Public Key Cryptography and RSA

Chapter 9

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
- Two keys has profound consequences on the areas of confidentiality, key distribution and authentication
- **Asymmetric** since parties are **not** equal
- Complements **rather than** replaces private key cryptography

Public-Key Cryptography

- Based on mathematical functions rather than on substitution and permutation
- Uses clever application of number theoretic concepts to function
- More secure than symmetric encryption
- The security of any encryption schemes depends on
  - The length of the key
  - The computational work involved in breaking a cipher

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

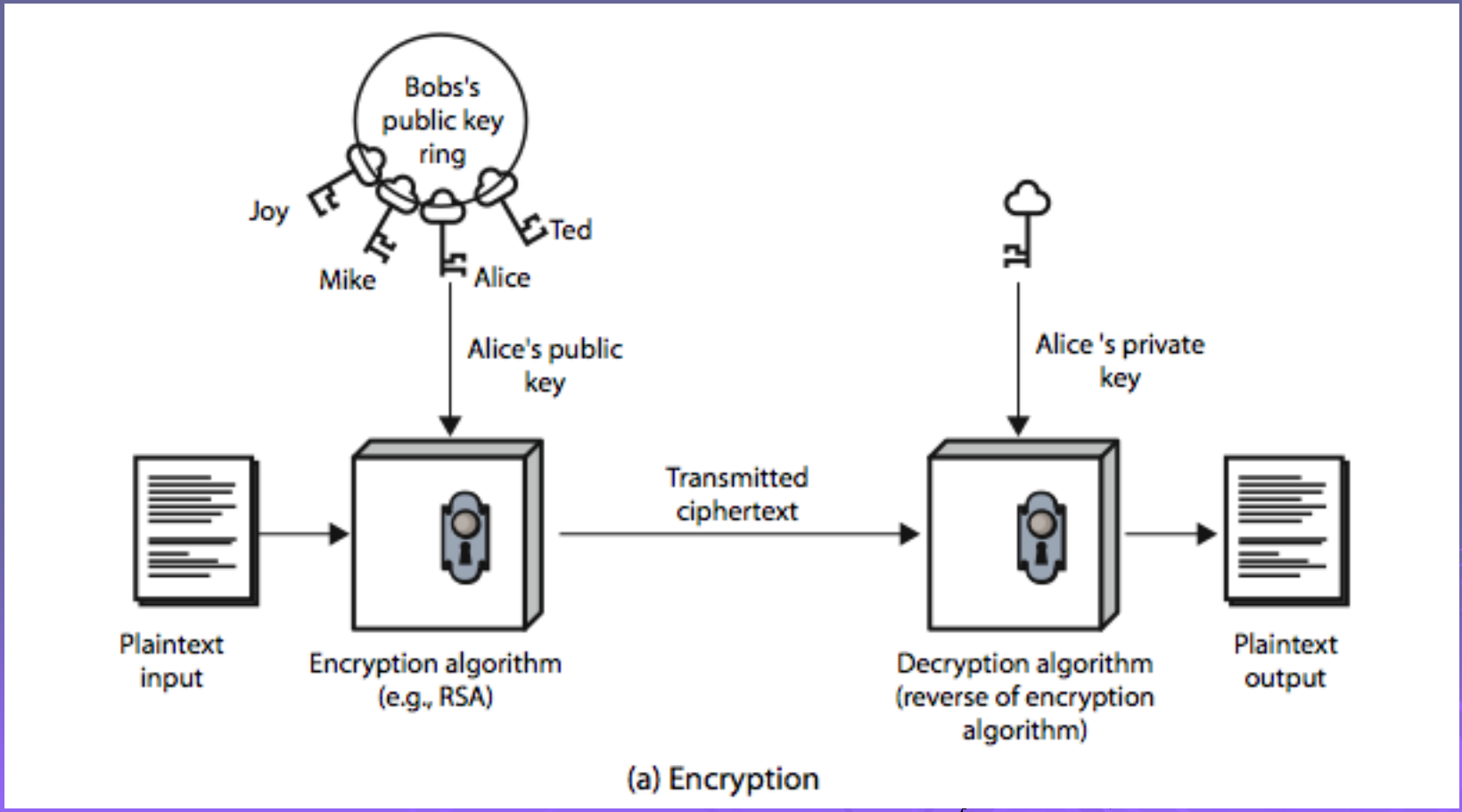
Symmetric vs Public-Key

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Public-Key Cryptosystem

- **A public-key encryption scheme has six ingredients:**
  - **Plain text**
  - **Encryption algorithm**
  - **Public and private key**
  - **Ciphertext**
  - **Decryption algorithm**

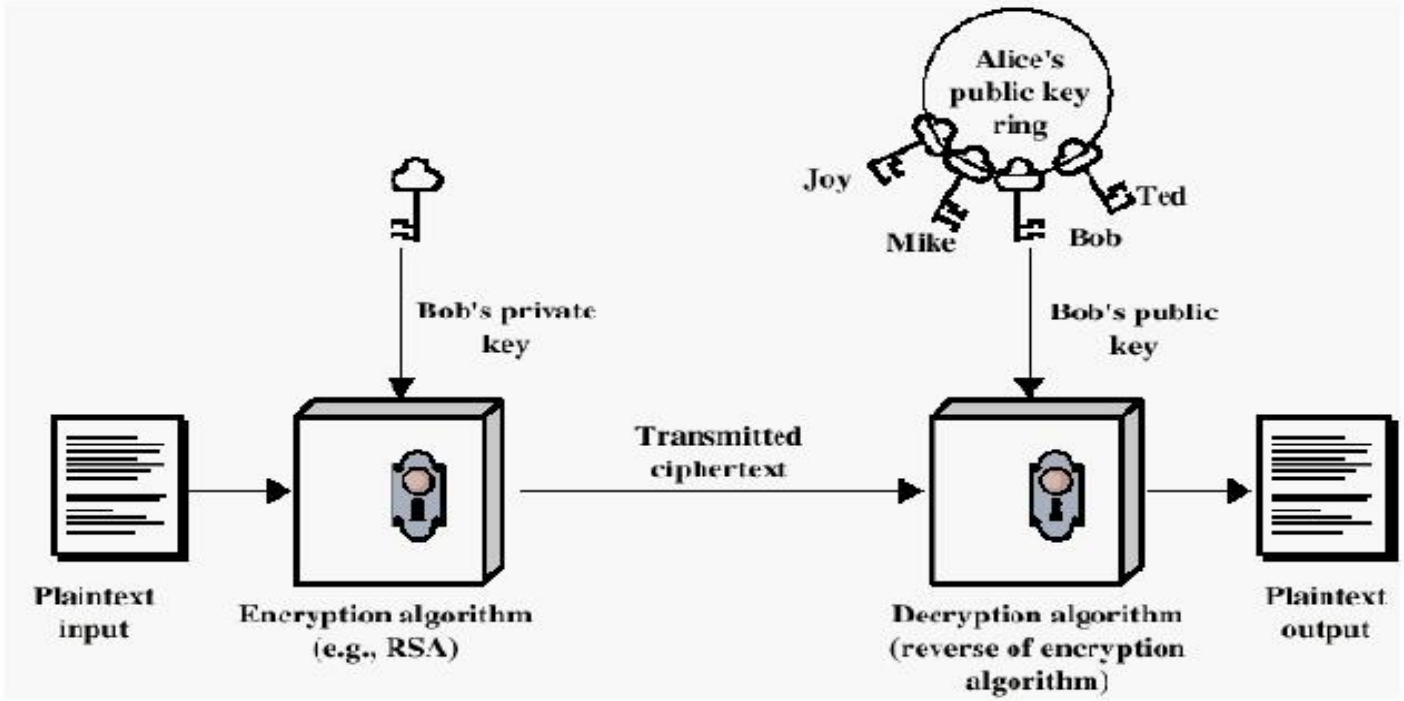
Encryption using Public-Key System



Public-Key Encryption

- **Public-key Encryption**
  1. Each end system generates a **pair** of keys.
  2. Each end system publishes its "**public**" **encryption key**.
  3. A sends B a message using **B's public key**.
  4. B decrypts the message using **B's private key.**

Authentication using Public-Key System



Public-Key Authentication

- **Public-key Authentication**

  1. A sends B a message encrypted with **A's private key**.

  2. B decrypts the message using **A's public key.**

  3. Entire message acts as a **digital signature**, since only A could have encrypted the message.

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 : Secrecy and Authentication

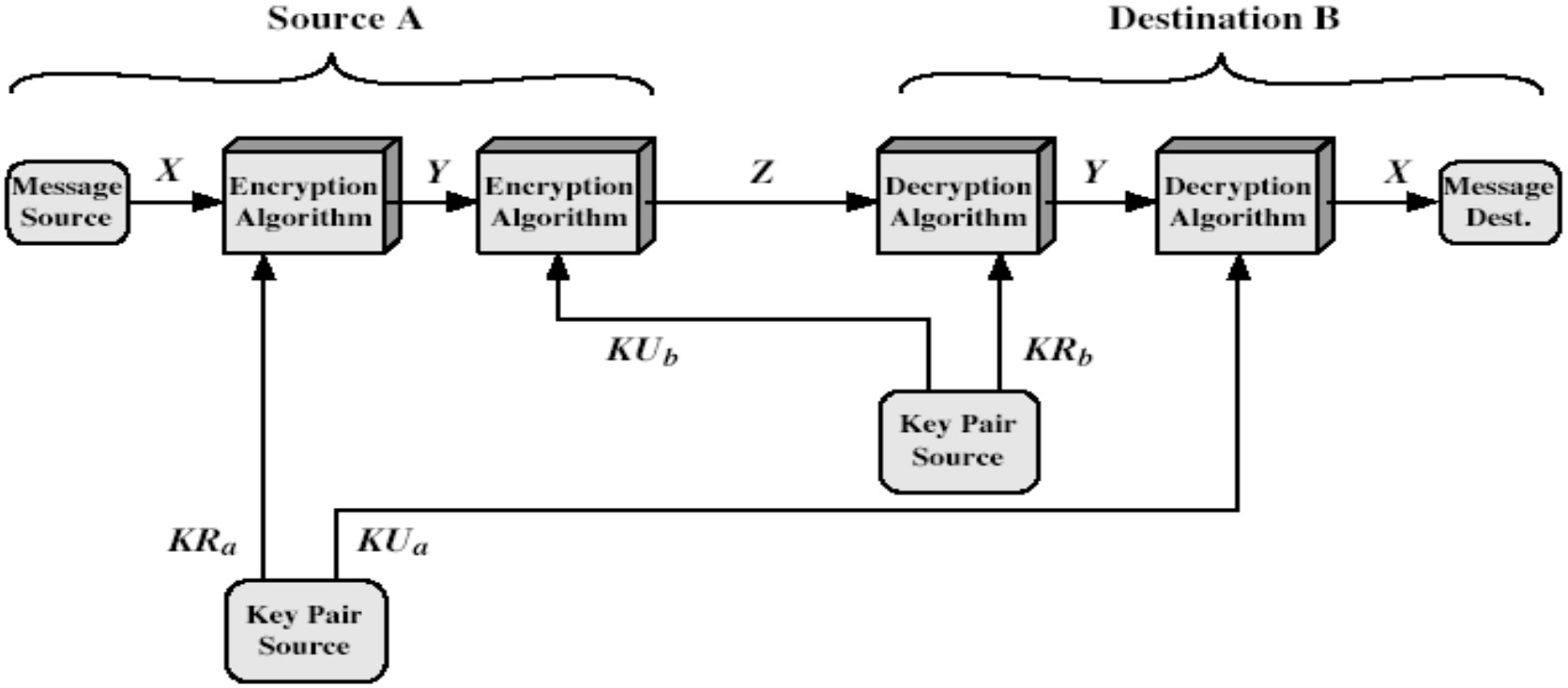


Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication

Digital Signatures

- Instead of the entire message, a small block can be used.
- This block, the **authenticator**, can then be encrypted using the sender's private key.
- This serves as a signature that verifies origin, content, and sequencing.
- SHA-1 could serve as the authenticator.

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 Alogarthim

- **RSA** (Rivest-Shamir-Adelman) is the most commonly used public key algorithm.
- Can be used both for encryption and for digitally signing.
- It is generally considered to be secure when sufficiently long keys are used (512 bits is insecure, 768 bits is moderately secure, and 1024 bits is good, for now).
- The security of RSA relies on the difficulty of factoring large integers. Dramatic advances in factoring large integers would make RSA vulnerable.
- RSA is currently the most important public key algorithm. It is patented in the United States (expires year 2000), and free elsewhere.

RSA Algorithm

- The **RSA algorithm** (1977) is widely used for public-key encryption.
- It is a block cipher, with plaintext and cipher text represented as integers from 0 to n-1.
- Based on exponentiation in a finite (Galois) field over integers modulo a prime
- Uses large integers (eg. 1024 bits)
- Security due to cost of factoring large numbers

Requirements for the RSA Algorithm

- It is **possible** to find values of e, d, and n such that $M^{ed} = M \bmod n$ for all integers $M < n$.
- It is **relatively easy** to calculate M^e and C^d for all values of $M < n$.
- It is **infeasible** to determine d given e and n.

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
  - $e.d=1 \bmod \varnothing(n)$ and $0 \leq d \leq n$
- Publish their public encryption key: PU={e,n}
- Keep secret private decryption key: PR={d,n}

RSA Algorithm Key Generation

- Select two different prime numbers, p and q.
- Calculate the product, n = p x q.
- Calculate the Euler totient.
  - $\phi(n) = (p - 1) \times (q - 1)$.
- Select integer e.
  - $\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$

RSA Algorithm Key Generation

- Calculate d.
  - de = mod ϕ (n) = 1.
- Public Key is KU = {e,n}.
- Private Key is KR= {d,n}.

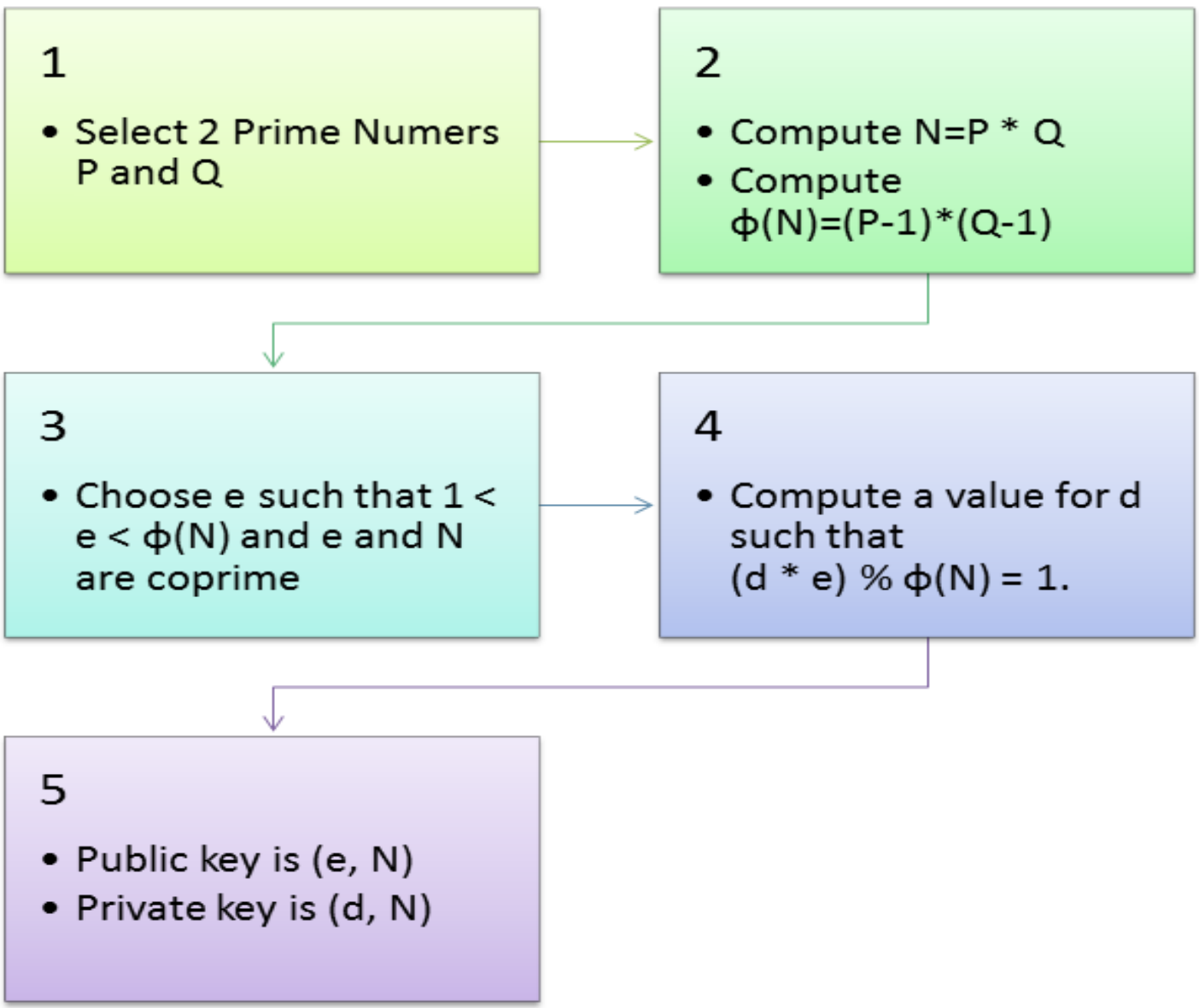
The RSA Algorithm

- **Encryption**
  - Public Key is KU = {e,n}.
  - $C = M^e \bmod n$, where M is the plaintext and C is the ciphertext (represented as integers).
- **Decryption**
  - Private Key is KR = {d,n}.
  - $M = C^d \bmod n$.

RSA Use

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

RSA Algorithm



Why RSA Works

- Because of Euler's Theorem:
  - $a^{\phi(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- In RSA have:
  - $n=p.q$
  - $\phi(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \phi(n)$
  - hence $e.d=1+k.\phi(n)$ for some k
- Hence :

$$C^d = M^{e.d} = M^{1+k.\phi(n)} = M^1.(M^{\phi(n)})^k$$
$$= M^1.(1)^k = M^1 = M \bmod n$$

RSA Example - Key Setup

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

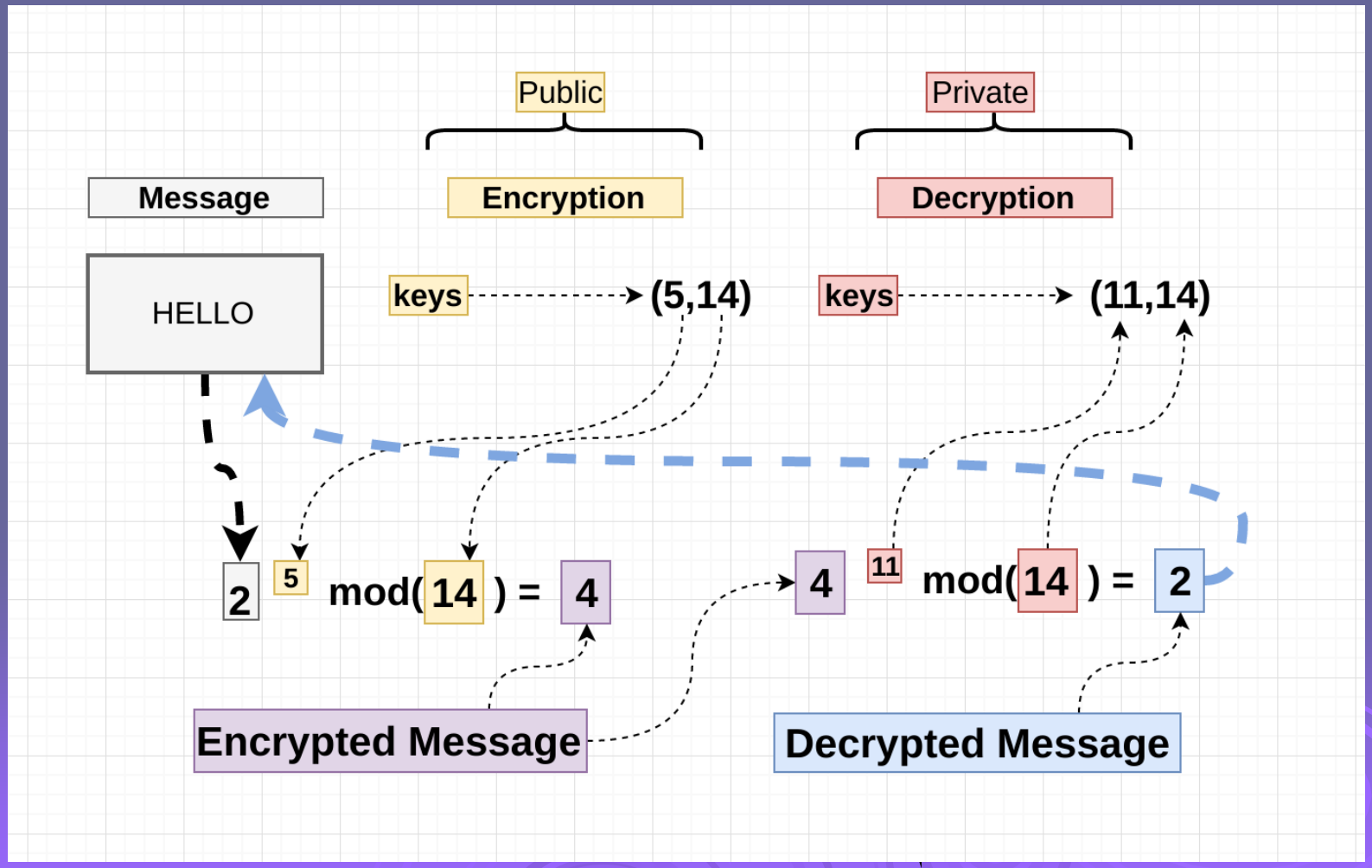
RSA Example-Encryption

- **Encryption Example**
  - Let M = 88.
  - $C = 88^7 \bmod 187$.
  - Now, consider the following property of modular arithmetic:
    - $X^{a+b} \bmod n = \{(X^a \bmod n)(X^b \bmod n)\} \bmod n$.
    - $C = \{(88^4 \bmod 187)(88^2 \bmod 187)(88^1 \bmod 187)\} \bmod 187$
    - $C = \{132 \times 77 \times 88\} \bmod 187$
    - $C = 11$.

RSA Example-Decryption

- **Decryption Example**
  - Let C = 11.
  - $M = 11^{23} \bmod 187$.
  - $M=\{(11^{1} \bmod 187)(11^{2} \bmod 187)(11^{4} \bmod 187)(11^{8} \bmod 187)\ (11^{8} \bmod 187)\} \bmod 187$
    - M={11 x 121 x 55 x 33x 33} mod 187
    - M = 88.

How does RSA works: Example



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_i == 1 then
       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 $\gcd(e, \phi(n)) = 1$
  - 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

Attacks on RSA

- **Brute force**--try all possible keys.
  - This means large keys need to be used, but implementations will have longer computation time.
- **Factor n**, into its prime factors (p and q.)
  - For n large, this is a hard problem.

RSA Security

- Possible approaches to attacking RSA are:
  - brute force key search (infeasible given size of numbers)
  - mathematical attacks (based on difficulty of computing ø(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 $\varnothing(n)$ and then d
  - determine $\varnothing(n)$ 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