.
- Biham and Shamir used differential cryptanalysis to break 6-round *N*-Hash.
- Their particular attack works for any *N* that is divisible by 3, and is more efficient than the birthday attack for any *N* less than 15.
- The same attack can find pairs of messages that hash to the same value for 12-round *N*-Hash in 2^{56} operations, compared to 2^{64} operations for a brute-force attack.
- *N*-hash with 15 rounds is safe from differential cryptanalysis: The attack requires 2^{72} operations.

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

MD4

- MD4 is a one-way hash function designed by Ron Rivest. MD stands for **Message Digest**; the algorithm produces a 128-bit hash, or message digest, of the input message.
- Rivest outlined his design goals for the algorithm:
  - *Security*. It is computationally infeasible to find two messages that hashed to the same value. No attack is more efficient than brute force.
  - *Direct Security*. MD4's security is not based on any assumption, like the difficulty of factoring.
  - *Speed*. MD4 is suitable for high-speed software implementations. It is based on a simple set of bit manipulations on 32-bit operands.
  - *Simplicity and Compactness*. MD4 is as simple as possible, without large data structures or a complicated program.
  - *Favor Little-Endian Architectures*. MD4 is optimized for microprocessor architectures (specifically Intel microprocessors); larger and faster computers make any necessary translations.

Attacks to MD4

- After the algorithm was first introduced, Bert den Boer and Antoon Bosselaers successfully cryptanalyzed the last two of the algorithm's three rounds.
- In an unrelated cryptanalytic result, Ralph Merkle successfully attacked the first two rounds.
- Eli Biham discussed a differential cryptanalysis attack against the first two rounds of MD4. Even though these attacks could not be extended to the full algorithm, Rivest strengthened the algorithm.
- The result is MD5.

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

MD5

- MD5 is an improved version of MD4. Although more complex than MD4, it is similar in design and also produces a 128-bit hash.
- After some initial processing, MD5 processes the input text in 512-bit blocks, divided into 16 32-bit sub-blocks.
- The output of the algorithm is a set of four 32-bit blocks, which concatenate to form a single 128-bit hash value.
- First, the message is padded so that its length is just 64 bits short of being a multiple of 512. This padding is a single 1-bit added to the end of the message, followed by as many zeros as are required.
- Then, a 64-bit representation of the message's length (before padding bits were added) is appended to the result. These two steps serve to make the message length an exact multiple of 512 bits in length.

Chaining variables

- Four 32-bit variables are initialized:
  - A = 0x01234567
  - B = 0x89abcdef
  - C = 0xfedcba98
  - D = 0x76543210
- These are called **chaining variables**.
- Now, the main loop of the algorithm begins. This loop continues for as many 512-bit blocks as are in the message.
- The four variables are copied into different variables: *a* gets *A*, *b* gets *B*, *c* gets *C*, and *d* gets *D*.

MD5 Rounds

- The main loop has four rounds (MD4 had only three rounds), all very similar.
- Each round uses a different operation 16 times. Each operation performs a **nonlinear function** on three of *a*, *b*, *c*, and *d*.
- Then it adds that result to the fourth variable, a sub-block of the text and a constant.
- Then it rotates that result to the right. a variable number of bits and adds the result to one of *a*, *b*, *c*, or *d*.
- Finally the result replaces one of *a*, *b*, *c*, or *d*.

MD5 Main Loop

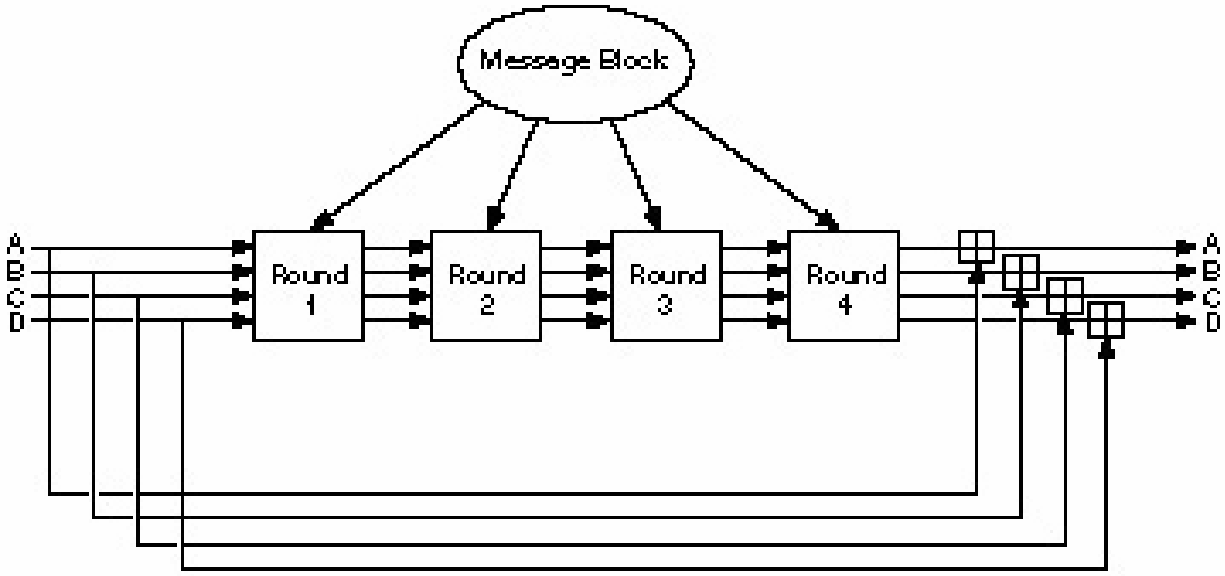


Figure 18.5 MD5 main loop.

MD5 Nonlinear Functions

- There are four nonlinear functions, one used in each operation (a different one for each round).
  - $F(X,Y,Z) = (X \perp Y) \vdash ((\neg X) \perp Z)$
  - $G(X,Y,Z) = (X \perp Z) \neg (Y (\neg Z))$
  - $H(X,Y,Z) = X \oplus Y \oplus Z$
  - $I(X,Y,Z) = Y \oplus (X \vdash (\neg Z))$
- ($\oplus$ is XOR, $\perp$ is AND, $\vdash$ is OR, and $\neg$ is NOT.)

- These functions are designed so that if the corresponding bits of *X*, *Y*, and *Z* are independent and unbiased, then each bit of the result will also be independent and unbiased.

One MD5 Operation

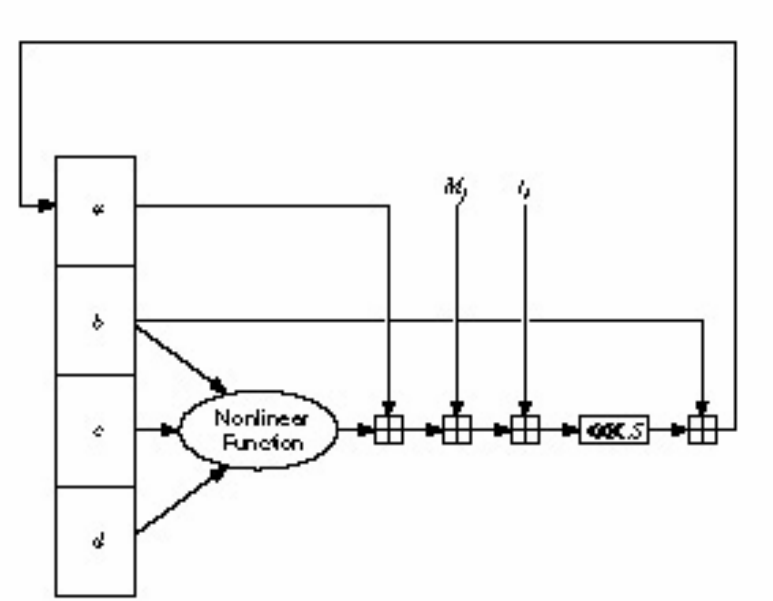


Figure 18.6 One MD5 operation.

- If *M*j represents the *j* th sub-block of the message (from 0 to 15), and <<<s represents a left circular shift of *s* bits, the four operations are:
  - FF(*a*,*b*,*c*,*d*,*M*j,*s*,*t*i) denotes $a = b + ((a + F(b,c,d) + M_j + t_i) <<< s)$
  - GG(*a*,*b*,*c*,*d*,*M*j,*s*,*t*i) denotes $a = b + ((a + G(b,c,d) + M_j + t_i) <<< s)$
  - HH(*a*,*b*,*c*,*d*,*M*j,*s*,*t*i) denotes $a = b + ((a + H(b,c,d) + M_j + t_i) <<< s)$
  - II(*a*,*b*,*c*,*d*,*M*j,*s*,*t*i) denotes $a = b + ((a + I(b,c,d) + M_j + t_i) <<< s)$

Rounds of MD5

Round 1:

- FF (*a*, *b*, *c*, *d*, *M*0, 7, 0xd76aa478)
- FF (*d*, *a*, *b*, *c*, *M*1, 12, 0xe8c7b756)
- FF (*c*, *d*, *a*, *b*, *M*2, 17, 0x242070db)
- FF (*b*, *c*, *d*, *a*, *M*3, 22, 0xc1bdceee)
- FF (*a*, *b*, *c*, *d*, *M*4, 7, 0xf57c0faf)
- FF (*d*, *a*, *b*, *c*, *M*5, 12, 0x4787c62a)
- FF (*c*, *d*, *a*, *b*, *M*6, 17, 0xa8304613)
- FF (*b*, *c*, *d*, *a*, *M*7, 22, 0xfd469501)
- FF (*a*, *b*, *c*, *d*, *M*8, 7, 0x698098d8)
- FF (*d*, *a*, *b*, *c*, *M*9, 12, 0x8b44f7af)
- FF (*c*, *d*, *a*, *b*, *M*10, 17, 0xffff5bb1)
- FF (*b*, *c*, *d*, *a*, *M*11, 22, 0x895cd7be)
- FF (*a*, *b*, *c*, *d*, *M*12, 7, 0x6b901122)
- FF (*d*, *a*, *b*, *c*, *M*13, 12, 0xfd987193)
- FF (*c*, *d*, *a*, *b*, *M*14, 17, 0xa679438e)
- FF (*b*, *c*, *d*, *a*, *M*15, 22, 0x49b40821)

Round 2:

- GG (*a*, *b*, *c*, *d*, *M*1, 5, 0xf61e2562)
- GG (*d*, *a*, *b*, *c*, *M*6, 9, 0xc040b340)
- GG (*c*, *d*, *a*, *b*, *M*11, 14, 0x265e5a51)
- GG (*b*, *c*, *d*, *a*, *M*0, 20, 0xe9b6c7aa)
- GG (*a*, *b*, *c*, *d*, *M*5, 5, 0xd62f105d)
- GG (*d*, *a*, *b*, *c*, *M*10, 9, 0x02441453)
- GG (*c*, *d*, *a*, *b*, *M*15, 14, 0xd8a1e681)
- GG (*b*, *c*, *d*, *a*, *M*4, 20, 0xe7d3fbc8)
- GG (*a*, *b*, *c*, *d*, *M*9, 5, 0x21e1cde6)
- GG (*d*, *a*, *b*, *c*, *M*14, 9, 0xc33707d6)
- GG (*c*, *d*, *a*, *b*, *M*3, 14, 0xf4d50d87)
- GG (*b*, *c*, *d*, *a*, *M*8, 20, 0x455a14ed)
- GG (*a*, *b*, *c*, *d*, *M*13, 5, 0xa9e3e905)
- GG (*d*, *a*, *b*, *c*, *M*2, 9, 0xfcefa3f8)
- GG (*c*, *d*, *a*, *b*, *M*7, 14, 0x676f02d9)
- GG (*b*, *c*, *d*, *a*, *M*12, 20, 0x8d2a4c8a)

Rounds of MD5

Round 3:

HH (*a*, *b*, *c*, *d*, *M*5, 4, 0xfffa3942)
HH (*d*, *a*, *b*, *c*, *M*8, 11, 0x8771f681)
HH (*c*, *d*, *a*, *b*, *M*11, 16, 0x6d9d6122)
HH (*b*, *c*, *d*, *a*, *M*14, 23, 0xfde5380c)
HH (*a*, *b*, *c*, *d*, *M*1, 4, 0xa4beea44)
HH (*d*, *a*, *b*, *c*, *M*4, 11, 0x4bdecfa9)
HH (*c*, *d*, *a*, *b*, *M*7, 16, 0xf6bb4b60)
HH (*b*, *c*, *d*, *a*, *M*10, 23, 0xbebfbc70)
HH (*a*, *b*, *c*, *d*, *M*13, 4, 0x289b7ec6)
HH (*d*, *a*, *b*, *c*, *M*0, 11, 0xeaa127fa)
HH (*c*, *d*, *a*, *b*, *M*3, 16, 0xd4ef3085)
HH (*b*, *c*, *d*, *a*, *M*6, 23, 0x04881d05)
HH (*a*, *b*, *c*, *d*, *M*9, 4, 0xd9d4d039)
HH (*d*, *a*, *b*, *c*, *M*12, 11, 0xe6db99e5)
HH (*c*, *d*, *a*, *b*, *M*15, 16, 0x1fa27cf8)
HH (*b*, *c*, *d*, *a*, *M*2, 23, 0xc4ac5665)

Round 4:

II (*a*, *b*, *c*, *d*, *M*0, 6, 0xf4292244)
II (*d*, *a*, *b*, *c*, *M*7, 10, 0x432aff97)
II (*c*, *d*, *a*, *b*, *M*14, 15, 0xab9423a7)
II (*b*, *c*, *d*, *a*, *M*5, 21, 0xfc93a039)
II (*a*, *b*, *c*, *d*, *M*12, 6, 0x655b59c3)
II (*d*, *a*, *b*, *c*, *M*3, 10, 0x8f0ccc92)
II (*c*, *d*, *a*, *b*, *M*10, 15, 0xffeff47d)
II (*b*, *c*, *d*, *a*, *M*1, 21, 0x85845dd1)
II (*a*, *b*, *c*, *d*, *M*8, 6, 0x6fa87e4f)
II (*d*, *a*, *b*, *c*, *M*15, 10, 0xfe2ce6e0)
II (*c*, *d*, *a*, *b*, *M*6, 15, 0xa3014314)
II (*b*, *c*, *d*, *a*, *M*13, 21, 0x4e0811a1)
II (*a*, *b*, *c*, *d*, *M*4, 6, 0xf7537e82)
II (*d*, *a*, *b*, *c*, *M*11, 10, 0xbd3af235)
II (*c*, *d*, *a*, *b*, *M*2, 15, 0x2ad7d2bb)
II (*b*, *c*, *d*, *a*, *M*9, 21, 0xeb86d391)

Security of MD5

- In 1996, a flaw was found with the design of MD5. While it was not a clearly fatal weakness, cryptographers began recommending the use of other algorithms, such as SHA-1 (which has since been found also to be vulnerable).
- In 2004, more serious flaws were discovered, making further use of the algorithm for security purposes questionable.
- In 2007 a group of researchers described how to create a pair of files that share the same MD5 checksum.
- In an attack on MD5 published in December 2008, a group of researchers used this technique to fake SSL certificate validity.
- US-CERT of the U. S. Department of Homeland Security said MD5 "should be considered cryptographically broken and unsuitable for further use," and most U.S. government applications will be required to move to the SHA-2 family of hash functions after 2010

MD6

- MD6 was submitted to the NIST SHA-3 competition.
- However, on July 1, 2009, Rivest posted a comment at NIST that MD6 is not yet ready to be a candidate for SHA-3 because of speed issues and an inability to supply a proof of security for a faster reduced-round version, though Rivest also stated at MD6 web site that it is not withdrawn formally.
- MD6 did not advance to the second round of the SHA-3 competition.
- In December 2008, a researcher at Fortify Software discovered a buffer overflow in the original MD6 hash algorithm's reference implementation.
- This error was later made public by professor Ron Rivest on 19 February 2009, with a release of a corrected reference implementation in advance of the Fortify Report.

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

MD2

- MD2 is another 128-bit one-way hash function designed by Ron Rivest.
- It, along with MD5, is used in the PEM protocols.
- The security of MD2 is dependent on a random permutation of bytes.

MD2 Security

- In 2004, MD2 was shown to be vulnerable to an attack with time complexity equivalent to 2^{104} applications of the compression function (Muller, 2004).
- The author concludes, "*MD2 can no longer be considered a secure one-way hash function*".
- In 2009, security updates were issued disabling MD2 in OpenSSL, GnuTLS, and Network Security Services.

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

Secure Hash Algorithm

- NIST, along with the NSA, designed the Secure Hash Algorithm (SHA) for use with the Digital Signature Standard.
- SHA produces a 160-bit hash, longer than MD5.

Description of SHA

- First, the message is padded to make it a multiple of 512 bits long. Padding is exactly the same as in MD5: First append a one, then as many zeros as necessary to make it 64 bits short of a multiple of 512, and finally a 64-bit representation of the length of the message before padding.
- Five 32-bit variables (MD5 has four variables, but this algorithm needs to produce a 160-bit hash) are initialized as follows:
  - A = 0x67452301
  - B = 0xefcdab89
  - C = 0x98badcfe
  - D = 0x10325476
  - E = 0xc3d2e1f0

Processing in SHA

- First the five variables are copied into different variables: *a* gets *A*, *b* gets *B*, *c* gets *C*, *d* gets *D*, and *e* gets *E*.
- The main loop has four rounds of 20 operations each (MD5 has four rounds of 16 operations each).
- Each operation performs a nonlinear function on three of *a*, *b*, *c*, *d*, and *e*, and then does shifting and adding similar to MD5.
- SHA's set of nonlinear functions is:
  - $f_t(X,Y,Z) = (X \perp Y) \vdash ((\neg X) \perp Z)$, for $t = 0$ to 19.
  - $f_t(X,Y,Z) = X \oplus Y \oplus Z$, for $t = 20$ to 39.
  - $f_t(X,Y,Z) = (X \perp Y) \vdash (X \perp Z) \vdash (Y \perp Z)$, for $t = 40$ to 59.
  - $f_t(X,Y,Z) = X \oplus Y \oplus Z$, for $t = 60$ to 79.

(*t* is the operation number)

Main Loop of SHA

- If t is the operation number (from 0 to 79), W_t represents the t^{th} sub-block of the expanded message, and <<< s represents a left circular shift of s bits, then the main loop looks like:

FOR t = 0 to 79

$TEMP = (a <<< 5) + f_t(b,c,d) + e + W_t + K_t$

$e = d$

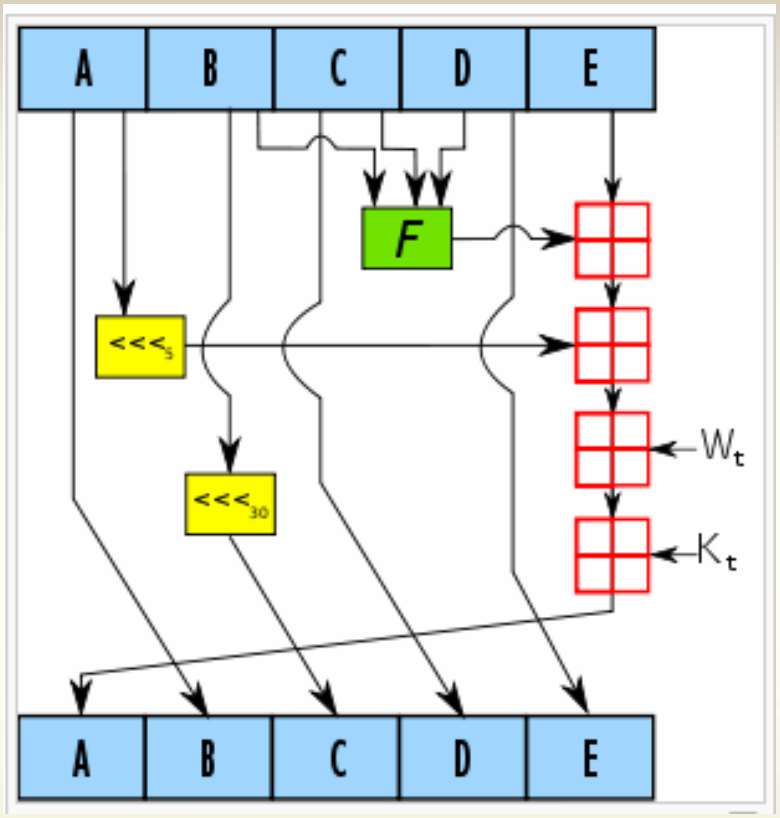
$d = c$

$c = b <<< 30$

$b = a$

$a = TEMP$

SHA-1



- One iteration within the SHA-1 compression function: A, B, C, D and E are 32-bit words of the state; *F* is a nonlinear function that varies; $\lll n$ denotes a left bit rotation by *n* places; *n* varies for each operation;
  W_t is the expanded message word of round t;
  K_t is the round constant of round t; $\boxplus$ denotes addition modulo 2^{32}.

Comparison of SHA

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations	Collisions found
SHA-0		160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rot	Yes
SHA-1									None (2^{63} attack)[5]
SHA-2	*SHA-256/224*	256/224	256	512	$2^{64} - 1$	32	64	+,and,or,xor,shr,rot	None
	SHA-512/384	512/384	512	1024	$2^{128} - 1$	64	80	+,and,or,xor,shr,rot	None

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

Using Symmetric Block Algorithms

- It is possible to use a symmetric block cipher algorithm as a one-way hash function.
- The idea is that if the block algorithm is secure, then the one-way hash function will also be secure.
- The most obvious method is to encrypt the message with the algorithm in CBC or CFB mode, a fixed key, and IV; the last ciphertext block is the hash value.

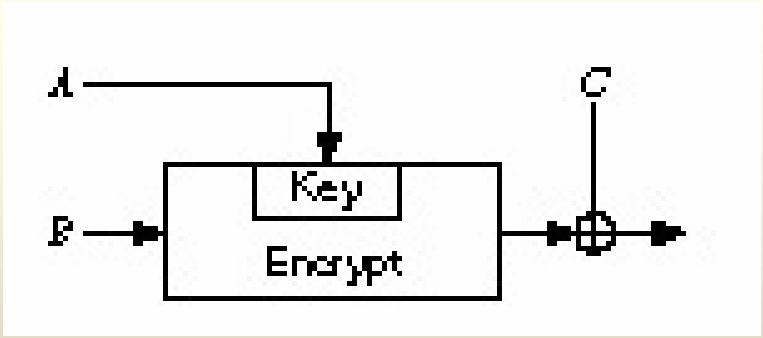
Schemes Where the Hash Length Equals the Block Size

- The general scheme is as follows:

  $H_0 = I_H$, where I_H is a random initial value

  $H_i = E_A(B) \oplus C$

where A, B, and C can be either M_i, H_{i-1}, $(M_i \oplus H_{i-1})$, or a constant (assumed to be 0).



Secure Hash Functions

Table 18.1
Secure Hash Functions Where the Block Length Equals the Hash Size

$$H_i = E_{H_{i-1}}(M_i) \oplus M_i$$

$$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$$

$$H_i = E_{H_{i-1}}(M_i) \oplus H_{i-1} \oplus M_i$$

$$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i$$

$$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$$

$$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$$

$$H_i = E_{M_i}(H_{i-1}) \oplus M_i \oplus H_{i-1}$$

$$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus H_{i-1}$$

$$H_i = E_{M_i \oplus H_{i-1}}(M_i) \oplus M_i$$

$$H_i = E_{M_i \oplus H_{i-1}(H_{i-1})} \oplus H_{i-1}$$

$$H_i = E_{M_i \oplus H_{i-1}}(Mi) \oplus H_{i-1}$$

$$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus M_i$$

- The three different variables can take on one of four possible values, so there are 64 total schemes of this type. Bart Preneel studied them all.
- Fifteen are trivially weak because the result does not depend on one of the inputs. Thirty-seven are insecure for more subtle reasons.
- The Table lists the 12 secure schemes remaining: The first 4 are secure against all attacks and the last 8 are secure against all but a fixed-point attack,

Secure Hash Functions

- The following schemes, proposed in the literature, have been shown to be secure:

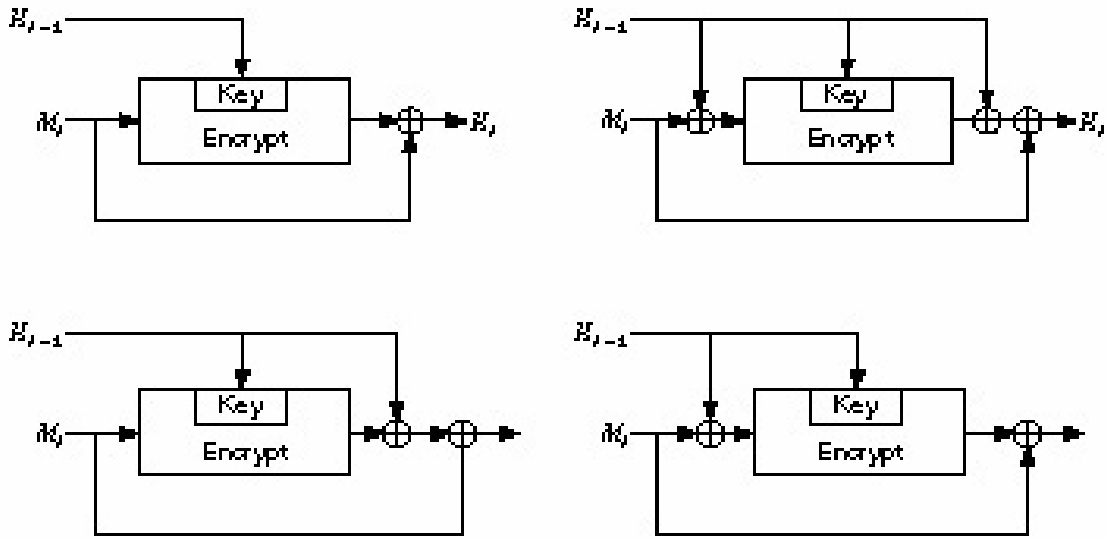


Figure 18.9 *The four secure hash functions where the block length equals the hash size.*

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

Using Public-Key Algorithms

- It is possible to use a public-key encryption algorithm in a block chaining mode as a one-way hash function.

- If you then throw away the private key, breaking the hash would be as difficult as reading the message without the private key.

Using RSA

- Here's an example using RSA.
- If M is the message to be hashed, n is the product of two primes p and q, and e is another large number relatively prime to $(p - 1)(q - 1)$, then the hash function, $H(M)$, would be
  - $H(M) = M^e \bmod n$
- An even easier solution would be to use a single strong prime as the modulus p. Then:
  - $H(M) = M^e \bmod p$
- Breaking this problem is probably as difficult as finding the discrete logarithm of e.
- The problem with this algorithm is that it's far slower than many any others.

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

Choosing among one-way Hash Functions

- The contenders seem to be SHA, MD5, and constructions based on block ciphers.
- Schneier votes for SHA.
  - it has a longer hash value than MD5
  - is faster than the various block-cipher constructions, and
  - was developed by the NSA.
    - We trust the NSA's abilities at cryptanalysis, even if they don't make their results public. ☺

Speed of Hash Functions

Table 18.2
Speeds of Some Hash Functions on a 33 MHz 486SX

Algorithm	Hash Length	Encryption Speed (kilobytes/second)
Abreast Davies-Meyer (with IDEA)	128	22
Davies-Meyer (with DES)	64	9
GOST Hash	256	11
HAVAL (3 passes)	variable	168
HAVAL (4 passes)	variable	118
HAVAL (5 passes)	variable	95
MD2	128	23
MD4	128	236
MD5	128	174
N-HASH (12 rounds)	128	29
N-HASH (15 rounds)	128	24
RIPE-MD	128	182
SHA	160	75
SNEFRU (4 passes)	128	48
SNEFRU (8 passes)	128	23

Outline

One-Way Hash Functions

- Background
- Snefru
- N- Hash
- MD4
- MD5
- MD2
- Secure Hash Algorithm (SHA)
- One-Way Hash Functions Using Symmetric Block Algorithms
- Using Public-Key Algorithms
- Choosing a One-Way Hash Function
- Message Authentication Codes

MACs

- A message authentication code, or MAC, is a key-dependent one-way hash function.
- MACs have the same properties as the one-way hash functions discussed previously, but they also include a key.
- Only someone with the identical key can verify the hash.

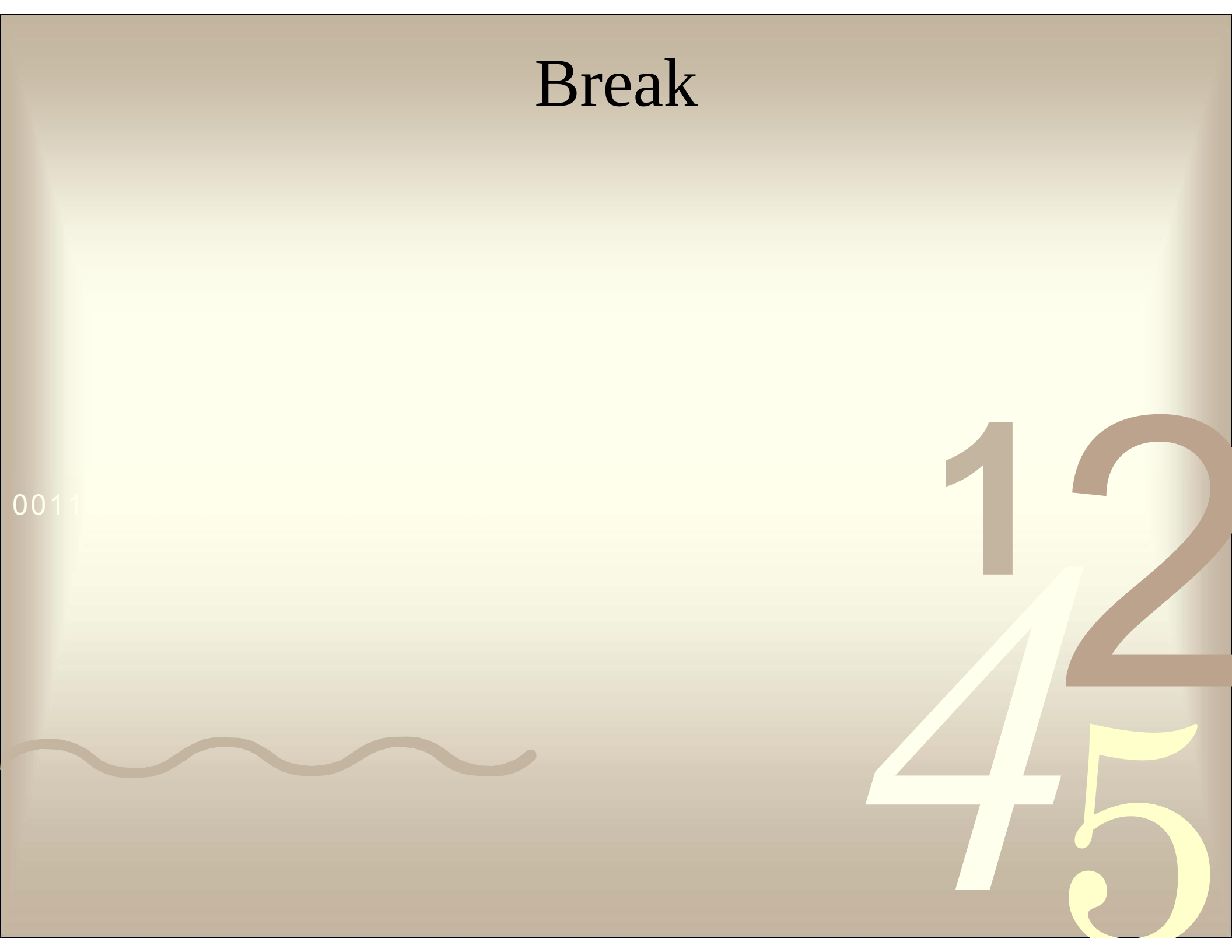
Use of MACs

- MACs can be used to authenticate files between users.
- They can also be used by a single user to determine if his files have been altered, perhaps by a virus.
- A user could compute the MAC of his files and store that value in a table.
- If the user used instead a one-way hash function, then the virus could compute the new hash value after infection and replace the table entry.
- A virus could not do that with a MAC, because the virus does not know the key.

MAC with Symmetric Algorithms

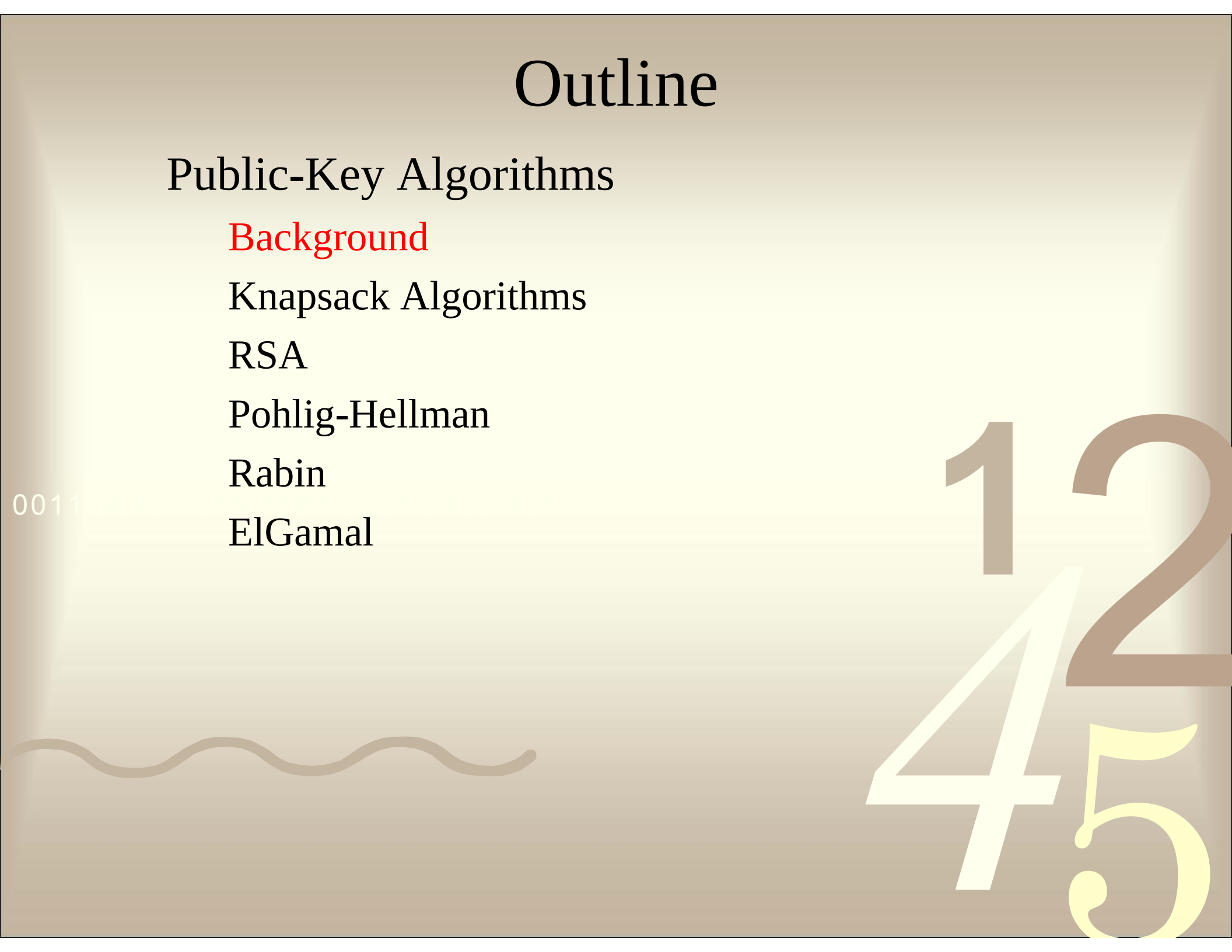
- An easy way to turn a one-way hash function into a MAC is to encrypt the hash value with a symmetric algorithm.

Break



Outline

- Public-Key Algorithms
  - Background
  - Knapsack Algorithms
  - RSA
  - Pohlig-Hellman
  - Rabin
  - ElGamal



The birth of Public-key Cryptography

- The concept of public-key cryptography was invented by Whitfield Diffie and Martin Hellman, and independently by Ralph Merkle.
- Their contribution to cryptography was the notion that keys could come in pairs — an encryption key and a decryption key — and that it could be infeasible to generate one key from the other.
- Diffie and Hellman first presented this concept at the 1976 National Computer Conference, when their seminal paper “New Directions in Cryptography” was published.

Public-key algorithms

- Since 1976, numerous public-key cryptography algorithms have been proposed. Many of these are insecure.
- Of those still considered secure, many are impractical. Either they have too large a key or the ciphertext is much larger than the plaintext.
- Only a few algorithms are both secure and practical. These algorithms are generally based on one of the hard problems we discussed in complexity theory.
- Of these secure and practical public-key algorithms, some are only suitable for key distribution.
- Others are suitable for encryption (and by extension for key distribution). Still others are only useful for digital signatures.
- Only three algorithms work well for both encryption and digital signatures: RSA, ElGamal, and Rabin.

Public-key algorithms are slow

- The RSA, ElGamal, and Rabin algorithms are slow.
- They encrypt and decrypt data much more slowly than symmetric algorithms; usually that's too slow to support bulk data encryption.
- Hybrid cryptosystems speed things up: A symmetric algorithm with a random session key is used to encrypt the message, and a public-key algorithm is used to encrypt the random session key.

Security of Public-key Algorithms

- Since a cryptanalyst has access to the public key, he can always choose any message to encrypt.
- This means that a cryptanalyst, given $C = E_K(P)$, can guess the value of P and easily check his guess.
- This is a serious problem if the number of possible plaintext messages is small enough to allow exhaustive search, but can be solved by padding messages with a string of random bits.
- This makes identical plaintext messages encrypt to different ciphertext messages.

Outline

- Public-Key Algorithms
  - Background
  - Knapsack Algorithms
  - RSA
  - Pohlig-Hellman
  - Rabin
  - ElGamal

The Knapsack problem

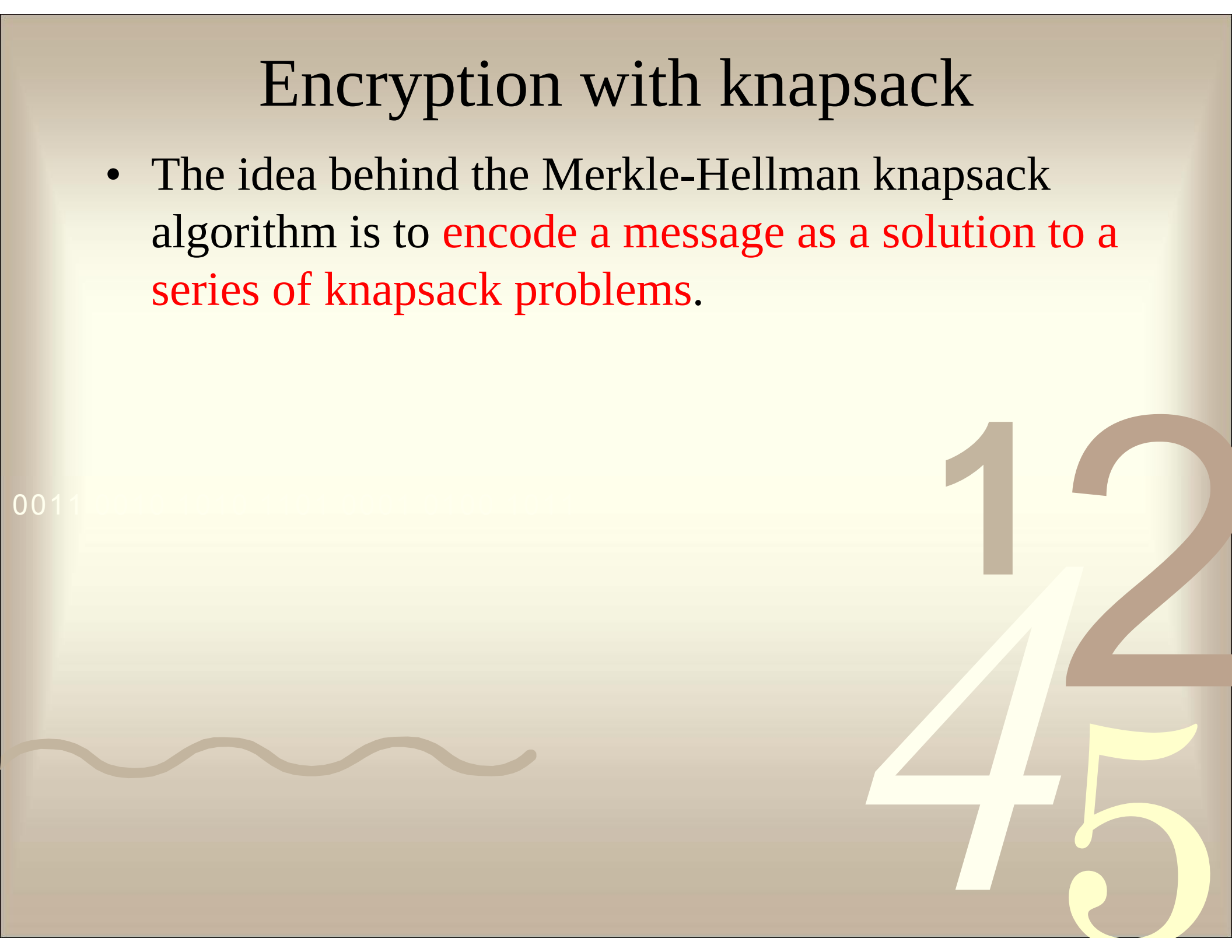
- The first algorithm for generalized public-key encryption was the knapsack algorithm developed by Ralph Merkle and Martin Hellman.
- It could only be used for encryption, although Adi Shamir later adapted the system for digital signatures.
- Knapsack algorithms get their security from the knapsack problem, an **NP-complete** problem.
- Although this algorithm was later found to be **insecure**, it is worth examining because it demonstrates how an **NP-complete** problem can be used for public-key cryptography.

The Knapsack problem

- The knapsack problem is a simple one. Given a pile of items, each with different weights, is it possible to put some of those items into a knapsack so that the knapsack weighs a given amount?
- More formally: Given a set of values M_1, M_2,..., M_n , and a sum S, compute the values of b_i such that
  - $S = b_1M_1 + b_2M_2 + ...+ b_nM_n$
- The values of b_i can be either zero or one. A one indicates that the item is in the knapsack; a zero indicates that it isn't.
- For example, the items might have weights of 1, 5, 6, 11, 14, and 20. You could pack a knapsack that weighs 22; use weights 5, 6, and 11. You could not pack a knapsack that weighs 24. In general, the time required to solve this problem seems to grow exponentially with the number of items in the pile.

Encryption with knapsack

- The idea behind the Merkle-Hellman knapsack algorithm is to encode a message as a solution to a series of knapsack problems.



Superincreasing Knapsacks

- What is the easy knapsack problem?
- If the list of weights is a **superincreasing sequence**, then the resulting knapsack problem is easy to solve.
- A superincreasing sequence is a sequence in which every term is greater than the sum of all the previous terms.
- For example, {1, 3, 6, 13, 27, 52} is a superincreasing sequence, but {1, 3, 4, 9, 15, 25} is not.

Creating the Public Key from the Private Key

- This is how the algorithm works: To get a normal knapsack sequence, take a superincreasing knapsack sequence, for example {2, 3, 6, 13, 27, 52}, and multiply all of the values by a number *n* mod *m*.
- The modulus should be a number greater than the sum of all the numbers in the sequence: for example, 105. The multiplier should have no factors in common with the modulus: for example, 31. The normal knapsack sequence would then be
  - 2 * 31 mod 105 = 62
  - 3 * 31 mod 105 = 93
  - 6 * 31 mod 105 = 81
  - 13 * 31 mod 105 = 88
  - 27 * 31 mod 105 = 102
  - 52 * 31 mod 105 = 37
- The knapsack would then be {62, 93, 81, 88, 102, 37}.
- The superincreasing knapsack sequence is the private key. The normal knapsack sequence is the public key.

Encryption

- To encrypt a binary message, first break it up into blocks equal to the number of items in the knapsack sequence.
- Then, allowing a one to indicate the item is present and a zero to indicate that the item is absent, compute the total weights of the knapsacks — one for every message block.
- For example, if the message were 0110001101011101110 in binary, encryption using the previous knapsack would proceed like this:
  - message = 011000 110101 101110
  - 011000 corresponds to 93 + 81 = 174
  - 110101 corresponds to 62 + 93 + 88 + 37 = 280
  - 101110 corresponds to 62 + 81 + 88 + 102 = 333
- The ciphertext would be
  - 174,280,333

Security of Knapsack

- It wasn’t a million machines that broke the knapsack cryptosystem, but a pair of cryptographers.
- First a single bit of plaintext was recovered.
- Then, Shamir showed that knapsacks can be broken in certain circumstances.
- Finally, Shamir and Zippel found flaws in the transformation that allowed them to reconstruct the superincreasing knapsack from the normal knapsack.
- At the conference where the results were presented, the attack was demonstrated on stage using an Apple II computer.

Outline

- Public-Key Algorithms
  - Background
  - Knapsack Algorithms
  - RSA
  - Pohlig-Hellman
  - Rabin
  - ElGamal

RSA

- Soon after Merkle's knapsack algorithm came the first full-fledged public-key algorithm, one that works for encryption and digital signatures: RSA.
- Of all the public-key algorithms proposed over the years, RSA is by far the easiest to understand and implement.
- It is also the most popular. Named after the three inventors—Ron Rivest, Adi Shamir, and Leonard Adleman — it has since withstood years of extensive cryptanalysis.
- Although the cryptanalysis neither proved nor disproved RSA's security, it does suggest a confidence level in the algorithm.

RSA Description

- RSA gets its security from the difficulty of factoring large numbers.
- The public and private keys are functions of a pair of large (100 to 200 digits or even larger) prime numbers.
- Recovering the plaintext from the public key and the ciphertext is conjectured to be equivalent to factoring the product of the two primes.

RSA Description

- To generate the two keys, choose two random large prime numbers, p and q. For maximum security, choose p and q of equal length. Compute the product:
  - $n = pq$
- Then randomly choose the encryption key, e, such that e and $(p - 1)(q - 1)$ are relatively prime.
- Finally, use the extended Euclidean algorithm to compute the decryption key, d, such that
  - $ed \equiv 1 \bmod (p - 1)(q - 1)$
- In other words,
  - $d = e^{-1} \bmod ((p - 1)(q - 1))$
- Note that d and n are also relatively prime.
- The numbers e and n are the public key; the number d is the private key.
- The two primes, p and q, are no longer needed. They should be discarded, but never revealed.

Encrypting with RSA

- To encrypt a message *m*, first divide it into numerical blocks smaller than *n* (with binary data, choose the largest power of 2 less than *n*).
- That is, if both *p* and *q* are 100-digit primes, then *n* will have just under 200 digits and each message block, m_i , should be just under 200 digits long.
- The encrypted message, *c*, will be made up of similarly sized message blocks, c_i, of about the same length. The encryption formula is simply:

$$c_i = m_i^e \bmod n$$

Decryption with RSA

- To decrypt a message, take each encrypted block c_i and compute:
  - $m_i = c_i^d \bmod n$

RSA Summary

Table 19.2
RSA Encryption

	Public Key:
n	product of two primes, p and q (p and q must remain secret)
e	relatively prime to $(p - 1)(q - 1)$
	Private Key:
d	$e^{-1} \bmod ((p - 1)(q - 1))$
	Encrypting:
$c = m^e \bmod n$	
	Decrypting:
$m = c^d \bmod n$	

RSA Example

- If $p = 47$ and $q = 71$, then $n = pq = 3337$
- The encryption key, e, must have no factors in common with
  - $(p - 1)(q - 1) = 46 * 70 = 3220$
- Choose e (at random) to be 79. In that case
  - $d = 79^{-1} \bmod 3220 = 1019$
- This number is calculated using the extended Euclidean algorithm.
- Publish e and n, and keep d secret. Discard p and q.
- To encrypt the message
  - $m = 6882326879666683$

RSA Example

- First break it into small blocks. Three-digit blocks work nicely in this case. The message is split into six blocks, m_i, in which
  - $m1 = 688$
  - $m2 = 232$
  - $m3 = 687$
  - $m4 = 966$
  - $m5 = 668$
  - $m6 = 003$
- The first block is encrypted as
  - $688^{79} \bmod 3337 = 1570 = c_1$
- Performing the same operation on the subsequent blocks generates an encrypted message:
  - $c = 1570\ 2756\ 2091\ 2276\ 2423\ 158$
- Decrypting the message requires performing the same exponentiation using the decryption key of 1019, so
  - $1570^{1019} \bmod 3337 = 688 = m_1$
- The rest of the message can be recovered in this manner.

Speed of RSA

- RSA is much slower than DES and other symmetric cryptosystems.
- In hardware, RSA is estimated to be about 1000 times slower than DES.
- In software, DES is about 100 times faster than RSA.
- These numbers may change slightly as technology changes, but RSA will never approach the speed of symmetric algorithms.

Security of RSA

- The security of RSA depends wholly on the problem of factoring large numbers.
- Factoring *n* is the most obvious means of attack. Any adversary will have the public key, *e*, and the modulus, *n*. To find the decryption key, *d*, he has to factor *n*.
- It is certainly possible for a cryptanalyst to try every possible *d* until he stumbles on the correct one.
  - This brute-force attack is even less efficient than trying to factor *n*.

Chosen Ciphertext Attack against RSA

- Some attacks work against the implementation of RSA.
- These are not attacks against the basic algorithm, but against the protocol. It's important to realize that it's not enough to use RSA.

Example:

- Eve, listening in on Alice's communications, manages to collect a ciphertext message, *c*, encrypted with RSA in her public key.
- Eve wants to be able to read the message. Mathematically, she wants *m*, in which:
  - $m = c^d$

Chosen Ciphertext Attack against RSA

- To recover m, she first chooses a random number, r, such that r is less than n.
- She gets Alice's public key, e. Then she computes:
  - $x = r^e \bmod n$
  - $y = x^c \bmod n$
  - $t = r^{-1} \bmod n$
- If $x = r^e \bmod n$, then $r = x^d \bmod n$.

Chosen Ciphertext Attack against RSA

- Now, Eve gets Alice to sign y with her private key, thereby decrypting y. (Alice has to sign the message, not the hash of the message.) Remember, Alice has never seen y before. Alice sends Eve
  - $u = y^d \bmod n$
- Now, Eve computes
  - $tu \bmod n = r^{-1}y^d \bmod n = r^{-1}x^d c^d \bmod n = c^d \bmod n = m$
- Eve now has m.

- Moral: Never use RSA to sign a random document presented to you by a stranger. Always use a one-way hash function first. The ISO 9796 block format prevents this attack.

Common Modulus Attack on RSA

- A possible RSA implementation gives everyone the same *n*, but different values for the exponents *e* and *d*. Unfortunately, this doesn't work.

- Moral: Don't share a common *n* among a group of users.

Low Encryption Exponent Attack against RSA

- RSA encryption and signature verification are faster if you use a low value for *e*, but that can also be insecure.

- Moral: Pad messages with random values before encrypting them; make sure *m* is about the same size as *n*.

Low Decryption Exponent Attack against RSA

- Another attack, this one by Michael Wiener, will recover *d*, when *d* is up to one quarter the size of *n* and *e* is less than *n*.
- This rarely occurs if *e* and *d* are chosen at random, and cannot occur if *e* has a small value.
- Moral: Choose a large value for *d*.

Lessons Learned

- Judith Moore lists several restrictions on the use of RSA, based on the success of these attacks:
  - Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to factor the modulus.
  - Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to calculate other encryption/decryption pairs without having to factor *n*.
  - A common modulus should not be used in a protocol using RSA in a communications network.
  - Messages should be padded with random values to prevent attacks on low encryption exponents.
  - The decryption exponent should be large.

Most important lesson

- Remember, it is not enough to have a secure cryptographic algorithm.
- The entire cryptosystem must be secure, and the cryptographic protocol must be secure.
- A failure in any of those three areas makes the overall system insecure.

Attack on Encrypting and Signing with RSA

- It makes sense to sign a message before encrypting it, but not everyone follows this practice.
- With RSA, there is an attack against protocols that encrypt before signing.

Standards and Patents

- RSA is a *de facto* standard in much of the world. The ISO almost, but not quite, created an RSA digital-signature standard; RSA is in an information annex to ISO 9796.
- MIT was granted U.S. Patent 4,405,829 for a "Cryptographic communications system and method" that used the algorithm in 1983.
- The patent would have expired on September 21, 2000 (the term of patent was 17 years at the time), but the algorithm was released to the public domain by RSA Security on 6 September 2000, two weeks earlier.

Outline

- Public-Key Algorithms
  - Background
  - Knapsack Algorithms
  - RSA
  - Pohlig-Hellman
  - Rabin
  - ElGamal

Pohlig-Hellman

- The Pohlig-Hellman encryption scheme is similar to RSA.
- It is not a symmetric algorithm, because different keys are used for encryption and decryption.
- It is not a public-key scheme, because the keys are easily derivable from each other;
- Both the encryption and decryption keys must be kept secret.

Pohlig-Hellman

- Like RSA,

  $C = P^e \bmod n$

  $P = C^d \bmod n$ where

  ed ≡ 1 (mod some complicated number)

- Unlike RSA, n is not defined in terms of two large primes, it must remain part of the secret key.
- If someone had e and n, they could calculate d.
- Without knowledge of e or d, an adversary would be forced to calculate

  $e = \log_P C \bmod n$

- This is a hard problem.

Outline

- Public-Key Algorithms
  - Background
  - Knapsack Algorithms
  - RSA
  - Pohlig-Hellman
  - Rabin
  - ElGamal

Rabin

- Rabin's scheme gets its security from the difficulty of finding square roots modulo a composite number.
- This problem is equivalent to factoring. Here is one implementation of this scheme.
- First choose two primes, p and q, both congruent to 3 mod 4. These primes are the private key; the product $n = pq$ is the public key.
- To encrypt a message, M (M must be less than n), simply compute
  - $C = M^2 \bmod n$
- Rabin is completely insecure against a chosen-ciphertext attack.

Outline

- Public-Key Algorithms
  - Background
  - Knapsack Algorithms
  - RSA
  - Pohlig-Hellman
  - Rabin
  - ElGamal

ElGamal

- The ElGamal scheme can be used for both digital signatures and encryption.
- It gets its security from the difficulty of calculating discrete logarithms in a finite field.
- To generate a key pair, first choose a prime, *p*, and two random numbers, *g* and *x*, such that both *g* and *x* are less than *p*. Then calculate
  - $y = g^x \bmod p$
- The public key is y, g, and p. Both g and p can be shared among a group of users. The private key is x.

ElGamal Signatures

- To sign a message, *M*, first choose a random number, *k*, such that *k* is relatively prime to *p* - 1. Then compute

  $a = g^k \bmod p$

  and use the extended Euclidean algorithm to solve for b in the following equation:

  $M = (xa + kb) \bmod (p - 1)$

- The signature is the pair: a and b. The random value, k, must be kept secret.
- To verify a signature, confirm that

  $y^a a^b \bmod p = g^M \bmod p$

ElGamal Encryption

- A modification of ElGamal can encrypt messages.
- To encrypt message M, first choose a random k, such that k is relatively prime to p - 1. Then compute

  $a = g^k \bmod p$

  $b = y^k M \bmod p$
- The pair, a and b, is the ciphertext. Note that the ciphertext is twice the size of the plaintext.
- To decrypt a and b, compute
  - $M = b/a^x \bmod p$

End of Lecture

- Readings
  - Applied Crypto: Chapters 18 and 19.