Public-Key Cryptography

• **What we already know**
  – symmetric key cryptography enables **confidentiality**
    • achieved through secret key encryption
  – symmetric key cryptography enables **authentication and integrity**
    • achieved through MACs

• In all of the above the sender and received must share a secret key
  – need a secure channel for key distribution
  – not possible for parties with no prior relationship
  – public-key cryptography can aid with this
Public-Key Cryptography

- Other limitations of symmetric key cryptography
  - authentication to multiple receivers is difficult
  - non-repudiation cannot be achieved

- What’s the solution?
  - the concept of more powerful asymmetric key encryption

- Public-key cryptography was proposed by Diffie and Hellman
  - it was in 1976 in their work “New directions in cryptography”
• **Diffie and Hellman** introduced
  – public-key encryption
  – public-key key agreement protocols
  – digital signatures

• It also turned out that public-key encryption was proposed earlier
  – James Ellis proposed it in 1970 in a classified paper
  – the paper was made public by the British government in 1997

• The concept of key agreement and digital signatures is still due to Diffie and Hellman
Public-Key Cryptography

• Public-key encryption
  – a party creates a public-private key pair
    • the public key is $pk$
    • the private or secret key is $sk$
  – the public key is used for encryption $Enc_{pk}(m)$ and is publicly available
  – the private key is used for decryption only $Dec_{sk}(c)$
  – knowing the public key and the encryption algorithm only, it is computationally infeasible to find the secret key
• (Public-key) **Key agreement or key distribution**
  – prior to the protocol the parties do not share a common secret
  – after the protocol execution they hold a key not known to any eavesdropper

• **Digital signatures**
  – a party generates a public-private signing key pair
  – private key is used to sign a message
  – public key is used to verify a signature on a message
  – can be viewed as single-source message authentication
A public-key encryption scheme consists of three PPT algorithms (Gen, Enc, Dec) such that:

1. **key generation** Gen, on input security parameter $1^n$, outputs a public-private key pair $(pk, sk)$

2. **encryption** Enc, on input public key $pk$ and messages $m$ from the message space, outputs ciphertext $c \leftarrow \text{Enc}_{pk}(m)$
   - message space often depends on $pk$

3. **decryption** Dec, on input private key $sk$ and ciphertext $c$, outputs a message $m := \text{Dec}_{sk}(c)$ or a special failure symbol ⊥.
Public Key Encryption

- **Message space** $\mathcal{M}$ can now be different from, e.g., all strings of size $n$
  - if we use arithmetic modulo $p$, a message can be any number in
    \[ \{0, \ldots, p - 1\} \]

- **Properties**
  - correctness
    - as before, we want $\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m$
    - but we can permit a negligible probability of failure
  - security
    - what is different from our previous definitions?
• We are given public-key encryption scheme $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec})$

• The eavesdropping indistinguishability experiment $\text{PubK}_{\mathcal{A},\mathcal{E}}^{\text{eav}}(n)$
  
  1. $\text{Gen}(1^n)$ is run to produce keys $(pk, sk)$
  
  2. adversary $\mathcal{A}$ is given $pk$ and outputs two messages $m_0, m_1$ from message space
  
  3. random bit $b \leftarrow \{0, 1\}$ is chosen, and ciphertext $c \leftarrow \text{Enc}_{pk}(m_b)$ is given to $\mathcal{A}$
  
  4. $\mathcal{A}$ outputs bit $b'$; if $b = b'$, the experiment outputs 1 ($\mathcal{A}$ wins), and 0 otherwise
The CPA indistinguishability experiment $\text{PubK}_{\mathcal{A}, \mathcal{E}}^\text{cpa}(n)$

1. $\text{Gen}(1^n)$ is run to produce keys $(pk, sk)$

2. adversary $\mathcal{A}$ is given $pk$ and oracle access to $\text{Enc}_{pk}(\cdot)$; it outputs two messages $m_0, m_1$ from message space

3. random bit $b \leftarrow \{0, 1\}$ is chosen, and ciphertext $c \leftarrow \text{Enc}_{pk}(m_b)$ is given to $\mathcal{A}$

4. $\mathcal{A}$ continues to have oracle access to $\text{Enc}_{pk}(\cdot)$ and outputs bit $b'$

5. if $b = b'$, the experiment outputs 1 ($\mathcal{A}$ wins), and 0 otherwise
Notions of Security

• A public-key encryption scheme $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec})$ has indistinguishable encryptions under a chosen-plaintext attack (or is CPA-secure) if for all PPT adversaries $A$,

$$\Pr[\text{PubK}_{A,\mathcal{E}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

i.e., $A$ cannot win the game with significantly better chances than random guess

• Similar definition can be constructed for eavesdropping adversaries

• What is the gap between the two notions of security?
Notions of Security

- We obtain that no deterministic public-key encryption scheme has indistinguishable encryptions in the presence of eavesdropper and under CPA attack.
- Does anything change if we deal with multiple messages?
- What can we say about encrypting long messages?
- How about perfect secrecy in the public-key setting?
In practice, to encrypt long messages **hybrid encryption** is used

- the simplest way is to choose a random symmetric key $k$ and send it encrypted with the recipient’s public key $\text{Enc}_{pk}(k)$
- encrypt the message $m$ itself using $k$ and symmetric key encryption $\mathcal{E}' = (\text{Gen}', \text{Enc}', \text{Dec}')$
  - $m$ might need to be partitioned as $m_1, \ldots, m_t$
  - send $\text{Enc}'_k(m_1), \ldots, \text{Enc}'_k(m_t)$

- Why do we use a combination of two different encryption algorithms?
RSA Cryptosystem

- The RSA algorithm
  - invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1978
  - its security requires that factoring large numbers is hard
  - but there is no proof that the algorithm is as hard to break as factoring
  - sustained many years of attacks on it
Background

- Recall Euler’s \( \phi \) function
  - for a product of two primes \( n = pq \), \( \phi(n) = (p - 1)(q - 1) \)

- Euler’s theorem
  - given \( m > 1 \) and \( a \) with \( \gcd(a, m) = 1 \), \( a^{\phi(m)} \equiv 1 \pmod{m} \)

- Recall Euler’s theorem’s corollary
  - given \( x, y, m, \) and \( a \) with \( \gcd(m, a) = 1 \), if \( x \equiv y \pmod{\phi(m)} \),
    then \( a^x \equiv a^y \pmod{m} \)

- Computation of a multiplicative inverse modulo \( m \)
  - given \( a \) and \( m \) with \( \gcd(a, m) = 1 \), there is a unique \( x \) (between 0 and \( m \)) such that \( ax \equiv 1 \pmod{m} \)
The idea

- for modulus $n > 1$ and integer $e > 0$, let $x \in \mathbb{Z}_n^*$
- then $f(x) = x^e \mod n$ is a permutation if $\gcd(e, n) = 1$
- if $d = e^{-1} \mod \phi(n)$, $f'(x) = x^d \mod n$ is the inverse of $f$.

The hardness assumption is called the RSA problem and is to compute the inverse function

- easy if factorization of $n$ or $\phi(n)$ is known
- believed to be hard otherwise
Plain or “Textbook” RSA

• Key generation
  – given security parameter $1^k$, generate two large prime numbers $p$ and $q$, each $k/2$ bits long
  – compute $n = pq$
  – select a small prime number $e$
  – compute $\phi(n) = (p - 1)(q - 1)$
  – and then compute $d$ – the inverse of $e$ modulo $\phi(n)$
    • i.e., $ed \equiv 1 \pmod{\phi(n)}$

• The public key is $pk = (e, n)$
The private key is $sk = d$
Plain RSA

- **Encryption**
  - given a message \( m \in \mathbb{Z}_n^* \)
  - given a public key \( pk = (e, n) \)
  - encrypt as \( c = \text{Enc}_{pk}(m) = m^e \mod n \)

- **Decryption**
  - given a ciphertext \( c \)
  - given a public key \( pk = (e, n) \) and the corresponding private key \( sk = d \)
  - decrypt as \( m = \text{Dec}_{sk}(c) = c^d \mod n \)
Example

- generate a key pair
  - pick $p = 7$, $q = 11$
  - compute $n = 77$
  - pick $e = 37$
  - compute $\phi(n) = 6 \cdot 10 = 60$
  - compute $d \equiv e^{-1} \equiv 13 \pmod{60}$
- public key $(37, 77)$
- private key 13
• **Example** (cont.)
  
  – encryption
    
    • given a message \( m = 15 \)
    
    • encryption is \( c = m^e \mod n \)
    
    • \( c = 15^{37} \mod 77 = 71 \)
  
  – decryption
    
    • given ciphertext \( c = 71 \)
    
    • decryption is \( m = c^d \mod n \)
    
    • \( m = 71^{13} \mod 77 = 15 \)
• **Why does it work?**
  
  – we would like to see how the message is recovered from the ciphertext

• **Decrypting encrypted message**
  
  – $\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m$

  – recall that $ed \equiv 1 \mod \phi(n)$
  
  – also recall that $x \equiv y \mod \phi(n) \Rightarrow m^x \equiv m^y \mod n$
  
  – thus, we obtain $m^{ed} \equiv m$
More on RSA

• All of the above works when a message \( m \in \mathbb{Z}_n^* \)
  – the algorithm doesn’t go through if \( \gcd(m, n) \neq 1 \)
  – the problem is that the space \( \mathbb{Z}_n^* \) is not known without private key

• The good news is that we can still use any \( m \) between 0 and \( n - 1 \)
  – for \( n = pq \), the probability that \( \gcd(m, n) \neq 1 \) is negligible
  – and if \( \gcd(m, n) \neq 1 \), there are bigger problems than algorithm’s failure
RSA Security

• Security of RSA requires that the RSA problem is hard

• We start with factoring which must also be hard
  – let algorithm GenMod on input $1^k$ output $n = pq$, where $p$ and $q$ are $k/2$-bit primes

• The factoring experiment $\text{Factor}_{A,\text{GenMod}}(k)$
  1. run $\text{GenMod}(1^k)$ and obtain $(p, q, n)$
  2. $A$ is given $n$ and outputs $p', q' > 1$
  3. output 1 ($A$ wins) if $p' \cdot q' = n$, and 0 otherwise

• Factoring is hard (relative to GenMod) if for all PPT algorithms $A$

$$\Pr[\text{Factor}_{A,\text{GenMod}}(k) = 1] \leq \text{negl}(k)$$
Let $\text{GenRSA}$ be the key generation algorithm for RSA that takes $1^k$ and outputs $(n, e, d)$

The RSA experiment $\text{RSAInv}_{A, \text{GenRSA}}(k)$

1. run $\text{GenRSA}(1^k)$ to obtain $(n, e, d)$
2. choose $y \in \mathbb{Z}_n^*$ and give $n, e,$ and $y$ to $A$
3. $A$ outputs $x \in \mathbb{Z}_n^*$ and wins (the experiment outputs 1) iff $y = x^e \mod n$

The RSA problem is hard (relative to $\text{GenRSA}$) if any PPT algorithm $A$ wins the RSA experiment with at most negligible probability

$$\Pr[\text{RSAInv}_{A, \text{GenRSA}}(k) = 1] \leq \text{negl}(k)$$
Insecurity of Plain RSA

- Hardness of RSA problem implies that it can generally be hard to decrypt messages without the private key (or factorization of the modulus)

- The above description of RSA, however, is not secure
  - why?

- What does the above construction exactly guarantee?
  - given a message $m$ chosen uniformly at random from $\mathbb{Z}_n^*$ and the public key $(n, e)$
  - adversary cannot recover the entire $m$
• **Choosing** \( p, q, \) **and** \( n \)

  – today the modulus \( n \) needs to be at least 1536 bits long
  – often a random number is chosen for \( p \) and \( q \) and is tested for primality
  – **Miller-Rabin** primality test is common
    • the algorithm has a probability of error
    • but it is popular due to its speed
    • how large the error is can be controlled
    • composite numbers that pass this primality test are called strong pseudo-prime numbers
• Choosing $e$
  
  – the smaller $e$ is, the faster encryption is performed
  
  – recall that the square-and-multiply algorithm for computing $m^e \mod n$ depends on the length of the exponent
    
    • the number of multiplications also directly depends on the number of 1’s in the binary representation of $e$
  
  – common choices for $e$ are 3, 17, $2^{16} + 1 = 65537$
    
    • such numbers require only a few modulo multiplications to encrypt
• **Speeding up decryption**
  
  – we don’t have control over $d$ – it’ll have to be long
  
  – but we can still decrypt faster using smaller moduli
  
  – since $p$ and $q$ are known, we can exploit their shorter size
  
  – we apply the **Chinese Remainder Theorem**
    
    • recall that the CRT solves a system of congruences
      
      $x_i \equiv a_i \pmod{n_i}$
      
    • the solution is a congruence modulo $n = \prod n_i$
• Using the CRT for decryption
  - we have $c$ and the goal is to compute $m = c^d \mod n$
  - we first compute $m_1 = c^d \mod p$ and $m_2 = c^d \mod q$
  - this gives us $m_1 = m \mod p$ and $m_2 = m \mod q$
  - we then combine $m_1$ and $m_2$ using the CRT to obtain $m \mod n$
    - the equations we are solving are $m \equiv m_1 \pmod{p}$ and $m \equiv m_2 \pmod{q}$
    - the unique solution is
      $$m \equiv m_1(q^{-1} \mod p)q + m_2(p^{-1} \mod q)p \pmod{n}$$
Public key cryptography achieves many objectives

Security of public key encryption can be modeled similar to symmetric encryption

- but security against chosen-plaintext attack (CPA) is now the weakest reasonable security model

RSA is the most commonly used public-key encryption algorithm

- requires that factoring large numbers is hard
- the plain or “textbook” RSA doesn’t meet our definition of security

RSA implementations target at faster performance