Chapter 9 Public Key Cryptography and RSA


9.1 Principles of Public Key Cryptosystems

- 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.
9.1 Principles of Public Key cryptosystems

- 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.

---

- 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.
9.1 Principles of Public Key cryptosystems

Public-Key Cryptography: Encryption

(notation and explanation)

Public-Key Cryptography: Authentication

(notation and explanation)
9.1 Principles of Public Key cryptosystems

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

---

Table 9.1 CONVENTIONAL AND PUBLIC-KEY ENCRYPTION

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

- Public-Key Characteristics
  - Public-Key algorithms rely on two keys with the characteristics that it is:
    - computationally infeasible to find decryption key knowing only algorithm & encryption key
    - 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 (in some schemes)

9.1 Principles of Public Key cryptosystems

- Public-Key Cryptosystems: Secrecy

![Diagram showing encryption and decryption process with key pairs KU_b and K_R_b]
9.1 Principles of Public Key cryptosystems

❖ Public-Key Cryptosystems: Authentication

Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication
9.1 Principles of 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Encryption/Decryption</th>
<th>Digital Signature</th>
<th>Key Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Elliptic Curve</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Diffie-Hellman</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DSS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

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, its just made too hard to do in practise
- requires the use of very large numbers
- hence is slow compared to private key schemes.
9.2 The RSA Algorithm

- 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 (e.g. 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 $N=p \times q$
  - note $\phi(N)=(p-1)\times(q-1)$
- selecting at random the encryption key $e$
  - where $1<e<\phi(N)$, $\gcd(e,\phi(N))=1$
- solve following equation to find decryption key $d$
  - $e \times d \equiv 1 \pmod{\phi(N)}$ and $0 \leq d < N$
- publish their public encryption key: $K_U=\{e,N\}$
- keep secret private decryption key: $K_R=\{d,p,q\}$.
9.2 The RSA Algorithm

- **RSA Use**
  - to encrypt a message $M$ the sender:
    - obtains **public key** of recipient $K_U = \{e, N\}$
    - computes: $C = M^e \mod N$, where $0 \leq M < N$
  - to decrypt the ciphertext $C$ the owner:
    - uses their private key $K_R = \{d, p, q\}$
    - computes: $M = C^d \mod N$
  - note that the message $M$ must be smaller than the modulus $N$ (block if needed).

---

### Brief Summary of RSA

#### Key Generation
- Select $p, q$
- Calculate $n = pq$
- Calculate $\phi(n) = (p-1)(q-1)$
- Select integer $e$, $\gcd(e, \phi(n)) = 1$, $1 < e < \phi(n)$
- Calculate $d = e^{-1} \mod \phi(n)$
- Public key $K_U = \{e, n\}$
- Private key $K_R = \{d, n\}$

#### Encryption
- Plaintext: $M < n$
- Ciphertext: $C = M^e \mod n$

#### Decryption
- Ciphertext: $C$
- Plaintext: $M = C^d \mod n$
9.2 The RSA Algorithm

Why RSA Works
- because of Euler’s Theorem:
  \[ a^{\phi(N)} \mod N = 1 \]
  where \( \gcd(a, N) = 1 \)
- in RSA have:
  \[ N = p \cdot q \]
  \[ \phi(N) = (p-1)(q-1) \]
  carefully chosen \( e \) & \( d \) to be inverses mod \( \phi(N) \)
  hence \( e \cdot d = 1 + k \cdot \phi(N) \) for some \( k \)

hence:
\[
E^d = (M^e)^d = M^{e \cdot d} = M^{1 + k \cdot \phi(N)} = M^{(1)(1)q}
= M^1 = M \mod N.
\]

RSA Example
1. Select primes: \( p = 17 \) & \( q = 11 \)
2. Compute \( n = p \cdot q = 17 \times 11 = 187 \)
3. Compute \( \phi(n) = (p-1)(q-1) = 16 \times 10 = 160 \)
4. Select \( e : \gcd(e, 160) = 1 \); choose \( e = 7 \)
5. Determine \( d : d \cdot e = 1 \mod 160 \) and \( d < 160 \)
   Value is \( d = 23 \) since \( 23 \times 7 = 161 = 10 \times 160 + 1 \)
6. Publish public key \( KU = \{7, 187\} \)
7. Keep secret private key \( KR = \{23, 17, 11\} \)
9.2 The RSA Algorithm

- sample RSA encryption/decryption is:
  - given message \( M = 88 \) (nb. \( 88 < 187 \))
  - encryption:
    \[ C = 88^7 \mod 187 = 11 \]
  - decryption:
    \[ M = 11^{23} \mod 187 = 88. \]

9.2 The RSA Algorithm

- Computational Aspects: Encryption and Decryption
  - **Square and Multiply Algorithm** for Exponentiation
    - 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. \( 7^5 = 7^4.7^1 = 3.7 = 10 \mod 11 \)
      - eg. \( 3^{129} = 3^{128}.3^1 = 5.3 = 4 \mod 11 \)
9.2 The RSA Algorithm

- Example: $a^b \mod n$, where $a = 7$, $b = 560 = 1000110000$, $n = 561$.

$1000110000 = 1 \times 2^9 + 0 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$.

$a^{1000110000} \mod n$

$= (a^2 \mod n) \times (a^5 \mod n) \times (a^4 \mod n) \mod n$.

- Square: $(a^2 \mod n) \rightarrow (a^4 \mod n) \rightarrow (a^8 \mod n) \rightarrow (a^{16} \mod n) \rightarrow (a^{32} \mod n) \rightarrow (a^{64} \mod n) \rightarrow (a^{128} \mod n) \rightarrow (a^{256} \mod n) \rightarrow (a^{512} \mod n) \rightarrow (a^{1024} \mod n)$.

- Multiplication: $(a^3 \mod n) \times (a^5 \mod n) \times (a^4 \mod n) \mod n$.

9.2 Square and RSA Algorithm

- Square and Multiply Algorithm

```
  c ← 0; d ← 1
  for i ← k downto 0
      do c ← 2 \times c
          d ← (d \times d) \mod n
          if b_i = 1
              then c ← c + 1
                  d ← (d \times a) \mod n
  return d
```

Figure 9.8 Result of the Fast Modular Exponentiation Algorithm for $a^b \mod n$, where $a = 7$, $b = 560 = 1000110000$, $n = 561$.
9.2 The RSA Algorithm

- Computational Aspects: 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 \( N = p \cdot q \)
    - means must be sufficiently large
    - typically guess and use probabilistic test
  - exponents \( e, d \) are inverses, so use Inverse algorithm to compute the other.

- The Security of RSA
  - three approaches to attacking RSA:
    - brute force key search (infeasible given size of numbers)
    - mathematical attacks (based on difficulty of computing \( \varphi(N) \), by factoring modulus \( N \))
    - timing attacks (on running of decryption)
9.2 The RSA Algorithm

- Factoring Problem
  - mathematical approach takes 3 forms:
    - factor $N=p \cdot q$, hence find $\varphi(N)$ and then $d$
    - determine $\varphi(N)$ directly and find $d$
    - find $d$ directly
  - currently believe all equivalent to factoring
    - have seen slow improvements over the years
      - as of Aug-99 best is 130 decimal digits (512) bit with GNFS
    - biggest improvement comes from improved algorithm
      - cf “Quadratic Sieve” to “Generalized Number Field Sieve”
    - barring dramatic breakthrough 1024+ bit RSA secure
      - ensure $p, q$ of similar size and matching other constraints

---

9.2 The RSA Algorithm

- Progress in Factorization [Table 9.3]

<table>
<thead>
<tr>
<th>Number of Decimal Digits</th>
<th>Approximate Number of Bits</th>
<th>Date Achieved</th>
<th>MFS-years</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>232</td>
<td>April 1991</td>
<td>7</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>110</td>
<td>365</td>
<td>April 1992</td>
<td>75</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>120</td>
<td>396</td>
<td>June 1993</td>
<td>850</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>129</td>
<td>428</td>
<td>April 1994</td>
<td>5000</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>130</td>
<td>431</td>
<td>April 1996</td>
<td>1000</td>
<td>generalized number field sieve</td>
</tr>
<tr>
<td>140</td>
<td>465</td>
<td>February 1999</td>
<td>2000</td>
<td>generalized number field sieve</td>
</tr>
<tr>
<td>155</td>
<td>512</td>
<td>August 1999</td>
<td>8000</td>
<td>generalized number field sieve</td>
</tr>
</tbody>
</table>
9.2 The RSA Algorithm

- Timing Attacks
  - developed in mid-1990’s
  - exploit timing variations in operations
    - e.g. 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.

Summary

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