**18CS744 - CRYPTOGRAPHY**

## Private-Key Cryptography

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender

## Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

# Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community

# Public-Key Cryptography

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

# Public-Key Cryptography
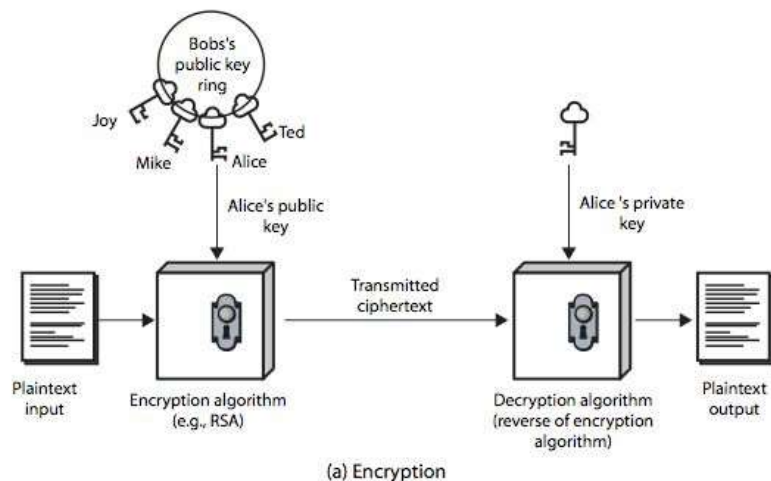


(a) Encryption

# Public-Key Characteristics

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)

# Public-Key Cryptosystems
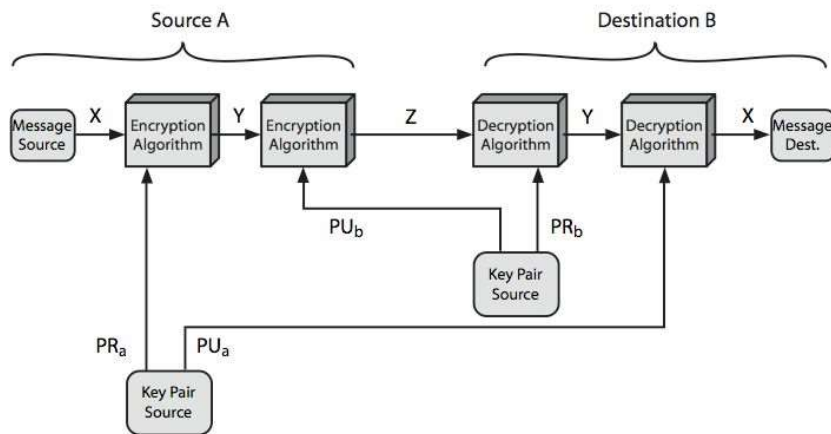


# Public-Key Applications

- can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- some algorithms are suitable for all uses, others are specific to one

# Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems
- more generally the **hard** problem is known, but is made hard enough to be impractical to break
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes

# RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

# RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random - `p, q`
- computing their system modulus `n=p.q`
  - note $\varnothing(n)=(p-1)(q-1)$
- selecting at random the encryption key `e`
  - where `1<e<`$\varnothing$`(n), gcd(e,`$\varnothing$`(n))=1`
- solve following equation to find decryption key `d`
  - `e.d=1 mod `$\varnothing$`(n) and 0≤d≤n`
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

# RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient `PU={e,n}`
  - computes: `C = `$M^e$` mod n`, where 0≤M<n
- to decrypt the ciphertext C the owner:
  - uses their private key `PR={d,n}`
  - computes: `M = `$C^d$` mod n`
- note that the message M must be smaller than the modulus n (block if needed)

# Why RSA Works

- because of Euler's Theorem:
  - $a^{\varnothing(n)}$`mod n = 1` where `gcd(a,n)=1`
- in RSA have:
  - `n=p.q`
  - $\varnothing$`(n)=(p-1)(q-1)`
  - carefully chose `e` & `d` to be inverses `mod `$\varnothing$`(n)`
  - hence `e.d=1+k.`$\varnothing$`(n)` for some `k`
- hence :
  $C^d = M^{e \cdot d} = M^{1+k.\varnothing(n)} = M^1 . (M^{\varnothing(n)})^k$
  $= M^1 . (1)^k = M^1 = M \bmod n$

# RSA Example - Key Setup

1. Select primes: $p$=17 & $q$=11
2. Compute $n = pq$ =17 x 11=187
3. Compute $\varnothing(n)=(p-1)(q-1)$=16 x 10=160
4. Select `e`: `gcd(e,160)=1;` choose $e$=7
5. Determine d: $de$=1 mod 160 and $d < 160$
   Value is d=23 since 23x7=161= 10x160+1
6. Publish public key `PU={7,187}`
7. Keep secret private key `PR={23,187}`

# RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message `M = 88` (nb. `88<187`)
- encryption:

  $$C = 88^7 \bmod 187 = 11$$

- decryption:

  $$M = 11^{23} \bmod 187 = 88$$

# Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

# Exponentiation

```
c = 0; f = 1
for i = k downto 0
    do c = 2 x c
        f = (f x f) mod n
    if b   == 1 then
         i
        c = c + 1
        f = (f x a) mod n
return f
```

# Efficient Encryption

- encryption uses exponentiation to power e
- hence if e small, this will be faster
  - often choose e=65537 ($2^{16}$-1)
  - also see choices of e=3 or e=17
- but if e too small (eg e=3) can attack
  - using Chinese remainder theorem & 3 messages with different modulii
- if e fixed must ensure `gcd(e,ø(n))=1`
  - ie reject any p or q not relatively prime to e

# Efficient Decryption

- decryption uses exponentiation to power d
  - this is likely large, insecure if not
- can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique

# RSA Key Generation

- users of RSA must:
  - determine two primes at random - `p`, `q`
  - select either `e` or `d` and compute the other
- primes `p`, `q` must not be easily derived from modulus `n=p.q`
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents `e`, `d` are inverses, so use Inverse algorithm to compute the other

# RSA Security

- possible approaches to attacking RSA are:
  - brute force key search (infeasible given size of numbers)
  - mathematical attacks (based on difficulty of computing ∅(n), by factoring modulus n)
  - timing attacks (on running of decryption)
  - chosen ciphertext attacks (given properties of RSA)

# Factoring Problem

- mathematical approach takes 3 forms:
  - factor `n=p.q`, hence compute ∅(n) and then d
  - determine ∅(n) directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS
  - biggest improvement comes from improved algorithm
    - cf QS to GHFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

# Timing Attacks

- developed by Paul Kocher in mid-1990's
- exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

# Chosen Ciphertext Attacks

RSA is vulnerable to a Chosen Ciphertext
Attack (CCA)
attackers chooses ciphertexts & gets decrypted
plaintext back
choose ciphertext to exploit properties of RSA
to provide info to help cryptanalysis
can counter with random pad of plaintext
or use Optimal Asymmetric Encryption
Padding (OASP)

# Summary

- have considered:
  - principles of public-key cryptography
  - RSA algorithm, implementation, security