

COMP202
Complexity of Algorithms
Number Theory and Cryptography
[See relevant Sections of
Chapter 10 in Goodrich and Tamassia.]

Learning Outcomes

At the conclusion of this set of lecture notes, you should:

1. Be familiar with the basic ideas of cryptography.
2. Have comfortable knowledge of the RSA encryption/decryption method.
3. Understand the mathematical background that underlies the RSA method, including the Euclidean algorithm, etc.

Cryptographic communications

Throughout history there has often been the need (or desire) to *securely* transmit information through *insecure* channels. Such applications include communications for military purposes and business reasons (to keep proprietary information secure), and most recently, transactions through the Internet.

Cryptographic communications

Throughout history there has often been the need (or desire) to *securely* transmit information through *insecure* channels. Such applications include communications for military purposes and business reasons (to keep proprietary information secure), and most recently, transactions through the Internet.

A variety of *cryptographic* methods have been developed to facilitate this type of communication.

These methods include encryption/decryption transformations and digital signatures.

Encryption schemes

Confidentiality in communication can be achieved by *encryption schemes*, or *ciphers*.

Encryption schemes

Confidentiality in communication can be achieved by *encryption schemes*, or *ciphers*.

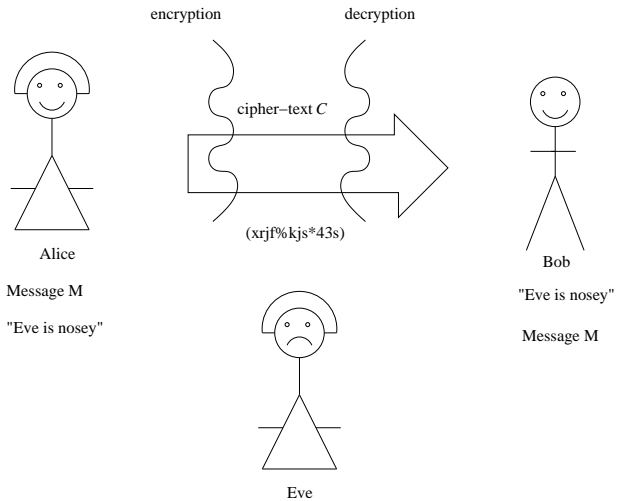
The general idea behind these schemes is that the message M to be sent, often referred to as the *plaintext*, is encrypted (before transmission) into an unrecognizable string (we hope!) of characters C , the *ciphertext*.

Encryption schemes

Confidentiality in communication can be achieved by *encryption schemes*, or *ciphers*.

The general idea behind these schemes is that the message M to be sent, often referred to as the *plaintext*, is encrypted (before transmission) into an unrecognizable string (we hope!) of characters C , the *ciphertext*.

The ciphertext C is then transmitted to the recipient who decrypts C to recover the original message M .



Symmetric encryption schemes and secret keys

In a traditional encryption scheme a common secret key, K , is shared by Alice and Bob.

Symmetric encryption schemes and secret keys

In a traditional encryption scheme a common secret key, K , is shared by Alice and Bob.

This common key K is used for both encryption and decryption of messages.

Symmetric encryption schemes and secret keys

In a traditional encryption scheme a common secret key, K , is shared by Alice and Bob.

This common key K is used for both encryption and decryption of messages.

Such an encryption scheme is called *symmetric* since the recipient and receiver both have access to the same secret key, and it is used for both the encryption and decryption processes.

Substitution ciphers

A classic example of a symmetric cipher is a *substitution* cipher.

In this case the secret key is a *permutation* π of the characters of the alphabet. (For example, each *A* gets replaced by the letter *D*, each *B* gets replaced by the letter *H*, etc.)

Substitution ciphers

A classic example of a symmetric cipher is a *substitution* cipher.

In this case the secret key is a *permutation* π of the characters of the alphabet. (For example, each *A* gets replaced by the letter *D*, each *B* gets replaced by the letter *H*, etc.)

Encryption of the plaintext M is accomplished by mapping each character x with its corresponding character $y = \pi(x)$.

Substitution ciphers

A classic example of a symmetric cipher is a *substitution* cipher.

In this case the secret key is a *permutation* π of the characters of the alphabet. (For example, each A gets replaced by the letter D , each B gets replaced by the letter H , etc.)

Encryption of the plaintext M is accomplished by mapping each character x with its corresponding character $y = \pi(x)$.

Decrypting the ciphertext C is easily accomplished with knowledge of the permutation π , i.e. each character y of C is replaced with $x = \pi^{-1}(y)$.

The Caesar cipher

The *Caesar cipher* is an early example of a substitution cipher wherein each character x is replaced by the character $y = (x + k) \bmod n$, where n is the size of the alphabet and the integer k , $1 \leq k < n$, is the secret key.

The Caesar cipher

The *Caesar cipher* is an early example of a substitution cipher wherein each character x is replaced by the character $y = (x + k) \bmod n$, where n is the size of the alphabet and the integer k , $1 \leq k < n$, is the secret key.

This type of cipher is called a “Caesar cipher” because Julius Caesar is known to have used it with $k = 3$.

Breaking substitution ciphers

While very easy to use, substitution ciphers are not secure.

Breaking substitution ciphers

While very easy to use, substitution ciphers are not secure.

The secret key of a substitution cipher is very easily broken by using frequency analysis, based on knowledge of the frequency of the various letters, or groups of letters, in the alphabet being used.

For example, “e” is the most common letter in the English language, followed by “t”, etc.

The One-time pad

Secure symmetric ciphers do exist!

The One-time pad

Secure symmetric ciphers do exist!

In fact, the most secure cipher known is the symmetric cipher that's referred to as the *one-time pad*.

The One-time pad

Secure symmetric ciphers do exist!

In fact, the most secure cipher known is the symmetric cipher that's referred to as the *one-time pad*.

For this encryption scheme Alice and Bob share a *random* bit string K that is as long as any message that they are going to send.

This key K is the symmetric key that is used for the encryption and decryption process.

The One-time pad (encryption)

To encrypt the message M , Alice computes $C = M \oplus K$, where the \oplus symbol denotes the bitwise “exclusive-or” operation.

(Note: $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$.)

Alice then sends C to Bob on any reliable communications channel.

The One-time pad (encryption)

To encrypt the message M , Alice computes $C = M \oplus K$, where the \oplus symbol denotes the bitwise “exclusive-or” operation.

(Note: $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$.)

Alice then sends C to Bob on any reliable communications channel.

The communication is secure because C is computationally indistinguishable from a *random* bit string. (This relies highly on the fact that K was selected randomly!!)

The One-time pad (decryption)

Bob can easily decrypt the ciphertext C to recover M in the following fashion:

$$C \oplus K = (M \oplus K) \oplus K = M \oplus (K \oplus K) = M \oplus \mathbf{0} = M$$

where $\mathbf{0}$ represents the all-zero string with the same length as M .

This is clearly a symmetric scheme as Alice and Bob use the same key K for encryption and decryption.

The One-time pad (analysis)

Advantages:

- ▶ Computationally efficient since the bitwise exclusive-or is easy to perform.
- ▶ Very secure (provided K is chosen randomly)!!

The One-time pad (analysis)

Advantages:

- ▶ Computationally efficient since the bitwise exclusive-or is easy to perform.
- ▶ Very secure (provided K is chosen randomly)!!

Disadvantages:

- ▶ Alice and Bob must share a very long key K .
- ▶ Security depends on the fact that the key is used *only once*!!

The One-time pad (analysis)

Advantages:

- ▶ Computationally efficient since the bitwise exclusive-or is easy to perform.
- ▶ Very secure (provided K is chosen randomly)!!

Disadvantages:

- ▶ Alice and Bob must share a very long key K .
- ▶ Security depends on the fact that the key is used *only once*!!

In practice, we prefer secret keys that can be reused, and that the keys we use are much shorter than the messages that we must transmit.

How can we do this?

Public-key cryptography

A major problem with symmetric encryption schemes is *key distribution*, or how to *securely distribute* the secret keys.

One idea is to dispense with using *symmetric* encryption schemes and seek another method for generating (and deciphering) the ciphertexts.

Public-key cryptography

A major problem with symmetric encryption schemes is *key distribution*, or how to *securely distribute* the secret keys.

One idea is to dispense with using *symmetric* encryption schemes and seek another method for generating (and deciphering) the ciphertexts.

In 1976 Diffie and Hellman described an *abstract* system that overcomes the problem of key distribution – *Public-key cryptosystems*.

Public-key cryptosystems

A *public-key cryptosystem* consists of an encryption function E and a decryption function D . For any message M , the following properties must hold:

- ▶ $D(E(M)) = M$.
- ▶ Both E and D are “easy” to compute.
- ▶ It is *computationally infeasible* to derive D from E .
- ▶ $E(D(M)) = M$.

Public-key cryptosystems (cont.)

The third property is the particularly important one. It means that knowledge of the encryption method gives *no information* about the decryption scheme. Anybody can send a private message to the holder of the function D , but *only that person* knows how to decrypt it.

Public-key cryptosystems (cont.)

The third property is the particularly important one. It means that knowledge of the encryption method gives *no information* about the decryption scheme. Anybody can send a private message to the holder of the function D , but *only that person* knows how to decrypt it.

For this reason E is referred to as a *one-way* function, or sometimes a *trapdoor function*.

Public-key cryptosystems (cont.)

The third property is the particularly important one. It means that knowledge of the encryption method gives *no information* about the decryption scheme. Anybody can send a private message to the holder of the function D , but *only that person* knows how to decrypt it.

For this reason E is referred to as a *one-way* function, or sometimes a *trapdoor function*.

In this kind of encryption method E is made *public* and D is kept *private*.

Public-key cryptosystems (cont.)

The third property is the particularly important one. It means that knowledge of the encryption method gives *no information* about the decryption scheme. Anybody can send a private message to the holder of the function D , but *only that person* knows how to decrypt it.

For this reason E is referred to as a *one-way* function, or sometimes a *trapdoor function*.

In this kind of encryption method E is made *public* and D is kept *private*.

The fourth property allows for *digital signatures*. This can allow someone to send a message to another person and the *recipient can verify that it came from the sender*, assuming that the sender is the only person who has the private key. (More on this later.)

The RSA encryption scheme

Diffie and Hellman's idea was ingenious, but it was an abstract concept about how such a system would operate.

The RSA encryption scheme

Diffie and Hellman's idea was ingenious, but it was an abstract concept about how such a system would operate.

Rivest, Shamir, and Adleman proposed a public-key encryption method that is probably the most well-known, and is still in use today for communications via web-browsers, etc.

Their method is tied to the difficulty of *factoring* large numbers.

The RSA encryption scheme

Diffie and Hellman's idea was ingenious, but it was an abstract concept about how such a system would operate.

Rivest, Shamir, and Adleman proposed a public-key encryption method that is probably the most well-known, and is still in use today for communications via web-browsers, etc.

Their method is tied to the difficulty of *factoring* large numbers.

Before we can get into the details of the RSA method, we must first discuss some concepts from the branch of mathematics called “number theory”.

Elementary number theory - Divisibility

Given integers a and b , we use the notation $a|b$ to denote that a divides b , i.e. b is a multiple of a .

If $a|b$ then there is another integer k such that $b = a \cdot k$.

Elementary number theory - Divisibility

Given integers a and b , we use the notation $a|b$ to denote that a divides b , i.e. b is a multiple of a .

If $a|b$ then there is another integer k such that $b = a \cdot k$.

We note the following properties about this divisibility relationship:

Theorem: Let a, b, c be integers.

1. If $a|b$ and $b|c$, then $a|c$. (transitivity)
2. If $a|b$ and $a|c$, then $a|(i \cdot b + j \cdot c)$ for all integers i and j .
3. If $a|b$ and $b|a$ then either $a = b$ or $a = -b$.

Prime numbers and composite numbers

An integer $n \geq 2$ is said to be *prime* if the only divisors of n are the trivial divisors 1 and n .

An integer $n \geq 2$ that is not prime is said to be *composite*.

Prime numbers and composite numbers

An integer $n \geq 2$ is said to be *prime* if the only divisors of n are the trivial divisors 1 and n .

An integer $n \geq 2$ that is not prime is said to be *composite*.

For example, 11, 107, and 98711 are prime, but 25, 69, and $10403 = 101 \cdot 103$ are composite.

Fundamental Theorem of Arithmetic

Theorem: Let $n > 1$ be an integer. Then there is a unique set of prime numbers $\{p_1, \dots, p_k\}$ and positive integers $\{e_1, \dots, e_k\}$ such that

$$n = p_1^{e_1} \cdots p_k^{e_k}.$$

The product $p_1^{e_1} \cdots p_k^{e_k}$ is known as the *prime decomposition* of n . It is unique, up to the ordering of the primes in the factorization.

Greatest common divisor (GCD)

Let a and b denote positive integers. The *greatest common divisor* of a and b , denoted $\gcd(a, b)$, is the *largest* integer that divides both a and b .

Greatest common divisor (GCD)

Let a and b denote positive integers. The *greatest common divisor* of a and b , denoted $\gcd(a, b)$, is the *largest* integer that divides both a and b .

If $\gcd(a, b) = 1$, then we say that a and b are *relatively prime*.

Greatest common divisor (GCD)

Let a and b denote positive integers. The *greatest common divisor* of a and b , denoted $\gcd(a, b)$, is the *largest* integer that divides both a and b .

If $\gcd(a, b) = 1$, then we say that a and b are *relatively prime*.

The definition of the GCD can be extended in a natural fashion:

- ▶ If $a > 0$, then $\gcd(a, 0) = \gcd(0, a) = a$.
- ▶ $\gcd(a, b) = \gcd(|a|, |b|)$ if a and/or b is negative.

For example, $\gcd(56, 24) = 8$, $\gcd(25, 31) = 1$, $\gcd(45, 25) = 5$, and $\gcd(-27, 51) = 3$.

Greatest common divisor (GCD)

Let a and b denote positive integers. The *greatest common divisor* of a and b , denoted $\gcd(a, b)$, is the *largest* integer that divides both a and b .

If $\gcd(a, b) = 1$, then we say that a and b are *relatively prime*.

The definition of the GCD can be extended in a natural fashion:

- ▶ If $a > 0$, then $\gcd(a, 0) = \gcd(0, a) = a$.
- ▶ $\gcd(a, b) = \gcd(|a|, |b|)$ if a and/or b is negative.

For example, $\gcd(56, 24) = 8$, $\gcd(25, 31) = 1$, $\gcd(45, 25) = 5$, and $\gcd(-27, 51) = 3$.

Note that $\gcd(0, 0)$ is undefined (as, of course, there is no largest integer that divides 0).

Greatest common divisor (cont.)

We note the following important fact:

Theorem: If $d = \gcd(a, b)$, then there exist (unique) integers j and k such that

$$d = j \cdot a + k \cdot b.$$

In other words, the greatest common divisor of a and b is expressible as a *linear combination* of a and b .

Greatest common divisor (cont.)

We note the following important fact:

Theorem: If $d = \gcd(a, b)$, then there exist (unique) integers j and k such that

$$d = j \cdot a + k \cdot b.$$

In other words, the greatest common divisor of a and b is expressible as a *linear combination* of a and b .

$$\gcd(56, 24) = 8 \quad 8 = 1 \cdot 56 + (-2) \cdot (24)$$

$$\gcd(25, 31) = 1 \quad 1 = 5 \cdot 25 + (-4) \cdot 31$$

$$\gcd(45, 25) = 5 \quad 5 = (-1) \cdot 45 + 2 \cdot 25$$

$$\gcd(57, 363) = 3 \quad 3 = 57 \cdot 51 + (-8) \cdot 363$$

The modulo operator and congruences

The modulo operator, denoted by $a \bmod b$, defines the remainder of a when divided by b . That is $r = a \bmod b$ means that $r = a - \lfloor \frac{a}{b} \rfloor b$.

In other words, r is always an integer in the set $\{0, 1, 2, \dots, b - 1\}$ (even when a is negative), and there is an integer q such that

$$a = q \cdot b + r.$$

The modulo operator and congruences

The modulo operator, denoted by $a \bmod b$, defines the remainder of a when divided by b . That is $r = a \bmod b$ means that $r = a - \lfloor \frac{a}{b} \rfloor b$.

In other words, r is always an integer in the set $\{0, 1, 2, \dots, b - 1\}$ (even when a is negative), and there is an integer q such that

$$a = q \cdot b + r.$$

Sometimes we find it convenient to talk about *congruence modulo n* . If

$$a \bmod n = b \bmod n,$$

we say that a is *congruent to b modulo n* and write

$$a \equiv b \pmod{n}.$$

If $a \equiv b \pmod{n}$, then $a - b = kn$ for some integer k .

Examples of congruences and modular arithmetic

- ▶ $27 = 2 \pmod{5}$
- ▶ $12 = 0 \pmod{3}$
- ▶ $91 = 1 \pmod{10}$
- ▶ $-10 = 2 \pmod{3}$ (Note that $\lfloor \frac{-10}{3} \rfloor = -4$.)

- ▶ $16 \equiv 8 \pmod{2}$ as $16 - 8$ is a multiple of 2.
- ▶ $97 \equiv 6 \pmod{7}$
- ▶ $80 \equiv -1 \pmod{3}$

Euclid's algorithm

Euclid's algorithm is a method to find the greatest common divisor of two integers a and b .

Euclid's algorithm

Euclid's algorithm is a method to find the greatest common divisor of two integers a and b .

This algorithm relies on an important property that involves the modulo operator, namely, the following:

Theorem: Suppose that $a \geq b > 0$. Then

$$\gcd(a, b) = \gcd(b, a \bmod b).$$

Euclid's algorithm (cont.)

EUCLIDGCD(a, b)

Input: Nonnegative integers a and b (not both zero).

Output: $\gcd(a, b)$.

```
1 while  $b \neq 0$  do  
2      $(a, b) \leftarrow (b, a \bmod b)$   
3 return  $a$ 
```

Note: If $b = 0$, this routine will return the value of a (which, by our assumption on the input, is not zero), giving the correct result.

Examples of Euclid's algorithm

	1	2	3	4	5	6	7
<i>a</i>	412	260	152	108	44	20	4
<i>b</i>	260	152	108	44	20	4	0

Hence, $\gcd(412, 260) = 4$.

	1	2	3	4	5
<i>a</i>	408	162	84	78	6
<i>b</i>	162	84	78	6	0

Thus, $\gcd(408, 162) = 6$.

Euclid's algorithm - Complexity

For $i > 0$, let a_i denote the first element of the ordered pair during the i th step in the “while” loop in Euclid's algorithm. The second argument is equal to a_{i+1} . So

$$a_{i+2} = a_i \bmod a_{i+1}.$$

This implies that, after the first time through the loop, the sequence a_i is strictly decreasing.

Euclid's algorithm - Complexity

For $i > 0$, let a_i denote the first element of the ordered pair during the i th step in the “while” loop in Euclid's algorithm. The second argument is equal to a_{i+1} . So

$$a_{i+2} = a_i \bmod a_{i+1}.$$

This implies that, after the first time through the loop, the sequence a_i is strictly decreasing.

We can show that $a_{i+2} < \frac{1}{2} \cdot a_i$.

This leads to the following result:

Theorem: Let a, b be two positive integers. Euclid's algorithm computes $\gcd(a, b)$ by executing $O(\log(\max\{a, b\}))$ arithmetic operations.

The Extended Euclidean Algorithm

As mentioned earlier, if $d = \gcd(a, b)$, there are integers j and k such that $d = j \cdot a + k \cdot b$.

We can modify Euclid's algorithm to find these numbers j and k while we compute $\gcd(a, b)$. This is the so-called "Extended Euclidean algorithm."

The Extended Euclidean Algorithm (cont.)

EXTENDED EUCLIDGCD(a, b)

Input: Nonnegative integers a and b (not both zero).

Output: $d = \gcd(a, b)$, integers j, k where $d = j \cdot a + k \cdot b$.

- 1 **if** $b = 0$ **then**
- 2 return ($a, 1, 0$)
- 3 $r \leftarrow a \bmod b$
- 4 Let q be the integer such that $a = q \cdot b + r$ (that is, $q = \lfloor \frac{a}{b} \rfloor$).
- 5 $(d, j, k) \leftarrow \text{ExtendedEuclidGCD}(b, r)$
- 6 return ($d, k, j - kq$)

Extended Euclidean Algorithm examples

Note that by the recursive nature of the algorithm, the values of a , b , q , and r are filled in the table from left-to-right, but those of j and k are filled in from right-to-left.

To find $\gcd(412, 260)$ we have these values:

	1	2	3	4	5	6	7
a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0
$q = \lfloor \frac{a}{b} \rfloor$	1	1	1	2	2	5	*
$r = a \bmod b$	152	108	44	20	4	0	*
j	12	-7	5	-2	1	0	1
k	-19	12	-7	5	-2	1	0

As can be checked, we have $12 \cdot 412 + (-19) \cdot 260 = 4$.

Extended Euclidean Algorithm examples (cont.)

Finding $\gcd(408, 162)$ gives this table:

	1	2	3	4	5
<i>a</i>	408	162	84	78	6
<i>b</i>	162	84	78	6	0
<i>q</i>	2	1	1	13	*
<i>r</i>	84	78	6	0	*
<i>j</i>	2	-1	1	0	1
<i>k</i>	-5	2	-1	1	0

We see that $2 \cdot 408 + (-5) \cdot 162 = 6$, as guaranteed previously.

The Extended Euclidean algorithm is useful for the RSA encryption method, as we will soon see.

The RSA method

Given the previous information (review?) about some very basic number theory, we are now ready to describe the basic method of the RSA encryption/decryption method.

The RSA method

Given the previous information (review?) about some very basic number theory, we are now ready to describe the basic method of the RSA encryption/decryption method.

Recall that this is a public-key encryption method. So this is a *non-symmetric* encryption scheme, where there is an encryption function and a separate decryption function.

The RSA method

Given the previous information (review?) about some very basic number theory, we are now ready to describe the basic method of the RSA encryption/decryption method.

Recall that this is a public-key encryption method. So this is a *non-symmetric* encryption scheme, where there is an encryption function and a separate decryption function.

The main idea of the RSA method is that I can publish my encryption function (i.e. make it freely available to anyone who wishes to use it to send a message to me), but only I know the decryption function.

The RSA method (cont.)

Let p and q denote two (large) prime numbers.

Let $n = p \cdot q$ and define $\phi(n) = (p - 1)(q - 1)$.

The RSA method (cont.)

Let p and q denote two (large) prime numbers.

Let $n = p \cdot q$ and define $\phi(n) = (p - 1)(q - 1)$.

We then choose two numbers e and d such that

1. e and $\phi(n)$ are relatively prime, i.e. $\gcd(e, \phi(n)) = 1$, and
2. $ed \equiv 1 \pmod{\phi(n)}$.

(We can use the Extended Euclidean algorithm to find d , given e .)

The RSA method (cont.)

Let p and q denote two (large) prime numbers.

Let $n = p \cdot q$ and define $\phi(n) = (p - 1)(q - 1)$.

We then choose two numbers e and d such that

1. e and $\phi(n)$ are relatively prime, i.e. $\gcd(e, \phi(n)) = 1$, and
2. $ed \equiv 1 \pmod{\phi(n)}$.

(We can use the Extended Euclidean algorithm to find d , given e .)

The pair of values e and n form the *public* key and d is the *private* key.

Encryption via RSA

Let us assume that the message M is an integer and that $0 < M < n$.

(We can suppose that M is obtained by concatenating the bits of the ASCII representation of its characters, for example, and break up M into blocks of appropriate sizes if $M \geq n$. This characterizes a *padding scheme* which we won't discuss here.)

Encryption via RSA

Let us assume that the message M is an integer and that $0 < M < n$.

(We can suppose that M is obtained by concatenating the bits of the ASCII representation of its characters, for example, and break up M into blocks of appropriate sizes if $M \geq n$. This characterizes a *padding scheme* which we won't discuss here.)

The plaintext M is encrypted using the public keys e and n by the following operation:

$$C \leftarrow M^e \pmod n.$$

Decryption with RSA

Decryption of the received ciphertext, C , is again handled by modular exponentiation:

$$M \leftarrow C^d \pmod{n}.$$

Decryption with RSA

Decryption of the received ciphertext, C , is again handled by modular exponentiation:

$$M \leftarrow C^d \pmod{n}.$$

The correctness of the RSA method is guaranteed because it can be shown that with the choices of e , n , and d (with the properties listed earlier), then for every integer $0 < x < n$ we have

$$x^{ed} \equiv x \pmod{n}.$$

Digital signatures

As mentioned earlier, the RSA cryptosystem supports *digital signatures*. Suppose that Bob sends a message M to Alice and that Alice wants to *verify* that it was Bob who sent it. Bob can create a *signature* using the decryption function applied to M :

$$S \leftarrow M^d \pmod{n}.$$

Digital signatures

As mentioned earlier, the RSA cryptosystem supports *digital signatures*. Suppose that Bob sends a message M to Alice and that Alice wants to *verify* that it was Bob who sent it. Bob can create a *signature* using the decryption function applied to M :

$$S \leftarrow M^d \pmod{n}.$$

Alice verifies the digital signature using the encryption function, that is by checking that

$$M \equiv S^e \pmod{n}.$$

Digital signatures

As mentioned earlier, the RSA cryptosystem supports *digital signatures*. Suppose that Bob sends a message M to Alice and that Alice wants to *verify* that it was Bob who sent it. Bob can create a *signature* using the decryption function applied to M :

$$S \leftarrow M^d \pmod{n}.$$

Alice verifies the digital signature using the encryption function, that is by checking that

$$M \equiv S^e \pmod{n}.$$

Since only Bob knows the decryption function, this will verify that it was indeed Bob who sent the message. (Of course, *any* person can use the encryption function as well to reconstruct the message M , so this is not a method to *secretly* pass information from Bob to Alice.)

The difficulty of breaking RSA

Note that even knowing e doesn't allow us to figure out d , unless we know $\phi(n)$.

The difficulty of breaking RSA

Note that even knowing e doesn't allow us to figure out d , unless we know $\phi(n)$.

Most cryptographers believe that breaking RSA requires the computation of $\phi(n)$, which in turn requires factoring n .

Factoring has *not* been *proven* to be difficult, but many (many!) people have worked on this problem over the last several hundred years.

The difficulty of breaking RSA

Note that even knowing e doesn't allow us to figure out d , unless we know $\phi(n)$.

Most cryptographers believe that breaking RSA requires the computation of $\phi(n)$, which in turn requires factoring n .

Factoring has *not* been *proven* to be difficult, but many (many!) people have worked on this problem over the last several hundred years.

For example, it took some heavy duty mathematics and a network of 700 computers (including one supercomputer) four months to factor the number $2^{512} - 1$ which is 155 digits long.

As the ability to factor larger numbers increases, we simply have to choose larger primes p and q so that $n = p \cdot q$ is outside of the current factoring capabilities.

Fast exponentiation

A possible bottleneck in the RSA algorithm is computing expressions of the form

$$x^k \pmod n.$$

The “naive approach” is to calculate $x^2 \pmod n$, then use that to get $x^3 \pmod n$, then $x^4 \pmod n$, etc.

Fast exponentiation (cont.)

We can do much better with an algorithm based on “repeated squaring.” For example, if we wanted to compute x^{16} , we could first find x^2 , then $(x^2)^2 = x^4$, then $(x^4)^2 = x^8$, and finally $(x^8)^2 = x^{16}$. This requires only four multiplications instead of fifteen with the “naive method.”

If we are performing “modular exponentiation” as in RSA, after each step we can find $x^i \bmod n$ to keep the results small (between 0 and $n - 1$).

Fast exponentiation (cont.)

FASTEXPONENTIATION(x, k, n)

Input: Integers $x, k \geq 0$, and $n > 0$.

Output: $r = x^k \bmod n$.

```
1   $r \leftarrow 1$ 
2   $t \leftarrow x$ 
3  while  $k \neq 0$  do
4      if  $k$  is odd then
5           $r \leftarrow r \cdot t \bmod n$ 
6           $t \leftarrow t^2 \bmod n$ 
7           $k \leftarrow \lfloor k/2 \rfloor$ 
8  return  $r$ 
```

Complexity of RSA

Using the `FASTEXPONENTIATION(x, k, n)` algorithm, the *size* of the operands is never more than $O(\log n)$ bits, and it takes $O(\log k)$ *arithmetic operations* to find $x^k \bmod n$.

This leads to the following result:

Theorem: Let n be the modulus of the RSA algorithm. Then RSA encryption, decryption, signature, and verification each take $O(\log n)$ arithmetic operations (per block).