1.2 Public-key cryptography

In a public-key cryptographic scheme, a key pair is selected so that the problem of deriving the private key from the corresponding public key is equivalent to solving a computational problem that is believed to be intractable. Number-theoretic problems whose intractability form the basis for the security of commonly used public-key schemes are:

1. The integer factorization problem, whose hardness is essential for the security of RSA public-key encryption and signature schemes.
2. The discrete logarithm problem, whose hardness is essential for the security of the ElGamal public-key encryption and signature schemes and their variants such as the Digital Signature Algorithm (DSA).
3. The elliptic curve discrete logarithm problem, whose hardness is essential for the security of all elliptic curve cryptographic schemes.

In this section, we review the basic RSA, ElGamal, and elliptic curve public-key encryption and signature schemes. We emphasize that the schemes presented in this section are the basic "textbook" versions, and enhancements to the schemes are required (such as padding plaintext messages with random strings prior to encryption) before they can be considered to offer adequate protection against real attacks. Nevertheless, the basic schemes illustrate the main ideas behind the RSA, discrete logarithm, and elliptic curve families of public-key algorithms. Enhanced versions of the basic elliptic curve schemes are presented in Chapter 4.

1.2.1 RSA systems

RSA, named after its inventors Rivest, Shamir and Adleman, was proposed in 1977 shortly after the discovery of public-key cryptography.

RSA key generation

An RSA key pair can be generated using Algorithm 1.1. The public key consists of a pair of integers (n, e) where the *RSA modulus* n is a product of two randomly generated (and secret) primes p and q of the same bitlength. The *encryption exponent* e is an integer satisfying $1 < e < \phi$ and $\gcd(e, \phi) = 1$ where $\phi = (p-1)(q-1)$. The private key d, also called the *decryption exponent*, is the integer satisfying $1 < d < \phi$ and $ed \equiv 1 \pmod{\phi}$. It has been proven that the problem of determining the private key d from the public key (n, e) is computationally equivalent to the problem of determining the factors p and q of n; the latter is the *integer factorization problem* (IFP).

Algorithm 1.1 RSA key pair generation

INPUT: Security parameter l.
OUTPUT: RSA public key (n, e) and private key d.

1. Randomly select two primes p and q of the same bitlength $l/2$.
2. Compute $n = pq$ and $\phi = (p-1)(q-1)$.
3. Select an arbitrary integer e with $1 < e < \phi$ and $\gcd(e, \phi) = 1$.
4. Compute the integer d satisfying $1 < d < \phi$ and $ed \equiv 1 \pmod{\phi}$.
5. Return(n, e, d).

RSA encryption scheme

RSA encryption and signature schemes use the fact that

$$m^{ed} \equiv m \pmod{n} \tag{1.1}$$

for all integers m. The encryption and decryption procedures for the (basic) RSA public-key encryption scheme are presented as Algorithms 1.2 and 1.3. Decryption works because $c^d \equiv (m^e)^d \equiv m \pmod{n}$, as derived from expression (1.1). The security relies on the difficulty of computing the plaintext m from the ciphertext $c = m^e \bmod n$ and the public parameters n and e. This is the problem of finding eth roots modulo n and is assumed (but has not been proven) to be as difficult as the integer factorization problem.

Algorithm 1.2 Basic RSA encryption

INPUT: RSA public key (n, e), plaintext $m \in [0, n-1]$.
OUTPUT: Ciphertext c.

1. Compute $c = m^e \bmod n$.
2. Return(c).

Algorithm 1.3 Basic RSA decryption

INPUT: RSA public key (n, e), RSA private key d, ciphertext c.
OUTPUT: Plaintext m.

1. Compute $m = c^d \bmod n$.
2. Return(m).

RSA signature scheme

The RSA signing and verifying procedures are shown in Algorithms 1.4 and 1.5. The signer of a message m first computes its message digest $h = H(m)$ using a cryptographic hash function H, where h serves as a short fingerprint of m. Then, the signer

uses his private key d to compute the eth root s of h modulo n: $s = h^d \bmod n$. Note that $s^e \equiv h \pmod{n}$ from expression (1.1). The signer transmits the message m and its signature s to a verifying party. This party then recomputes the message digest $h = H(m)$, recovers a message digest $h' = s^e \bmod n$ from s, and accepts the signature as being valid for m provided that $h = h'$. The security relies on the inability of a forger (who does not know the private key d) to compute eth roots modulo n.

Algorithm 1.4 Basic RSA signature generation

INPUT: RSA public key (n, e), RSA private key d, message m.
OUTPUT: Signature s.

1. Compute $h = H(m)$ where H is a hash function.
2. Compute $s = h^d \bmod n$.
3. Return(s).

Algorithm 1.5 Basic RSA signature verification

INPUT: RSA public key (n, e), message m, signature s.
OUTPUT: Acceptance or rejection of the signature.

1. Compute $h = H(m)$.
2. Compute $h' = s^e \bmod n$.
3. If $h = h'$ then return("Accept the signature");
   Else return("Reject the signature").

The computationally expensive step in any RSA operation is the modular exponentiation, e.g., computing $m^e \bmod n$ in encryption and $c^d \bmod n$ in decryption. In order to increase the efficiency of encryption and signature verification, one can select a small encryption exponent e; in practice, $e = 3$ or $e = 2^{16} + 1$ is commonly chosen. The decryption exponent d is of the same bitlength as n. Thus, RSA encryption and signature verification with small exponent e are significantly faster than RSA decryption and signature generation.

1.2.2 Discrete logarithm systems

The first discrete logarithm (DL) system was the key agreement protocol proposed by Diffie and Hellman in 1976. In 1984, ElGamal described DL public-key encryption and signature schemes. Since then, many variants of these schemes have been proposed. Here we present the basic ElGamal public-key encryption scheme and the Digital Signature Algorithm (DSA).

DL key generation

In discrete logarithm systems, a key pair is associated with a set of public domain parameters (p, q, g). Here, p is a prime, q is a prime divisor of $p-1$, and $g \in [1, p-1]$ has order q (i.e., $t = q$ is the smallest positive integer satisfying $g^t \equiv 1 \pmod{p}$). A private key is an integer x that is selected uniformly at random from the interval $[1, q-1]$ (this operation is denoted $x \in_R [1, q-1]$), and the corresponding public key is $y = g^x \bmod p$. The problem of determining x given domain parameters (p, q, g) and y is the *discrete logarithm problem* (DLP). We summarize the DL domain parameter generation and key pair generation procedures in Algorithms 1.6 and 1.7, respectively.

Algorithm 1.6 DL domain parameter generation

INPUT: Security parameters l, t.
OUTPUT: DL domain parameters (p, q, g).

1. Select a t-bit prime q and an l-bit prime p such that q divides $p-1$.
2. Select an element g of order q:
   2.1 Select arbitrary $h \in [1, p-1]$ and compute $g = h^{(p-1)/q} \bmod p$.
   2.2 If $g = 1$ then go to step 2.1.
3. Return(p, q, g).

Algorithm 1.7 DL key pair generation

INPUT: DL domain parameters (p, q, g).
OUTPUT: Public key y and private key x.

1. Select $x \in_R [1, q-1]$.
2. Compute $y = g^x \bmod p$.
3. Return(y, x).

DL encryption scheme

We present the encryption and decryption procedures for the (basic) ElGamal public-key encryption scheme as Algorithms 1.8 and 1.9, respectively. If y is the intended recipient's public key, then a plaintext m is encrypted by multiplying it by $y^k \bmod p$ where k is randomly selected by the sender. The sender transmits this product $c_2 = my^k \bmod p$ and also $c_1 = g^k \bmod p$ to the recipient who uses her private key to compute

$$c_1^x \equiv g^{kx} \equiv y^k \pmod{p}$$

and divides c_2 by this quantity to recover m. An eavesdropper who wishes to recover m needs to calculate $y^k \bmod p$. This task of computing $y^k \bmod p$ from the domain parameters (p, q, g), y, and $c_1 = g^k \bmod p$ is called the *Diffie-Hellman problem* (DHP).

The DHP is assumed (and has been proven in some cases) to be as difficult as the discrete logarithm problem.

Algorithm 1.8 Basic ElGamal encryption

INPUT: DL domain parameters (p, q, g), public key y, plaintext $m \in [0, p-1]$.
OUTPUT: Ciphertext (c_1, c_2).

1. Select $k \in_R [1, q-1]$.
2. Compute $c_1 = g^k \bmod p$.
3. Compute $c_2 = m \cdot y^k \bmod p$.
4. Return(c_1, c_2).

Algorithm 1.9 Basic ElGamal decryption

INPUT: DL domain parameters (p, q, g), private key x, ciphertext (c_1, c_2).
OUTPUT: Plaintext m.

1. Compute $m = c_2 \cdot c_1^{-x} \bmod p$.
2. Return(m).

DL signature scheme

The Digital Signature Algorithm (DSA) was proposed in 1991 by the U.S. National Institute of Standards and Technology (NIST) and was specified in a U.S. Government Federal Information Processing Standard (FIPS 186) called the Digital Signature Standard (DSS). We summarize the signing and verifying procedures in Algorithms 1.10 and 1.11, respectively.

An entity A with private key x signs a message by selecting a random integer k from the interval $[1, q-1]$, and computing $T = g^k \bmod p$, $r = T \bmod q$ and

$$s = k^{-1}(h + xr) \bmod q \tag{1.2}$$

where $h = H(m)$ is the message digest. A's signature on m is the pair (r, s). To verify the signature, an entity must check that (r, s) satisfies equation (1.2). Since the verifier knows neither A's private key x nor k, this equation cannot be directly verified. Note, however, that equation (1.2) is equivalent to

$$k \equiv s^{-1}(h + xr) \pmod q. \tag{1.3}$$

Raising g to both sides of (1.3) yields the equivalent congruence

$$T \equiv g^{hs^{-1}} y^{rs^{-1}} \pmod p.$$

The verifier can therefore compute T and then check that $r = T \bmod q$.

Algorithm 1.10 DSA signature generation

INPUT: DL domain parameters (p, q, g), private key x, message m.
OUTPUT: Signature (r, s).

1. Select $k \in_R [1, q-1]$.
2. Compute $T = g^k \bmod p$.
3. Compute $r = T \bmod q$. If $r = 0$ then go to step 1.
4. Compute $h = H(m)$.
5. Compute $s = k^{-1}(h + xr) \bmod q$. If $s = 0$ then go to step 1.
6. Return(r, s).

Algorithm 1.11 DSA signature verification

INPUT: DL domain parameters (p, q, g), public key y, message m, signature (r, s).
OUTPUT: Acceptance or rejection of the signature.

1. Verify that r and s are integers in the interval $[1, q-1]$. If any verification fails then return("Reject the signature").
2. Compute $h = H(m)$.
3. Compute $w = s^{-1} \bmod q$.
4. Compute $u_1 = hw \bmod q$ and $u_2 = rw \bmod q$.
5. Compute $T = g^{u_1} y^{u_2} \bmod p$.
6. Compute $r' = T \bmod q$.
7. If $r = r'$ then return("Accept the signature");
   Else return("Reject the signature").

1.2.3 Elliptic curve systems

The discrete logarithm systems presented in §1.2.2 can be described in the abstract setting of a finite cyclic group. We introduce some elementary concepts from group theory and explain this generalization. We then look at elliptic curve groups and show how they can be used to implement discrete logarithm systems.

Groups

An *abelian group* $(G, *)$ consists of a set G with a binary operation $* : G \times G \to G$ satisfying the following properties:

(i) (*Associativity*) $a*(b*c) = (a*b)*c$ for all $a, b, c \in G$.
(ii) (*Existence of an identity*) There exists an element $e \in G$ such that $a*e = e*a = a$ for all $a \in G$.
(iii) (*Existence of inverses*) For each $a \in G$, there exists an element $b \in G$, called the *inverse* of a, such that $a*b = b*a = e$.
(iv) (*Commutativity*) $a*b = b*a$ for all $a, b \in G$.

The group operation is usually called addition ($+$) or multiplication ($\cdot$). In the first instance, the group is called an *additive* group, the (additive) identity element is usually denoted by 0, and the (additive) inverse of a is denoted by $-a$. In the second instance, the group is called a *multiplicative* group, the (multiplicative) identity element is usually denoted by 1, and the (multiplicative) inverse of a is denoted by a^{-1}. The group is *finite* if G is a finite set, in which case the number of elements in G is called the *order* of G.

For example, let p be a prime number, and let $\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}$ denote the set of integers modulo p. Then $(\mathbb{F}_p, +)$, where the operation $+$ is defined to be addition of integers modulo p, is a finite additive group of order p with (additive) identity element 0. Also, $(\mathbb{F}_p^*, \cdot)$, where $\mathbb{F}_p^*$ denotes the nonzero elements in $\mathbb{F}_p$ and the operation $\cdot$ is defined to be multiplication of integers modulo p, is a finite multiplicative group of order $p-1$ with (multiplicative) identity element 1. The triple $(\mathbb{F}_p, +, \cdot)$ is a *finite field* (cf. §2.1), denoted more succinctly as $\mathbb{F}_p$.

Now, if G is a finite multiplicative group of order n and $g \in G$, then the smallest positive integer t such that $g^t = 1$ is called the *order* of g; such a t always exists and is a divisor of n. The set $\langle g \rangle = \{g^i : 0 \leq i \leq t-1\}$ of all powers of g is itself a group under the same operation as G, and is called the *cyclic subgroup of G generated by g*. Analogous statements are true if G is written additively. In that instance, the order of $g \in G$ is the smallest positive divisor t of n such that $tg = 0$, and $\langle g \rangle = \{ig : 0 \leq i \leq t-1\}$. Here, tg denotes the element obtained by adding t copies of g. If G has an element g of order n, then G is said to be a *cyclic group* and g is called a *generator* of G.

For example, with the DL domain parameters (p, q, g) defined as in §1.2.2, the multiplicative group $(\mathbb{F}_p^*, \cdot)$ is a cyclic group of order $p-1$. Furthermore, $\langle g \rangle$ is a cyclic subgroup of order q.

Generalized discrete logarithm problem

Suppose now that $(G, \cdot)$ is a multiplicative cyclic group of order n with generator g. Then we can describe the discrete logarithm systems presented in §1.2.2 in the setting of G. For instance, the domain parameters are g and n, the private key is an integer x selected randomly from the interval $[1, n-1]$, and the public key is $y = g^x$. The problem of determining x given g, n and y is the *discrete logarithm problem in G*.

In order for a discrete logarithm system based on G to be efficient, fast algorithms should be known for computing the group operation. For security, the discrete logarithm problem in G should be intractable.

Now, any two cyclic groups of the same order n are essentially the same; that is, they have the same structure even though the elements may be written differently. The different representations of group elements can result in algorithms of varying speeds for computing the group operation and for solving the discrete logarithm problem.

The most popular groups for implementing discrete logarithm systems are the cyclic subgroups of the multiplicative group of a finite field (discussed in §1.2.2), and cyclic subgroups of elliptic curve groups which we introduce next.

Elliptic curve groups

Let p be a prime number, and let $\mathbb{F}_p$ denote the field of integers modulo p. An *elliptic curve* E over $\mathbb{F}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b, \tag{1.4}$$

where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. A pair (x, y), where $x, y \in \mathbb{F}_p$, is a *point* on the curve if (x, y) satisfies the equation (1.4). The *point at infinity*, denoted by ∞, is also said to be on the curve. The set of all the points on E is denoted by $E(\mathbb{F}_p)$. For example, if E is an elliptic curve over $\mathbb{F}_7$ with defining equation

$$y^2 = x^3 + 2x + 4,$$

then the points on E are

$$E(\mathbb{F}_7) = \{\infty, (0,2), (0,5), (1,0), (2,3), (2,4), (3,3), (3,4), (6,1), (6,6)\}.$$

Now, there is a well-known method for adding two elliptic curve points (x_1, y_1) and (x_2, y_2) to produce a third point on the elliptic curve (see §3.1). The addition rule requires a few arithmetic operations (addition, subtraction, multiplication and inversion) in $\mathbb{F}_p$ with the coordinates x_1, y_1, x_2, y_2. With this addition rule, the set of points $E(\mathbb{F}_p)$ forms an (additive) abelian group with ∞ serving as the identity element. Cyclic subgroups of such elliptic curve groups can now be used to implement discrete logarithm systems.

We next illustrate the ideas behind elliptic curve cryptography by describing an elliptic curve analogue of the DL encryption scheme that was introduced in §1.2.2. Such elliptic curve systems, and also the elliptic curve analogue of the DSA signature scheme, are extensively studied in Chapter 4.

Elliptic curve key generation

Let E be an elliptic curve defined over a finite field $\mathbb{F}_p$. Let P be a point in $E(\mathbb{F}_p)$, and suppose that P has prime order n. Then the cyclic subgroup of $E(\mathbb{F}_p)$ generated by P is

$$\langle P \rangle = \{\infty, P, 2P, 3P, \ldots, (n-1)P\}.$$

The prime p, the equation of the elliptic curve E, and the point P and its order n, are the public domain parameters. A private key is an integer d that is selected uniformly at random from the interval $[1, n-1]$, and the corresponding public key is $Q = dP$.

The problem of determining d given the domain parameters and Q is the *elliptic curve discrete logarithm problem* (ECDLP).

Algorithm 1.12 Elliptic curve key pair generation

INPUT: Elliptic curve domain parameters (p, E, P, n).
OUTPUT: Public key Q and private key d.

1. Select $d \in_R [1, n-1]$.
2. Compute $Q = dP$.
3. Return(Q, d).

Elliptic curve encryption scheme

We present the encryption and decryption procedures for the elliptic curve analogue of the basic ElGamal encryption scheme as Algorithms 1.13 and 1.14, respectively. A plaintext m is first represented as a point M, and then encrypted by adding it to kQ where k is a randomly selected integer, and Q is the intended recipient's public key. The sender transmits the points $C_1 = kP$ and $C_2 = M + kQ$ to the recipient who uses her private key d to compute

$$dC_1 = d(kP) = k(dP) = kQ,$$

and thereafter recovers $M = C_2 - kQ$. An eavesdropper who wishes to recover M needs to compute kQ. This task of computing kQ from the domain parameters, Q, and $C_1 = kP$, is the elliptic curve analogue of the Diffie-Hellman problem.

Algorithm 1.13 Basic ElGamal elliptic curve encryption

INPUT: Elliptic curve domain parameters (p, E, P, n), public key Q, plaintext m.
OUTPUT: Ciphertext (C_1, C_2).

1. Represent the message m as a point M in $E(\mathbb{F}_p)$.
2. Select $k \in_R [1, n-1]$.
3. Compute $C_1 = kP$.
4. Compute $C_2 = M + kQ$.
5. Return(C_1, C_2).

Algorithm 1.14 Basic ElGamal elliptic curve decryption

INPUT: Domain parameters (p, E, P, n), private key d, ciphertext (C_1, C_2).
OUTPUT: Plaintext m.

1. Compute $M = C_2 - dC_1$, and extract m from M.
2. Return(m).