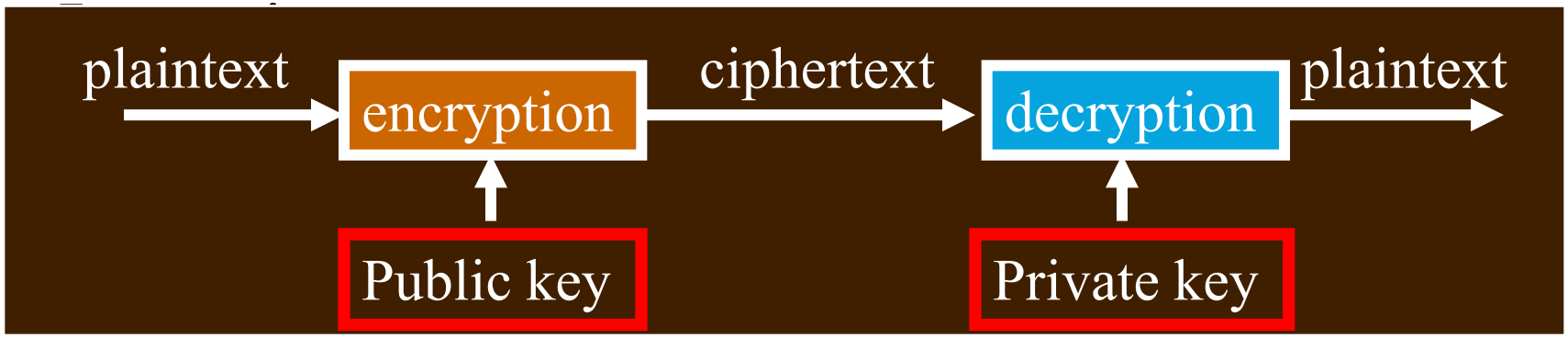
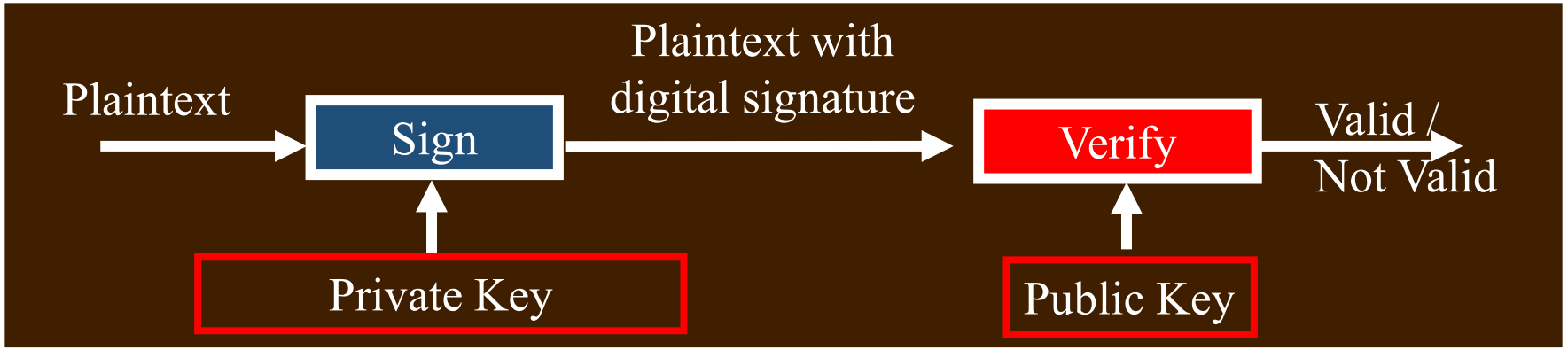
CS 4173/5173
COMPUTER SECURITY

RSA

REVIEW OF PUBLIC KEY



- Authentication



PUBLIC KEY VS. SYMMETRIC KEY

Symmetric key	Public key
Two parties MUST trust each other	Two parties DO NOT need to trust each other
Both share same key (or one key is computable from the other)	Two separate keys: a public and a private key
Typically faster	Typically slower
Examples: DES, RC5, AES, …	Examples: RSA, DSA, ECC…

COMPANY A'S PROBLEM I

- Company A is a big web service company with over 10,000 employees.
- The president Bob wants to make sure that all employees can verify the authenticity of the announcement emails that he sends.
- Q: How to ensure authenticity of these emails.

COMPANY A'S PROBLEM II

- Company A is accepting vulnerability report of their web system from the public.
- They need a design that someone can successfully send the report of a potential vulnerability via email to them.
- Q: How to ensure the confidentiality of reports?

COMPANY A'S PROBLEM III

- Company A provides an on-line chat service for vulnerability report.
  - Requirement 1: confidentiality.
  - Requirement 2: efficiency because there will be a number of message exchanges.

- Q: How to satisfy both requirements?

GREATEST COMMON DIVISOR (GCD)

- *gcd(a,b)* = max{*k* | *k*|*a* and *k*|*b*}

Example: gcd(60,24) = 12, gcd(a,0) = a

- Properties:
  - if 0 ≤ *n*, then gcd(*an*, *bn*) = *n**gcd(*a*,*b*)
  - If gcd(*a*,*b*)=1, *a* and *b* are relatively prime.
  - For all positive integers *d*, *a*, and *b*, if *d* | *ab* and gcd(*a*,*d*) = 1
    - then *d*|*b*
  - Example:
    - 3 | 4*9, gcd(3, 4) = 1, ➔ 3 | 9

MULTIPLICATIVE INVERSES

- Don't always exist!
  - Ex.: there is no *z* such that $6 \times z = 1 \bmod 8$ (m = 6 and n = 8)

z	0	1	2	3	4	5	6	7
6×z	0	6	12	18	24	30	36	42
6×z mod 8	0	6	4	2	0	6	4	2

- A positive integer $m \in \mathbb{Z}_n$ has a multiplicative inverse m^{-1} mod *n* if and only if (iff) gcd(*m, n*) = 1, i.e., *m* and *n* are relatively prime

  $\Rightarrow$ If *n* is a prime number, then all positive elements in $\mathbb{Z}_n$ have multiplicative inverses

ALGORITHMS

- Euclid Algorithm
- Extended Euclid Algorithm

THE TOTIENT FUNCTION

- $\phi(n) = |\mathcal{Z}_n^*|$: the number of elements in $\mathcal{Z}_n^*$.
  - $\mathcal{Z}_n^*$ is the set of integers less than *n* and relatively prime to *n.*
- Examples
  - $\phi(4)$=?
    - *GCD(1, 4) = ?*
    - *GCD(2, 4) = ?*
    - *GCD(3, 4) = ?*
  - $\phi(6)$=?
    - *GCD(1, 6) = ?*
    - *GCD(2, 6) = ?*
    - *GCD(3, 6) = ?*
    - *GCD(4, 6) = ?*
    - *GCD(5, 6) = ?*

PROPERTIES OF TOTIENT FUNCTION

a) if n is prime, then $\phi(n) = n-1$

Example: $\phi(7) = 6$

b) if $n = p^{\alpha}$, where p is prime and $\alpha > 0$, then
$\phi(n) = (p-1)*p^{\alpha-1}$

Example: $\phi(25) = \phi(5^2) = 4*5^1 = 20$

c) if $n=p*q$, and p, q are relatively prime, then
$\phi(n) = \phi(p)*\phi(q)$

Example: $\phi(15) = \phi(5*3) = \phi(5) * \phi(3) = 4 * 2 = 8$

MODULAR EXPONENTIATION

- $a^x \bmod n = a^{x \bmod \phi(n)} \bmod n$
  - *a* and *n* are relatively prime

Example: $5^7 \bmod 6 = 5^{7 \bmod \phi(6)} \bmod 6$

$= 5^{7 \bmod 2} \bmod 6 = 5$

Example: $2^{101} \bmod 33 = 2^{101 \bmod \phi(33)} \bmod 33$

$= 2^{101 \bmod 20} \bmod 33$

$= 2 \bmod 33$

$= 2$

DIFFICULTIES IN MODULAR ARITHMETIC

- Factoring large numbers
- Computing Totient function
  - Need factoring first
- Obtaining primitive roots
- Discrete logarithm

- Public key cryptography design should leverage all these difficulties!

RSA (RIVEST, SHAMIR, ADLEMAN)

- The most popular public key method
  - provides both public key encryption and digital signatures
- Variable key length (1024 bits or greater)
  - What are lengths for symmetric key schemes?
- Variable plaintext block size
  - plaintext block size must be smaller than key size
  - ciphertext block size is same as key size

The Founders

GENERATING A PUBLIC/PRIVATE KEY PAIR

- Find large primes *p* and *q*
- Let $n = p*q$
  - do not disclose *p* and *q*!
  - $\phi(n) = ???$
- Choose an *e* that is relatively prime to $\phi(n)$
  - **public** key = <*e*,*n*>
- Find *d* = multiplicative inverse of *e* mod $\phi(n)$ (i.e., $e*d = 1 \bmod \phi(n)$) (how?)
  - **private** key = <*d*,*n*>
- Q: Why choose *e* relatively prime to $\phi(n)$?

RSA OPERATIONS

- For plaintext message ***m*** (treated as large a number, why?) and ciphertext ***c***

Encryption: $c = m^e \bmod n, m < n$

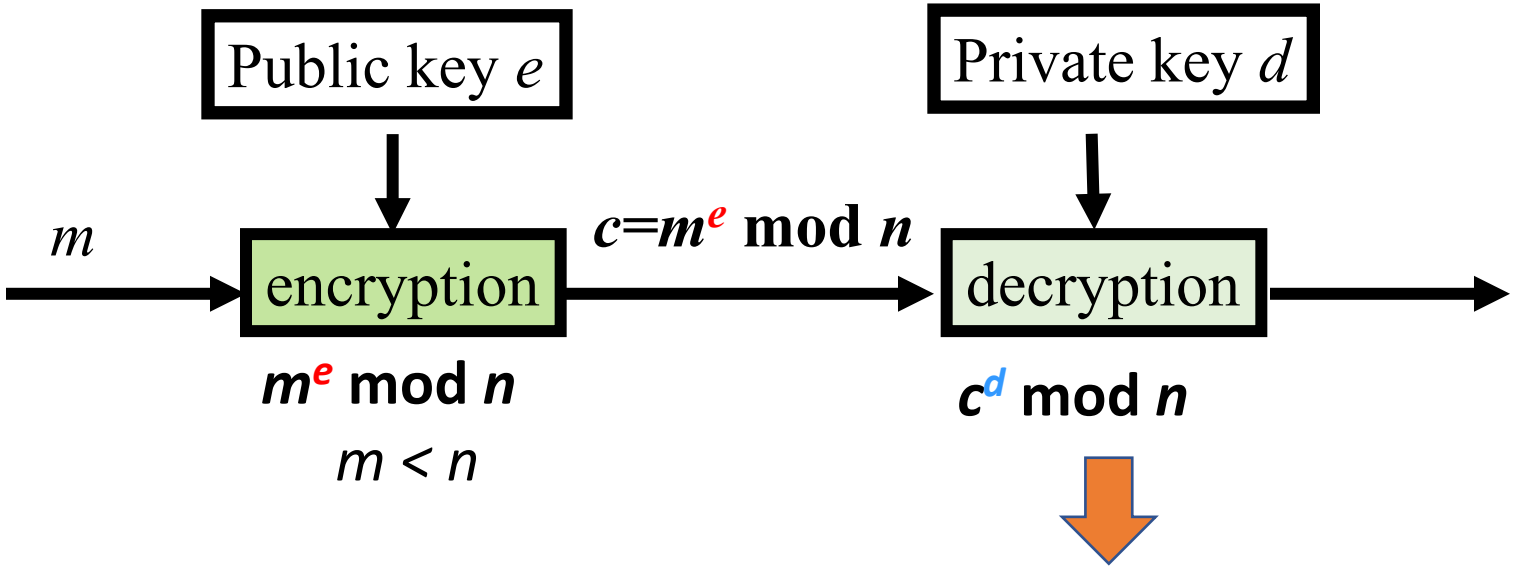
Decryption: $m = c^d \bmod n$

***e* – public key**
***d* – private key**

Signing: $s = m^d \bmod n, m < n$

Verification: $m = s^e \bmod n$

DECRYPTION AND DESCRIPTION FIGURE



$$m' = (m^e \bmod n)^d \bmod n$$

$$= (m^{ed} \bmod n) \bmod n$$

$$= (m^{ed \bmod \phi(n)} \bmod n) \bmod n$$

$$= (m^1 \bmod n) \bmod n = m$$

PROOF (D(E(M)) = M)

- Given
  - public key = <e, n> and private key = <d, n>
  - $n = p*q$, $\phi(n) = (p-1)(q-1)$
  - $e*d \equiv 1 \bmod \phi(n)$
- If encryption is $c = m^e \bmod n$, *d*ecryption…

  $= c^d \bmod n$

  $= (m^e)^d \bmod n = m^{ed} \bmod n$

  $= m \bmod n$

  $= m$

(digital signature proof is similar)

RSA EXAMPLE: ENCRYPTION AND SIGNING

- Choose p = 23, q = 11 (both primes)
  - $n = p*q = 253$
  - $\phi(n) = (p-1)(q-1) = 220$
- Choose e = **39** (relatively prime to 220)
  - public key = <**39**, 253>
- Find e^{-1} mod 220 = d = **79**
  (note: $39*79 \equiv 1 \bmod 220$)
  - private key = <**79**, 253>

EXAMPLE (CONT'D)

- Suppose plaintext **m = 80**

Encryption

$\mathbf{c} = 80^{39} \bmod 253 = \mathbf{37}$ $(c = m^e \bmod n)$

Decryption

$\mathbf{m} = 37^{79} \bmod 253 = \mathbf{80}$ $(c^d \bmod n)$

(Efficient methods are available to compute modular exponentiation)

Signing (in this case, for entire message **m**)

$\mathbf{s} = \mathbf{80}^{79} \bmod 253 = 224$ $(s = m^d \bmod n)$

Verification

$\mathbf{m} = 224^{39} \bmod 253 = \mathbf{80}$ $(s^e \bmod n)$

DISCUSSIONS ON RSA: I

Step 1: Find very large primes p and q, and let $n = p*q$

Q: Primes are so rare, can we find enough prime numbers?

- 2, 3, 5, 7, 11, 13, 17, 19, 23, 31, …
- Yes. There are sufficiently many primes that make exhaust search computationally infeasible.
- Recent advancement of twin primes:
  - There are infinitely many primes with bounded gap!

DISCUSSIONS ON RSA: II

Step 1: Find very large primes p and q, and let $n = p*q$

Q: Is it computationally feasible to find such p and q?

- *Yes.*
  - Primality test
  - Random probable prime generation.
    - Generate a prime with negligible error probability.

Q: Can we make a table so we can efficiently choose p and q from the table?

DISCUSSIONS ON RSA: III

Step2: choose e relatively prime to $\phi(n)$, d is the multiplicative inverse of e.

- $<e,n>$ is public information
- $<d,n>$ is private information
- e and d are multiplicative inverses mod $\phi(n)$

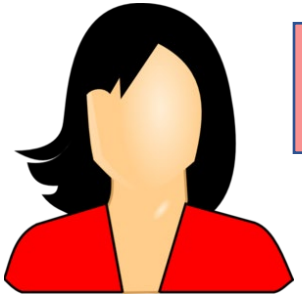
Q: Why is it hard to get $<d,n>$ from $<e,n>$?

We need to get $\phi(n)$, generally we have to factor n first to get $\phi(n)$

- Computationally difficult to factor a large number n!

USING RSA FOR KEY NEGOTIATION

Alice



Public key: Ka,p
Private key: Ka,i

Generate random number R1

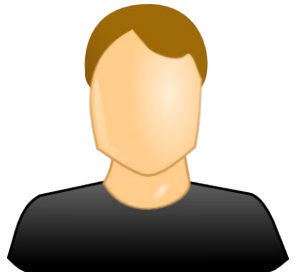
Send R1 encrypted using Kb,p

1) Get R1,
2) Generate random number R2,
3) get key as K = H(*R1*⊕*R2*),

Send R2 encrypted using Ka,p

1) get R2
2) get symmetric key as K = H(*R1*⊕*R2*),

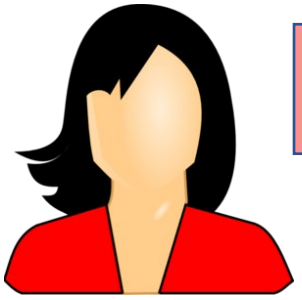
Bob



Public key: Kb,p
Private key: Kb,i

DIGITAL ENVELOPE

Alice



Knows the public key: Kp

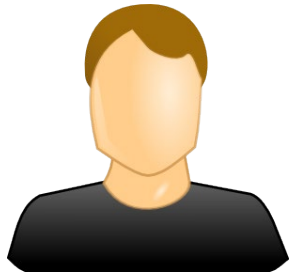
Generate a session key Ks

Encrypt message M using Ks
Encrypted Ks using Kp

Send a response using Ks

All remaining communication is based on Ks

Bob



Public key: Kp
Private key: Ki

1. Use Ki to decrypt Ks
2. Use Ks to decrypt M

GET THE PRIVATE FROM THE PUBLIC

- public information
  - Public key: <*e*,*n*>
  - Ciphertext: $c = m^e \bmod n$
- private information
  - Prime numbers: *p*, *q*
  - Totient function: $\phi(n) = (p-1)(q-1)$
  - Private key: <*d*,*n*>
  - Plaintext: *m*

Hard things:
1. Factoring large numbers
2. Computing Totient function
   Need factoring first
3. Obtaining primitive roots
4. Discrete logarithm

IS RSA SECURE?

- <*e*,*n*> is public information
- If you could factor *n* into *p***q*, then
  - could compute $\phi(n) = (p-1)(q-1)$
  - could compute $d = e^{-1} \bmod \phi(n)$
  - would know the private key <*d*,*n*>!
- But: factoring large integers is hard!
  - classical problem worked on for centuries; no known reliable, fast method
- Shor's quantum algorithm 1994:
  - Efficient factorization to break RSA if we have …

SECURITY (CONT'D)

- At present, key sizes of 1024 bits (i.e., $n=pq$ is on the order of 2^{1024}) are considered to be marginally secure, but 2048 bits or more is better
- Tips for making *n* more difficult to factor
  1. *p* and *q* lengths should be similar (ex.: ~500 bits each if key is 1024 bits)
  2. both (*p*-1) and (*q*-1) should contain a "large" prime factor
  3. gcd(*p*-1, *q*-1) should be "small"
  4. *d* should be larger than $n^{1/4}$

ATTACKS AGAINST RSA

- Brute force: try all possible private keys
  - can be defeated by using a large enough key space (e.g., 1024 bit keys or larger)
- Mathematical attacks
  - factor n into the product of two primes
    - possible for special cases of n
- Shor's algorithm, 1994
  - Breaking RSA and popular public key crypto
  - Quantum algorithm

ATTACKS (CONT'D)

- Probable-message attack (using <*e*,*n*>)
  - encrypt all possible plaintext messages
  - try to find a match between the ciphertext and one of the encrypted messages
  - only works for small plaintext message sizes
- Solution: pad plaintext message with random text before encryption (why random?)
- PKCS #1 v1 (public key standards)
  - specifies this padding format:

each 8 bits long

RSA

- Encryption:
  - Encrypt with public key, decrypt with private key
- Authentication:
  - Sign with private key, verify with public key
- Key Negotiation:
  - Digital envelope
  - RSA based negotiation
  - Alice and Bob must know each other's public key

OTHER SCENARIO

- What if Alice and Bob do not have their public/private keys, but want to communicate secretly?

SOLUTION

- Alice and Bob
  - generate a pair of public and private key individually
    - Public key <*e*, *n*> in RSA
    - Private key <*d*, *n*> in RSA
  - Send their public keys <*e*, *n*> to each other