MSc Distributed Computing Systems Engineering

Department of Electronic & Computer Engineering

Brunel University

Implementing Elliptic Curve Cryptography in Java

Andreas Gottschol

02/2001

Supervisor: Prof. Malcolm Irving

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science

MSc Distributed Computing Systems Engineering

Department of Electronic & Computer Engineering

Brunel University

Implementing Elliptic Curve Cryptography in Java

Andreas Gottschol

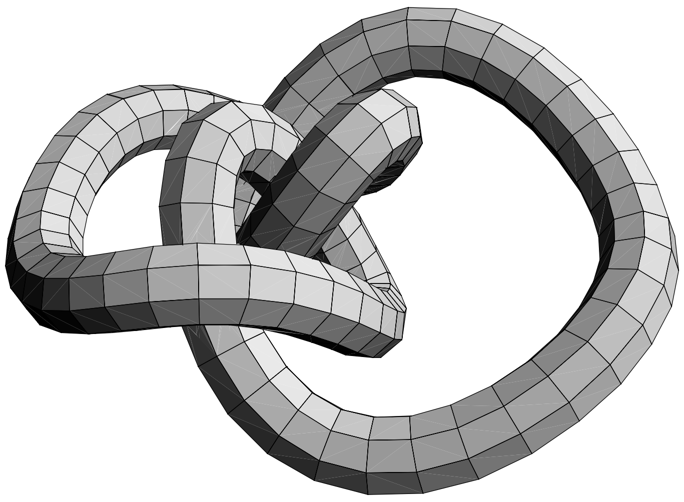
02/2001

Supervisor: Prof. Malcolm Irving

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science

To my Mum and Dad, and to Kerstin.

Thank you so much for your support.



Contents

1 Abstract

Elliptic curves as algebraic or geometric entities have been studied extensively for the past 150 years. They recently gained some reputation in the proof of Fermat’s Last Theorem by the mathematician Andrew Wiles [Singh 1997]. Originally persued for purly aestethic reasons, elliptic curves have recently been utilised in devising algorithms for factoring integers, primality proving and public key cryptography. They were first proposed for the application in cryptography in 1985 independently by Neal Koblitz from the University of Washington, and Victor Miller from IBM.

Elliptic curve cryptography represents a different way to do public key cryptography which offers certain advantages. The idea behind elliptic curve cryptography is to take a well chosen elliptic curve, define it over a certain field, and solve a certain *hard* problem (specifically, a trap-door one-way function) over the defined curve.

The aim of this thesis is to gain a deeper understanding of the mathematics of elliptic curves and their utilisation in cryptography and to implement a cryptographic scheme using elliptic curves in the Java programming language. Further elliptic curve cryptography and the results of the implementation should be compared to other conventional public key crypto scheme, like the RSA or ElGamal public key schemes for example, which were used for years and are still in use in industrial applications.

1.1 Motivation

Why is there so much interest in new cryptosystems based on a complex subject as elliptic curves? The first thing someone will possibly note when beginning to study elliptic curve cryptography is the following: The key size is significantly smaller for an elliptic curve cryptosystem than for a conventional public key crypto scheme, offering the same level of security. A fairly typical key size for RSA is 1024 bits which, according to the literature [Schneier 1996], would take approximately 10^{11} MIPs-years to break. A 160 bit key for an elliptic curve crypto scheme is believed to offer the same level of security which is a key size ration of approximately 1:7. This advantage increases with security level, something which will be important as computer power continually grows. A 2048 bit RSA key and a 210 bit elliptic curve key are nearly equivalent, which increases the elliptic curve to RSA key size ratio from 1:7 to 1:10 [Lenstra et al 1999]. Elliptic curve cryptography is also expected to have less computational overhead than e.g. RSA, primarily because it does not have to analyse prime numbers, which is a quite expensive operation.

Together, these facts seem to allow for a more efficient cryptosystem. Devices for elliptic curve cryptography are supposed to require less storage, less power, less memory, and less bandwidth. This make elliptic curves cryptography suitable for platforms that are constrained, such as wireless devices, handheld computers, smart cards, and ultra thin clients.

2 Cryptographic Background

2.1 Introduction to Cryptography

Cryptography can be defined as the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication.

The basic objectives of cryptography are the following, from which others may be derived:

Confidentiality	Used to keep the content of information from all but those authorised to have it. Secrecy is a term synonymous with confidentiality and privacy. There are a number of approaches to provide confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.
Data integrity	Addresses the unauthorised alteration of data. To assure data integrity, one must have the ability to detect data manipulation by an unauthorised party. Data manipulation includes the insertion, deletion and substitution of data.
Authentication	Is related to identification. This function can be applied to both entities and information itself. Two communicating parties should

identify each other. Information sent over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is devided in two further subsections, *entity authentication* and *data origin authentication*. Data origin authentication implicitly provides data integrity.

Non-repudiation	Prevents an entity from denying previous commitments or actions. A trusted third party is needed when disputes arise due to an entity denying that certain actions were taken.

The goal of cryptography is to adequately address these four areas in both theory and practice. So cryptography is about prevention and detection of cheating, fraud or other malicious activities which makes it a necessary tool in the modern economy.

2.2 Cryptographic Primitives

2.2.1 Hash Functions

A hash function or message digest function takes input data of arbitrary length and computes a fixed length output, called the message digest or hash of the input data. Let *m* be a message of arbitrary length and *h* the hash function. The outcome $z = h(m)$ is called the hash of *m* which has fixed length. Common values for the length of *z* are 128 bit or 160 bit.

To be useful for cryptography, the hash function *h* must have some special properties to prevent forgery. It should not be possible for an opponent to find a message *m'* in feasible time, such that $h(m') = h(m)$. A hash function is called collision-free, if it has this property.

By definition a hash function *h* is called weakly collision-free for *m* if it is computational infeasible to find a message $m' \neq m$ such that $h(m') = h(m)$.

A hash function is called strongly collision-free if it is computational infeasible to to find two arbitrary messages *m* and *m'* with $m' \neq m$ such that $h(m') = h(m)$.

A further property for a hash function in cryptography is that it must be a one-way function. A hash function is one-way if, given a message digest *z*, it is computational infeasible to find a message *m* such that $h(m) = z$.

Hash functions play a major role in the computation of digital signatures. For the digital signature of some data first the message digest for this data is calculated. This message digest is also known as the fingerprint of the message. Then the digest or fingerprint is signed, not the message itself. This is described in 2.2.4 in detail.

2.2.2 Symmetric Key Encryption

Symmetric key encryption schemes are also called secret key encryption schemes due to their property of a secret en-/decryption key which is shared between the communicating parties. Encryption is done with the same key as decryption, which is where the name for *symmetric cryptography* comes from.

Given a message m and a secret key k and an encryption function f such that the encrypted message e is given by $e = f(m,k)$. The decryption function is the inverse of *f*, such that $m = f^{-1}(e,k)$. There are symmetric algorithms where $f = f^{-1}$, but this is not necessarily so.

For f to be useful for cryptography it must provide certain security properties. The key space of k must be big enough to prevent an opponent from trying all possible keys in feasible time. The key space is the number of all possible keys for a particular encryption function *f*. For a key of 8 bit in length and the assumption that all possible bit combinations are valid keys for the algorithm, the number of possible keys is $2^8 = 256$, which gives only little security if at all. The well known DES algorithms has a key space of nearly 56 bit, which is even not enough for modern security needs. Note that the key length of DES is exactly 56 bit, but there are weak keys which reduces the effective key space. A DES key can be broken in feasible time on a modern computer. A good value for key length in modern symmetric algorithms are 128 bit or more.

Another property of f must be that there exist no other key u with $k \neq u$ such that $m = f^{-1}(e,u)$. Further there must be enough evidence to believe that there is no other way to obtain m from e but to solve the function f^{-1} using the key k. That is there is no other way to obtain m from e but to try all possible keys in the key space of k. This is called a brute-force attack.

The conclusion is that the security of symmetric encryption schemes lies in the secret key. The exposure of the secret key renders the system insecure.

2.2.3 Public Key Encryption

As seen in 2.2.2 the security of secret key schemes rely on the fact that the secret key remains secret between the communicating parties. A major drawback of secret key systems is the fact that it requires a prior communication over a secure channel between the parties to establish a shared secret key. This must happen before any ciphertext can be exchanged. As in the case of an email communication it is often difficult or even impossible to establish an secure channel for key exchange.

The idea behind public key cryptography is that it might be possible to find a cryptosystem where it is computational infeasible to determine the decryption function f^{-1} given only the encryption function *f*. If this can be achieved than the encryption function can be made public, e.g. placed in a public directory. This is where the name comes from. The advantage of public key cryptography is the possibility to exchange encrypted messages without prior communication of a shared secret over a secure channel. A message is simply encrypted by using the public encryption function.

The idea of a public key cryptosystem was due to Diffie and Hellman in 1976. The first realisation of a public key system came in 1977 by Rivest, Shamir and Adleman. They invented the well-known RSA public key cryptosystem which is based on the problem of factoring big integer numbers.

The following example should illustrate the RSA public key crypto system. To generate a key pair, entity A must do the following:

1. Generates two large random (and distinct) prime numbers *p* and *q*, each roughly the same size.

2. Compute $n = p \times q$ and $\phi = (p-1)(q-1)$

3. Select a random integer *e*, $1 < e < \phi$, such that $\gcd(\mathrm{e}, \phi) = 1$

4. Compute the unique integer *d*, $1 < d < \phi$, such that $ed \equiv 1 (\bmod \phi)$, using the extented Euclidean algorithm.

5. The public key consists of {*n*, *e*}; the private key is *d*

To encrypt a message for A, B must do the following:

1. Obtain A's authentic public key {*n*, *e*}

2. Respresent the message to encrypt as an integer *m* in the interval [0, *n*-1]

3. Compute $c = m^e \bmod n$

4. Send the ciphertext *c* to A.

To decrypt the message from B, A must do the following:

1. Use the private key *d* to recover the plaintext *m* with $m = c^d \bmod n$

The task for an passive adversary to recover the plaintext *m* from the given ciphertext *c* and the public key information {*n*, *e*} of the intended receiver A is called the RSA problem. A possible approach to solve this problem is to factor *n* first, and then compute ϕ and *d*. The problem of computing the RSA decryption exponent *d* from the public key {*n*, *e*} and the problem of factoring *n* are computational equivalent. When generating RSA keys, it is imperative that the primes *p* and *q* be selected in such a way that factoring $n = p \times q$ is computational infeasible, which means *p* and *q* must be big enough as a first consequence. The RSA public key algorithm may also be used for digital signatures, which is discussed later.

2.2.4 Digital Signatures

Handwritten signatures are used in everyday situations such as withdrawing money from a bank account or signing a contract. The conventional signature is attached to a document by hand to specify the person responsible for it. It is some form of authentication based on the assumption that the handwriting of some person is relativly hard to forge.

A signature scheme or digital signature is a method of signing a message stored in electronic form. As such, the signed message can be transmitted over a digital channel. As it is relatively easy to forge digital information, electronic signature schemes differ from the conventional signature in a number of ways.

First of all, with the conventional signature, the signature is physically part of the document being signed. However, it makes very little sense to simply attach a digital signature to the electronic document. So the algorithm being used must somehow tightly bind the signature to the document.

Second is the question of verification of the signature. A conventional signature is verified by comparing it to another, supposedly authentic signature. In case of a credit card purchase for example, the salesperson is supposed to compare the signature on the sales slip against the signature on the credit card in order to verify the signature. This works as long as the assumption is right, that it is relatively hard to forge the signature of someone else (which is not really the case, of course). On the other hand digital signatures must be verified using a publicly known verification algorithm. The use of a *secure* signature scheme will prevent the possibility of forgery. The meaning of the term secure here is discussed later.

Another fundamental difference between signatures in the *real* world and digital signatures is the fact that a copy of the signed digital message is identical to the original. On the other hand a physical copy of a signed paper document can usually be distinguished from the original. This is especially true in the case of documents which are printed on special paper (e.g. paper with a watermark or with other special features which are hard to forge or to copy). That means that special care must be taken to prevent the reuse of a digitally signed document or message. For example in the case of a digitally signed document, authorising the bearer to make a single withdrawal of £100 from some bank account, the signer of this document wants the bearer to do so only once. So the document itself must contain information which prevents it from being resused, e.g. a timestamp incorporated in the digital signature.

A digital signature scheme consists of two parts: the signing algorithm and the verification algorithm. One party can sign a message m using the secret signing algorithm s. The resulting signature $s(m)$ can subsequently be verified using the public verification algorithm *v*. Given a pair (m, y), the verification algorithm’s outcome is simply true or false, depending on whether the signature is authentic or not.

Note that normally not the signature algorithm itself is secret, but the key used in the signature algorithm is. In a public key scheme, the private key is used for computing signatures. As the name states, the private key is meant to stay *private* or *secret*. By definition a secret key is the key used in symmetric cryptography, which must remain a shared secret between the communicating parties. It should not be confused with the term private key which is used in the context of public key schemes.

A more formal definition of a digital signature scheme is given here. A digital signature scheme is a five-tupel (M, A, K, S, V), where the following conditions are satisfied:

1. M is a finite set of possible messages

2. A is a finite set of possible signatures

3. K is the keyspace, which is a finite set of possible keys

4. For each $k \in K$, there is a signing algorithm $s_k \in S$ and a corresponding verification algorithm $v_k \in V$. Each $s_k : M \to A$ and corresponding $v_k : M \times A \to \{true, false\}$ are functions such that the following equation is satisfied for every message $m \in M$ and for every signature $y \in A$:

$$v(m, y) = \begin{cases} true : y = s(m) \\ false : y \neq s(m) \end{cases}$$

For every $k \in K$, the functions s_k and v_k should be polynomial time functions. Here s_k is a secret function and v_k is public. It should be computationally infeasible for someone to forge somebody else's signature on a message *m*. A signature scheme cannot be unconditionally secure, since it is always possible to test all signatures *y* for a message *m* using the public verification algorithm v_k until the right signature is found. So given sufficient time and computational resources, a digital signature can always be forged. Thus, as is the problem with public key cryptosystems, the goal is to find a signature scheme which is computationally secure. This means that it should be infeasible to find the right signature *y* for a message *m* by simply computing all possible signatures for the message *m*.

Today's digitals signature schemes are mostly based on public key cryptography. However, this is not the only way digital signature can be realised. There are also signature schemes based on traditional, symmetric cryptography. This topic is not discussed in this document, but further material can be found in [Menezes et al 1997] and [Schneier 1996].

The following example should illustrate a scheme for digital signatures based on the RSA public key crypto system. The key generation process is the same as described in 2.2.3 and is ommited here for brevity. So the public key consists of $\{n, e\}$ and the private key is represented by *d*. Signing a message with RSA is simply a reversion of the encryption scheme. So to sign a message, A must use its private key and the decryption rule on the message. This is defacto an encryption operation with the own private key. To verify the signature, B must use A’s public key and the encryption rule on the signature, which is defacto a decryption with A’s public key.

So to sign a message *m*, A must do the following:

1. Take the private key *d* and the message $m \in M$
2. Transform the message to sign into an integer $\tilde{m}$ in the interval [0, *n*-1] . This is usually be done by a special transformation function such that $\tilde{m} = R(m)$
3. Compute $s = \tilde{m}^d \bmod n$
4. A’s signature for *m* is *s*.

To verify A‘s signature for the message *m*, B must do the following:

1. Obtain A’s authentic public key {*n*, *e*}

2. Compute $\tilde{m} = s^e \bmod n$

3. Verify that $\tilde{m} \in M_R$; if not, reject the signature.

4. Recover $m = R^{-1}(\tilde{m})$

A proof that the signature verification works is given in [Menezes et al 1997]: If *s* is a signature for the message *m*, then $s \equiv \tilde{m}^d \bmod n$ where $\tilde{m} = R(m)$. Since $ed \equiv 1 (\bmod \phi)$, $s^e \equiv \tilde{m}^{ed} \equiv \tilde{m} (\bmod n)$. Finally, $R^{-1}(\tilde{m}) = R^{-1}(R(m)) = m$.

2.3 Number Theory

This section provides an overview of basic algebraic objects and their properties which is helpful for understandig the further material. By definition a binary operation * on a set *S* is a mapping from *S* * *S* to *S*. That is, * is a rule which assigns to each ordered pair from *S* an element of *S*.

2.3.1 Group

A group (*G*, *) consists of a set *G* with a binary operation * on *G* satisfying the following axioms:

1. The group operation is associative. That is, $a*(b*c)=(a*b)*c$ for all $a,b,c\in G$.

2. There is an element $1\in G$, called the identity element, such that $a*1=1*a=a$ for all $a\in G$.

3. For each $a\in G$ there exists an element $a^{-1}\in G$, called the inverse of *a*, such that $a*a^{-1}=a^{-1}*a=1$.

A group *G* is abelian (or commutative) if, furthermore,

4. $a*b=b*a$ for all $a,b\in G$.

Note that in this case the multiplicative group notation (*G*, *) has been used for the group operation. If the group operation is addition, then the group is called an additive group, the identity element is denoted by 0, and the inverse of a is denoted $-a$.

A group *G* is finite, if the number of elements in *G*, written as $|G|$, is finite. The number of elements in a finite group is called its *order*.

As an example consider the following: The set of integers $\mathbb{Z}$ with the operation of addition forms a group. The identity element is 0 and the inverse of an integer a is $-a$. As there are infinitely many integers in $\mathbb{Z}$, the group is called infinite.

A non-empty subset *H* of a group *G* is a subgroup of *G* if *H* is itself a group with respect to the operation of *G*. If *H* is a subgroup of *G* and $H \neq G$, then *H* is called a proper subgroup of *G*.

A group *G* is cyclic if there exists an element $\alpha \in G$ such that for each $b \in G$ there is an integer i with $b = \alpha^i$. Such an element α is called a generator of *G*.

If *G* is a group and $a \in G$, then the set of all powers of *a* form a cyclic subgroup of *G*, called the subgroup generated by *a*, and denoted by $\langle a \rangle$.

Let *G* be a group and $a \in G$. The order of the group element *a* is defined to be the least positive integer *t* such that $a^t = 1$, provided that such an integer exists. If such a *t* does not exist, then the order of the group element *a* is defined to be ∞.

Let *G* be a group, and let $a \in G$ be an element of finite order *t*. Then $|\langle a \rangle|$, the size of the subgroup generated by *a*, is equal to *t*.

As Lagrange's theorem states: If *G* is a finite group and *H* is a subgroup of *G*, then $|H|$ devides $|G|$. Hence, if $a \in G$, the order of a devides $|G|$.

Every subgroup of a cyclic group *G* is also cyclic. In fact, if *G* is cyclic group of order *n*, then for each positive divisor *d* of *n*, *G* contains exactly one subgroup of order *d*.

Let *G* be a group. If the order of $a \in G$ is *t*, then the order of a^k is $\frac{t}{\gcd(t,k)}$.

If *G* is a cyclic group of order *n* and $d \mid n$ (i.e. *d* devides *n*), then *G* has exactly $\phi(d)$ elements of order *d*. In particular, *G* has $\phi(n)$ generators.

As an example consider the multiplicative group $\mathbb{Z}_{19}^*$={1,2,3,...,18} of order 18. The group is cyclic and a generator of the group is $\alpha = 2$. The subgroups of $\mathbb{Z}_{19}^*$, and their generators, are listed in Table 1: The subgroups of $\mathbb{Z}_{19}^*$.

Subgroup	*Generators*	*Order*
{1}	1	1
{1, 18}	18	2
{1, 7, 11}	7, 11	3
{1, 7, 8, 11, 12, 18}	8, 12	6
{1, 4, 5, 6, 7, 9, 11, 16, 17}	4, 5, 6, 9, 16, 17	9
{1, 2, 3,..., 18}	2, 3, 10, 13, 14, 15	18

Table 1: The subgroups of $\mathbb{Z}_{19}^*$

2.3.2 Ring

A ring (*R*, +, *) consists of a set *R* with two binary operations arbitrarily denoted + (addition) and * (multiplication) on *R*, satisfying the following axioms.

1. (*R*, +) is a abelian group with identity denoted 0.

2. The operation * is associative. That is, $a*(b*c)=(a*b)*c$ for all $a,b,c\in R$.

3. There is a multiplicative identity denoted 1, with $1\neq 0$, such that $1*a=a*1=a$ for all $a\in R$.

4. The operation * is distributive over +. That is , $a*(b+c)=(a*b)+(a*c)$ and $(b+c)*a=(b*a)+(c*a)$ for all $a,b,c\in R$.

The ring is a commutative ring if $a*b=b*a$ for all $a,b\in R$.

As an example consider the following: The set of integers $\mathbb{Z}$ with the usual operations of addition and multiplication is a commutative ring. The set of $\mathbb{Z}_n$ with addition and multiplication performed modulo *n* is also a commutative ring.

An element *a* of a ring *R* is called a *unit* or an *invertible element* if there is an element $b\in R$ such that $a*b=1$. The set of units in a ring *R* forms a group under multiplication, called the *group of units* of *R*.

As an example: The group of units of the ring $\mathbb{Z}_n$ is $\mathbb{Z}_n^*$.

2.3.3 Field

A field is a commutative ring in which all nonzero elements have multiplicative inverses. The characteristic of a field is 0 if 1+1+1+...+1 (*m* times) is never equal to 0 for any $m \geq 1$. Otherwise, the characteristic of the field is the least positive integer *m* such that $\sum_{i=1}^{m} 1$ equals 0.

Consider the following example: The set of integers under the usual operations of addition and multiplication is not a field, since the only nonzero integers with multiplicative inverses are 1 and –1. However, the rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$ form fields of characteristic 0 under the usual operations.

$\mathbb{Z}_n$ is a field (under the usual operations of addition and multiplication modulo *n*) if and only if *n* is a prime number. If *n* is prime, then $\mathbb{Z}_n$ has characteristic *n*. Furthermore if the characteristic *m* of a field is not 0, then *m* is a prime number.

A subset *F* of a field *E* is *subfield* of *E* if *F* itself is a field with respect to the operations of *E*. If this is the case, *E* is said to be an *extension field* of *F*.

A relatively informal definition for a field is given by [Berlekamp 1984]: A field is a set of elements, including 0 and 1, any pair of which may be added or multiplied (denoted by + and *, respectively) to give a unique result in the field. The addition and multiplication are associative and commutative, and multiplication distributes over addition in the usual way: $u*(v+w)=u*v+u*w$. Every nonzero field element *u* has a unique reciprocal field element $1/u$, such that $u*(1/u)=1$. For every field element *u*, $0+u=1*u$, and $0*u=0$. [...] The order of a field is the number of elements in the field. If the order is infinite, we call the field an *infinite field*; if the oder is finite we call the field a *finite field*. The rational numbers, the real number, and the complex numbers are examples of infinite fields. If *p* is a prime, the integers *mod p* form a finite field of order *p*. If *f(x)* is an irreducible polynomial of degree *m* with coefficients in the field of integers *mod p*, then the residue classes *mod f(x)* form a finite field of the order p^m.

2.3.4 Finite Field

By definition a finite field is a field *F* which contains a finite number of elements. The order of the field *F* is the number of elements in *F*.

If *F* is a finite field, then *F* contains p^m elements for some prime number *p* and integer $m \geq 1$. For every prime power oder p^m, there is a unique finite field of order p^m. This field is denoted by $\mathbb{F}_{p^m}$, or sometimes by $GF(p^m)$. This group of fields is also known as *Galois Fields*. Two fields are *isomorphic* if they are structurally the same, although the representation of their field elements may be different. Note that if *p* is a prime then $\mathbb{Z}_p$ is a field, and hence every field of order *p* is isomorphic to $\mathbb{Z}_p$.

If $\mathbb{F}_q$ is finite field of order $q = p^m$, with *p* a prime number, then the characteristic of $\mathbb{F}_q$ is *p*. Moreover, $\mathbb{F}_q$ contains a copy of $\mathbb{Z}_p$ as a subfield. Hence $\mathbb{F}_q$ can be regarded as an extension field of $\mathbb{Z}_p$ of degree *m*.

Let $\mathbb{F}_q$ be a finite field of order $q = p^m$. Then every subfield of $\mathbb{F}_q$ has order p^n, for some *n* that is a positive divisor of *m*. Conversely, if *n* is a positive divisor of *m*, then there is exactly one subfield of $\mathbb{F}_q$ of order p^n. An element $a \in \mathbb{F}_q$ is in the subfield $\mathbb{F}_q^n$ if and only if $a^{p^n} = a$.

By definition the non-zero elements of $\mathbb{F}_q$ form a group under multiplication called *multiplicative group* of $\mathbb{F}_q$, denoted by $\mathbb{F}_q^*$. $\mathbb{F}_q^*$ is a cyclic group of order $q-1$. Hence $a^q = a$ for all $a \in \mathbb{F}_q$.

A genererator of the cyclic group $\mathbb{F}_q^*$ is called a *primitive element* or a *generator* of $\mathbb{F}_q$.

A commonly used representation for the elements of a finite field $\mathbb{F}_q$, where $q = p^m$ and *p* a prime number, is a *polynomial basis representation*. The term of polynomial basis representation is discussed later. If $m = 1$, then $\mathbb{F}_q$ is just $\mathbb{Z}_p$ and arithmetic is performed modulo *p*.

Let $f(x) \in \mathbb{Z}_p$ [x] be an irreducible polynomial of degree *m*. Then $\mathbb{Z}_p$ [x] / ($f(x)$) is a finite field of order p^m. Addition and multiplication of polynomials is performed modulo $f(x)$. For each $m \geq 1$, there exists a monic irreducible polynomial of degree *m* over $\mathbb{Z}_p$. Hence, every finite field has a polynomial basis representation.

As an example regard the finite field $\mathbb{F}_{2^4}$ of characteristic 2 and order 16. It can be verified that the polynomial $f(x) = x^4 + x + 1$ is reducible over $\mathbb{Z}_2$. Hence the finite field $\mathbb{F}_{2^4}$ can be represented as the set of all polynomials over $\mathbb{F}_2$ of degree less than 4. This can be written as:

$$\mathbb{F}_{2^4} = \{a_3x^3 + a_2x^2 + a_1x + a_0 \mid a_i \in \{0,1\}\}$$

For convenience the polynomial $a_3x^3 + a_2x^2 + a_1x + a_0$ is normally represented by the vector $(a_3a_2a_1a_0)$ of length 4. So the term from above becomes:

$$\mathbb{F}_{2^4} = \{(a_3a_2a_1a_0) \mid a_i \in \{0,1\}\}$$

The following example shows how arithmetic is done in the field $\mathbb{F}_{2^4}$:

1. Field elementes are added componentwise which is simply xor-ing the components, e.g. $(1001)+(1011)=(0010)$

2. Multiplication is performed modulo the irreducible polynomial. The result is the polynomial multiplication of the field elements reduced modulo $f(x)$:

   $(x^3+x^2+1)\cdot(x^3+1)=x^6+x^5+x^2+1\equiv x^3+x^2+x^1+1 (\bmod f(x))$

   Or in vector representation:

   $(1101)\cdot(1001)=(110011) \bmod (10011)=(1111)$

3. The multiplicative identity of $\mathbb{F}_{2^4}$ is (0001)

4. The inverse of the element (1011) is (0101). This can be verified by observing that

   $(x^3+x+1)\cdot(x^2+1)=x^5+x^2+x+1\equiv 1 (\bmod f(x))$

   So $(1011)\cdot(0101)=$ (0001)

$f(x)$ is a primitive polynomial, or, equivalentely, the field element $x=(0010)$ is a generator of $\mathbb{F}^*_{2^4}$. This can be verified by checking that all non-zero field elements of $\mathbb{F}_{2^4}$ can be obtained as a power of *x*.

A list of all the powers of x and the resulting field elements in $\mathbb{F}_{2^4}$ is given in the following Table 2: The powers of x modulo the irreducible polynomial $f(x) = x^4 + x + 1$.

i	$x^i \bmod (x^4 + x + 1)$	Field element (vector notation)
0	1	(0001)
1	x	(0010)
2	x^2	(0100)
3	x^3	(1000)
4	$x+1$	(0011)
5	$x^2 + x$	(0110)
6	$x^3 + x^2$	(1100)
7	$x^3 + x + 1$	(1011)
8	$x^2 + 1$	(0101)
9	$x^3 + x$	(1010)
10	$x^2 + x + 1$	(0111)
11	$x^3 + x^2 + x$	(1110)
12	$x^3 + x^2 + x + 1$	(1111)
13	$x^3 + x^2 + 1$	(1101)
14	$x^3 + 1$	(1001)

Table 2: The powers of x modulo the irreducible polynomial $f(x) = x^4 + x + 1$

2.4 Number Theoretic Problems

The security of cryptographic algorithms requires the existence of a mathematical problem which is hard or infeasible to solve without the knowledge of some secret information. The following section briefly describes some of the number theoretic problems which are the basis of today's public key cryptography.

2.4.1 The RSA Problem

The examples of section 2.2.3 and 2.2.4 showed the principle of the RSA algorihm which is useful for public key encryption and digital signatures. The reason why RSA was chosen for the examples is the simplicity and elegance of the algorithm. The basis for the security of RSA is the intractability of the so called RSA problem:

Given a positive integer *n* that is a product of two distinct odd primes *p* and *q*, a positive integer *e* such that $\gcd(e,(p-1)(q-1)) = 1$, and an integer *c*, find an integer *m* such that $m^e \equiv c(\bmod n)$. [Note: Odd primes means all prime numbers excluding 2, which is sometimes called the "oddest" of all primes because of its special properties]

Less formal, the RSA problem is that of finding *e*'th roots modulo a composite integer *n*. The conditions imposed on the problem parameters *n* and *e* ensure that for each integer $c \in \{0,1,,...,n-1\}$ there is exactly one $m \in \{0,1,...,n-1\}$ such that $m^e \equiv c(\bmod n)$. As can be shown the RSA problem can be easily solved if the factors of *n* are known. That is, the RSA problem reduces to the integer factorisation problem. It is believed that the RSA problem and the integer factorisation problem are computationally equivalent, although no proof for that is known.

2.4.2 The Discrete Logarithm Problem (DLP)

The security of many cryptographic schemes depends on the intractability of the Discrete Logarithm Problem (abbreviated DLP). Examples for algorithms based on the DLP are the Diffie-Hellman key agreement scheme, the ElGamal encryption and signature scheme and its variant, the quite popular Digital Signature Algorithm (DSA). Exactly spoken these cryptographic schemes are based on the intractability of the Diffie-Hellman Problem, which is closely related to the DLP. However, this will be discussed in 2.4.3.

Let G be a finite cyclic group of order n. Let α be a generator of G, and let $\beta \in G$. The discrete logarithm of β to the base α, denoted $\log_\alpha \beta$, is the unique integer x, $0 \le x \le n-1$ such that $\beta = \alpha^x$.

The Discrete Logarithm Problems can be stated as:

Given a prime p, a generator α of $\mathbb{Z}_p^*$, and an element $\beta \in \mathbb{Z}_p^*$, find the integer x, $0 \le x \le p-2$, such that $\alpha^x \equiv \beta (\bmod p)$.

As an example let the prime $p = 97$. Then $\mathbb{Z}_{97}^{*}$ forms a finite cyclic group of order $n = 96$. A generator of $\mathbb{Z}_{97}^{*}$ is $\alpha = 5$. With $\beta = 35$ find the unique integer *x*, such that $\alpha^{x} \equiv \beta (\bmod p)$. Since $5^{32} \equiv 35 (\bmod 97)$, $x = \log_{5} 35 = 32$ in $\mathbb{Z}_{97}^{*}$.

The Generalised Discrete Logarithm Problem (GDLP) can be stated as:

Given a finite cyclic group *G* of order *n*, a generator α of *G*, and an element $\beta \in G$, find the integer *x*, $0 \le x \le n-1$, such that $\alpha^{x} = \beta$.

The most obvious algorithm to solve the GLDP is the method of exaustive search, which is to successively compute $\alpha^{0}, \alpha^{1}, \alpha^{2}, \alpha^{3}$, ... until β is obtained. This method requires $O(n)$ multiplications, where *n* is the order of α. This is inefficient and even infeasible if given the case that *n* is a large number. There are more effective algorithms known for the DLP, which are *Pollard's rho* algorithm for logarithms, the *Pohlig-Hellman* algorithm or the *Index-calculus* algorithm, which is the most effective algorithm for computing discrete logarithms known today. Those algorithms are not dicussed further here. See [Schneier 1996] or [Menezes et al 1997] for reference.

The groups of most interest in cryptography are the multiplicative group $\mathbb{F}_q^*$ of the finite field $\mathbb{F}_q$. This includes the particular case of the multiplicative group $\mathbb{Z}_p^*$ of the integers modulo a prime *p* as shown in the example above. Further particular groups are the multiplicative group $\mathbb{F}_{2^m}^*$ of the finite field $\mathbb{F}_{2^m}$ of characteristic two and the group of units $\mathbb{Z}_n^*$, where *n* is a composite integer.

Of special interest in this paper is the group of points on an elliptic curve defined over a finite field which is discussed in detail later.

2.4.3 The Diffie-Hellman Problem

The Diffie-Hellman Problem is mentioned here because of its significance for public key cryptography. Its intractability forms the basis for the security of many cryptographic schemes like Diffie-Hellman key aggreement, ElGamal public key encryption and signature and the Digital Signature Algorithm (DSA). It is closely related to the Discrete Logarithm Problem (DLP) of 2.4.2.

The Diffie-Hellman Problem (DHP) is the following:

Given a prime number *p*, a generator α of the multiplicative group $\mathbb{Z}_p^*$ and the elements $\alpha^a \bmod p$ and $\alpha^b \bmod p$, find $\alpha^{ab} \bmod p$.

There is also a generalised version of the Diffie-Hellman Problem, aquivalent to the generalised version of the Discrete Logarithm Problem. This is called the Generalised Diffie-Hellman Problem (GDHP):

Given a finite cyclic group *G*, a generator α of *G*, and the group elements α^a and α^b, find α^{ab}.

Suppose that the Discrete Logarithm Problem in $\mathbb{Z}_p^*$ could be efficiently solved. Then given α, *p*, $\alpha^a \bmod p$ and $\alpha^b \bmod p$, one could first find *a* by solving the DLP, and then compute $(\alpha^a)^b \bmod p = \alpha^{ab} \bmod p$. This results in the fact that the Diffie-Hellman Problem polytime reduces to the Discrete Logarithm Problem.

2.5 ElGamal Public Key Encryption Scheme

The ElGamal public-key encryption scheme is named after its inventor, Taher ElGamal. The security of the ElGamal scheme is based on the intractability of the discrete logarithm problem (DLP) of 2.4.2 and the Diffie-Hellman problem of 2.4.3. The following sections show how the scheme works in detail.

2.5.1 Key generation

Each entity generates a public key and a corresponding private key, using the following algorithm:

1. Generate a large random prime *p* and a generator α of the multiplicative group $\mathbb{Z}_p^*$ of the integers modulo *p*.

2. Select a random integer *a*, $1 \le a \le p-2$, and compute $y = \alpha^a \bmod p$.

3. A's public key is (p, α, y); A's private key is *a*.

2.5.2 Public Key Encryption

If B wishes to encrypt a message for A, B must do the following:

1. Obtain the (authentic) public key of A, which consists of (p, α, y).

2. Represent the message as an integer *m* in the range $\{0,1,...,p-1\}$.

3. Select a random integer *k*, $1 \le k \le p-2$.

4. Compute $\gamma = \alpha^k \bmod p$ and $\delta = m \cdot (\alpha^a)^k \bmod p$.

5. Send the ciphertext $c = (\gamma, \delta)$ to A.

To recover the plaintext *m* from *c*, A must do the following:

1. Use the private key *a* to compute $\gamma^{p-1-a} \bmod p$. Note that $\gamma^{p-1-a} = \gamma^{-a} = \alpha^{-ak}$.

2. Recover the plaintext *m* by computing $(\gamma^{-a}) \cdot \delta \bmod p$.

A short proof that the decryption works is given by:

$$\gamma^{-a} \cdot \delta \equiv \alpha^{-ak} \cdot m \cdot \alpha^{ak} \equiv m \pmod p.$$

2.5.3 ElGamal Signature

The ElGamal signature scheme requires a hash function to generate a digital signature as an appendix for messages of arbitrary length. The hash function is given by $h : \{0,1\}^* \to \mathbb{Z}_p$, where p is a prime number. This means that the hash function transforms an bitvector of arbitrary length into an integer within the group $\mathbb{Z}_p$. Key generation for the signature scheme is the same as for public key encryption. See 2.5.1 for detail.

So to sign a binary message *m* of arbitrary lenght, entity A must do the following:

1. Select a random secret integer *k*, $1 \le k \le p-2$, with $\gcd(k, p-1) = 1$
2. Compute $r = \alpha^k \bmod p$
3. Compute $k^{-1} \bmod (p-1)$
4. Compute $s = k^{-1}\{h(m) - ar\} \bmod (p-1)$
5. A’s signature for *m* is the pair (r, s)

To verify A’s signature (r,s) on the message *m*, entity B must do the following:

1. Obtain the (authentic) public key of A, which consists of (p,α,y).
2. Verify that $1 \le r \le p-1$; if not, the signature is not valid and should be rejected
3. Compute $v_1 = y^r r^s \bmod p$
4. Compute the hash for the message $h(m)$, and $v_2 = \alpha^{h(m)} \bmod p$
5. Verify that the signature is valid. This is the case if and only if $v_1 = v_2$.

A proof that the signature scheme works is given by:

If the signature was indeed generated by entity A, then $s \equiv k^{-1}\{h(m) - ar\}(\bmod p-1)$. Multiplying both sides of the congruence by k gives $ks \equiv h(m) - ar(\bmod p-1)$. Rearranging the term yields $h(m) \equiv ar + ks(\bmod p-1)$. This further implies $\alpha^{h(m)} \equiv \alpha^{ar+ks} \equiv (\alpha^a)^r r^s(\bmod p)$. Thus, $v_1 = v_2$, as required.

2.5.4 Generalised ElGamal Encryption Scheme

The ElGamal encryption scheme is typically decribed in the setting of the multiplicative group $\mathbb{Z}_p^*$. However it can be easily generalised to work in any finite cyclic group *G*. As with *normal* ElGamal, the security of the generalised form of ElGamal encryption is based on the intractability of the discrete logarithm problem in the group *G*. So the group *G* should be carefully chosen to satisfy the following conditions:

1. For *efficiency*, the group operation in *G* should be relatively easy to apply.
2. For *security*, the discrete logarithm problem (DLP) in *G* should be computationally infeasible to solve.

The following is a incomplete list of groups that appear to meet these criteria. They, among others, have received the most attention in cryptography in the recent years.

1. The multiplicative group $\mathbb{Z}_p^*$ of the integers modulo a prime number *p*.
2. The multiplicative group $\mathbb{F}_{2^m}^*$ of the finite field $\mathbb{F}_{2^m}$ of characteristic two.
3. The group of points on an elliptic curve over a finite field.

As this paper is supposed to be on elliptic curve cryptography, the group of points on an elliptic curve over a finite field is of special interest here. The ElGamal encryption and signature schemes are perfectly suitable for the implementation of elliptic curve cryptography, as is the later discussed DSA.

The algorithm for the generalised ElGamal is shown below. To generate a pair of private and public key, entity A must do the following:

1. Select an appropriate cyclic group *G* of order *n*, with a generator α.
2. Select a random integer *a*, $1 \leq a \leq n-1$, and compute the group element α^a.
3. A's public key is the pair (α, α^a), together with a description of the arithmetic in *G* (how to multiply elements in *G*); A's private key consists of *a*.

To encrypt a message for A, entity B must do the following:

1. Obtain the (authentic) public key of A, which consists of (α, α^a).
2. Represent the message as an element *m* of the group *G*.
3. Select a random integer *k*, $1 \leq k \leq n-1$.
4. Compute $\gamma = \alpha^k$ and $\delta = m(\alpha^a)^k$.
5. Send the ciphertext $c = (\gamma, \delta)$ to A.

To decrypt B's message, A should do the following:

1. Use the private key *a* to compute γ^a and then γ^{-a}.

2. Recover the message *m* by computing $(\gamma^{-a}) \cdot \delta$

To simplify things, all entities may elect to use the same cyclic group *G* and generator α. In this case the generator α and description of the arithmetic in *G* do not have to be published as part of the public key. The generalised version of the ElGamal public key encryption scheme makes it perfectly suitable for the implementation of elliptic curve cryptography, where *G* is the group of points on an elliptic curve over a finite field.

2.6 DSA Signature Scheme

The DSA was proposed as an algorithm for digital signatures by the U.S. National Institute of Standards and Technology (NIST) in 1991. It became a U.S. Federal Information Processing standard which is also known as DSS (Digital Signature Standard). The Digital Signature Algorithm (DSA) is a variant of the ElGamal signature scheme. The classic variant is defined over the multiplicative group $\mathbb{Z}_p^*$ of the integers modulo a prime number *p*. But all of the mechanisms can be generalised to any finite cyclic group as is the case with ElGamal. The signature mechanism requires a hash function $h:\{0,1\}^* \rightarrow \mathbb{Z}_q$ for some integer *q*. Note that the DSS (or DSA) explicitly requires the use of the SHA-1 secure hash function as part of the standard.

2.6.1 Key Generation

The key generation process is a bit more complex for DSA than for the ElGamal signature scheme. This is the result of working in a cyclic subgroup of order *q* of $\mathbb{Z}_p^*$ and not in the cyclic group of $\mathbb{Z}_p^*$ itself.

To create a pair of private and public key, entity A should to the following:

1. Select a prime number q such that $2^{159} < q < 2^{160}$.

2. Choose t so that $0 \le t \le 8$, and select a prime number p where $2^{511+64t} < p < 2^{512+64t}$, with the property $q \,|\, (p-1)$; (which means that q devides (p-1)).

3. Select a generator α of the unique cyclic subgroup of order q in $\mathbb{Z}_p^*$.

   3.1 Select an element $g \in \mathbb{Z}_p^*$ and compute $\alpha = g^{(p-1)/q} \bmod p$.

   3.2 If $\alpha = 1$ then repeat step 3.1 with another g. If $\alpha \ne 1$ then α is a generator.

4. Select a random integer a such that $1 \le a \le q-1$.

5. Compute $y = \alpha^a \bmod p$.

6. A's public key is (p, q, α, y); A's private key is a.

Note that for the algorithm the prime number q must be chosen first. Then one must try to find a second prime number p such that q devides (p-1).

2.6.2 Signature Generation and Verification

The message m of arbitry length must be represented as an integer in $\mathbb{Z}_q$. This is done by computing $h(m)$, with h a secure hash function.

To sign a message, entity A must do the following:

1. Select a random secret integer k with $1 < k < q - 1$.
2. Compute $r = (\alpha^k \bmod p) \bmod q$.
3. Compute $k^{-1} \bmod q$.
4. Compute $s = k^{-1}\{h(m) + ar\} \bmod q$. Note that $h(m)$ must be the SHA-1 secure hash algorithm for the DSA standard.
5. A's signature for the message m is the pair (r, s).

To verify A's signature, entity B must do the following:

1. Obtain the (authentic) public key of A, which consists of (p, q, α, y).
2. Verify that $0 < r < q$ and $0 < s < q$; if not, the signatur is not valid and should be rejected.
3. Compute $w = s^{-1} \bmod q$ and $h(m)$.

4. Compute $u_1 = w \cdot h(m) \bmod q$ and $u_2 = rw \bmod q$.

5. Compute $v = (\alpha^{u_1} y^{u_2} \bmod p) \bmod q$.

6. The signature is valid if and only if $v = r$.

A proof that the signature scheme works is given by:

If (r, s) is a legitimate signature of entity A on the message *m*, then $h(m) \equiv -ar + ks (\bmod q)$ must hold. Multiplying both sides of this congruence by *w* and rearranging the term gives $w \cdot h(m) + arw \equiv k (\bmod q)$. This can be simplified to $u_1 + au_2 \equiv k (\bmod q)$. Raising α to both sides of this equation simply results in $(\alpha^{u_1} y^{u_2} \bmod q) = (\alpha^k \bmod p) \bmod q$. Hence, $v = r$, as required.

There is also a generalised version of the DSA scheme which, as is the case for ElGamal, makes it perfectly suitable for the implementation of elliptic curve cryptography, where *G* is the group of points on an elliptic curve over a finite field. This is dicussed later.

3 The Arithmetic of Elliptic Curves

The term *elliptic curve* is derived from the theory of elliptic functions. As they are obviously not ellipses, a brief explanation of how the name arises seems appropriate here. Simply speaking elliptic integrals are for ellipses what the trigonometric functions sine, cosine and tangens are for circles.

An elliptic integral has the form $\int \frac{dx}{\sqrt{4x^3 - g_2 x - g_3}}$

The values taken by an elliptic integral can be considered to be on a torus. The inverse function of an elliptic integral is a doubly periodic function called an *elliptic function*. According to [Blake et al 1999] it turns out that every doubly periodic function $\wp$ with periods that are independant over $\mathbb{R}$ satisfies an equation of the form $\wp'^2 = 4\wp^3 - g_2\wp - g_3$ for some constants g_2 and g_3.

Such a function is refered to as a Weierstrass $\wp$ function. If the pair $(\wp, \wp')$ is considered as being a point in space, then the solution to the equation provides a mapping from a torus to the curve $y^2 = 4x^3 - g_2 x - g_3$ which is an example of an elliptic curve.

The elliptic curve equation mostly used is the *affine* version of the Weierstrass equation, which is given by:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Equation 3-1

Here $a_i \in \mathbb{K}$, where $\mathbb{K}$ is the field over which the elliptic curve is defined. The elliptic curve is defined as the set of sollutions of the equation in $\mathbb{K}$.

Now with the definition of an elliptic curve there exists the possibility of doing arithmetics over it. The arithmetic defined over elliptic curves is basically adding two point on a curve to get a third point, which is also on the curve. For cryptographic purposes another fact is interesting: the difficulty to tell which two points were added to get the third one. The explanation of elliptic curves arithmetic starts with elliptic curves over real numbers $\mathbb{R}$.

3.1 Elliptic Curves over Real Numbers

An elliptic curve over real numbers can be defined as a set of points (*x*,*y*) in the plane of real numbers which satisfies an elliptic curve equation of the following simplified Weierstrass form: $y^2 = x^3 + a_4 x + a_6$

This can be rewritten as:

$$y^2 = x^3 + ax + b \qquad \text{with } x, y, a, b \in \mathbb{R}$$

Equation 3-2

Each different choice of the parameters *a* and *b* results in a different elliptic curve. If the term $x^3 + ax + b$ contains no repeated factors, the elliptic curve $y^2 = x^3 + ax + b$ can be used to form a group. The term $x^3 + ax + b$ has no common factors, if the following is satisfied:

$$4a^3 + 27b^2 \neq 0$$

Equation 3-3

An elliptic curve over real numbers consists of the points on the corresponding elliptic curve equation, together with the point $\mathcal{O}$, which is called the point at infinity. As there are infinitely many points on an elliptic curve over real numbers it is unsuitable for cryptographic purposes. This is due to round of errors for example. For cryptographic purposes curves with discrete points are needed which allow for the wanted precision.

With the determined elliptic curve there are certain mathematical operations which can be defined on it. In this case it is addition and multiplication. This can be determined geometrically, but there are also algebraic equations which can be used. The geometric representation is only used here for clearification of the operations, but it is the algebraic representation which computer algorithms will depend upon. Whenever two points on a curve are added, the result will be either another point on the curve or the point O, called the point at infinity. The point at infinity is a special point that is defined especially to take care of cases when an addition would otherwise be undefined. All of the above explanations relate to elliptic curves over the field of real numbers. When elliptic curves are solved over other fields, the formulas used will change a little.

3.2 Arithmetic in Elliptic Curves over Real Numbers

Elliptic curves are additive groups which means that the basic function is addition. The addition of two points on an elliptic curve can be defined geometrically. One of the properties of elliptic curves is their symmetry along the x-axis. So given a point $P = (x_P, y_P)$ on the elliptic curve, the negative of this point is its reflection in the x-axis, i.e. $-P = (x_P, -y_P)$. For each point *P* on an elliptic curve, the point *–P* is also on the curve.

3.2.1 Adding two Distinct Points P and Q

Suppose that *P* and *Q* are two distinct points on an elliptic curve, and $P \neq -Q$. Then for the geometrical interpretation of adding the two points *P* and *Q*, a line is drawn through the two points which intersects the elliptic curve in exactly one more point, called *–R*. The point *–R* is reflected in the x-axis to obtain the point *R*. So the addition in an elliptic curve is defined as: $P + Q = R$

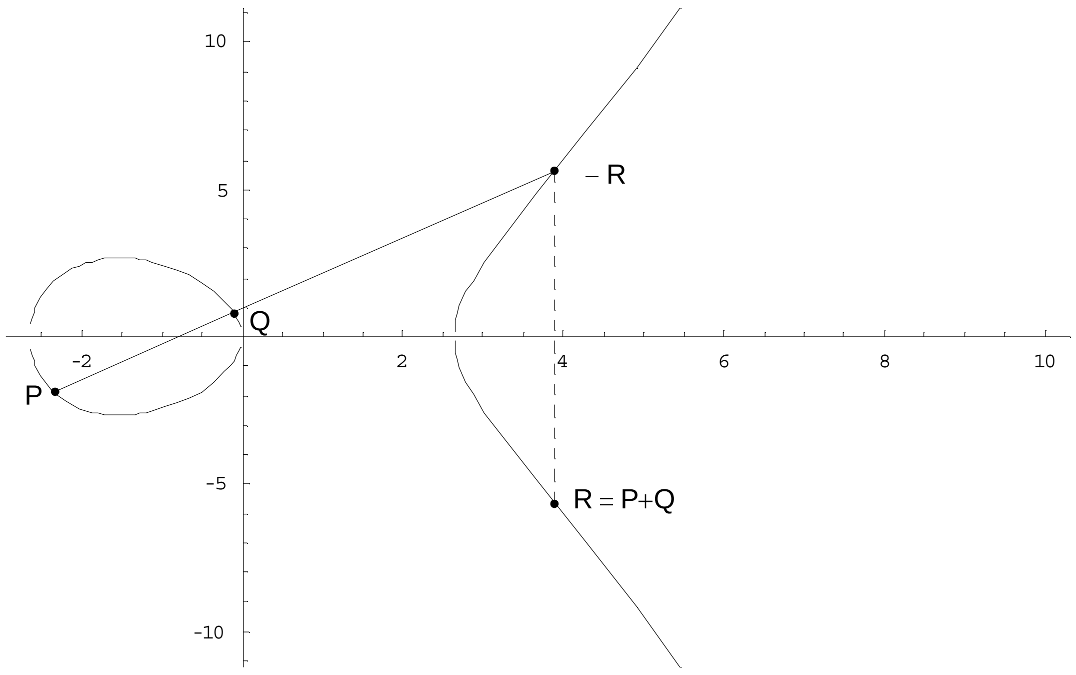


Figure 3-1

Figure 3-1 shows the geometrical addition of the two distinct points *P* and *Q* on and elliptic curve defined by the equation $y^2 = x^3 - 7x$.

3.2.2 Adding the Points P and –P

As the point *–P* is just the reflection of *P* in the x-axis as mentioned before, the line drawn through *P* and *–P* is a vertical line. It does not intersects the elliptic curve at a third point, which would make the addition of *P* and *–P* an undefined operation. To circumvent this, a special point is defined, the point at infinity. This point is also called the point *O*. Thus the elliptic curve consists of all points defined by the elliptic curve equation plus the point at infinity. So by definition the addtion of the points *P* and *–P* is:

$$P + (-P) = O$$

Equation 3-4

As a result, $P + O = P$ in the elliptic curve group. The point *O* is also called the additive identity of the elliptic curve group. All elliptic curves have such an additive identity.

3.2.3 Doubling of a Point P

To double the point *P*, it is simply added to itself. Geometrically this is done by drawing the tangent line to the elliptic curve at the point *P*. If $P = (x_P, y_P)$ and $y_P \neq 0$, then the line intersects the elliptic curve at exactly one more point, the point *–R*. The point *–R* is reflected in the x-axis to obtain *R*. So the doubling of point *P* is defined by:

$$2P = P + P = R$$

Equation 3-5

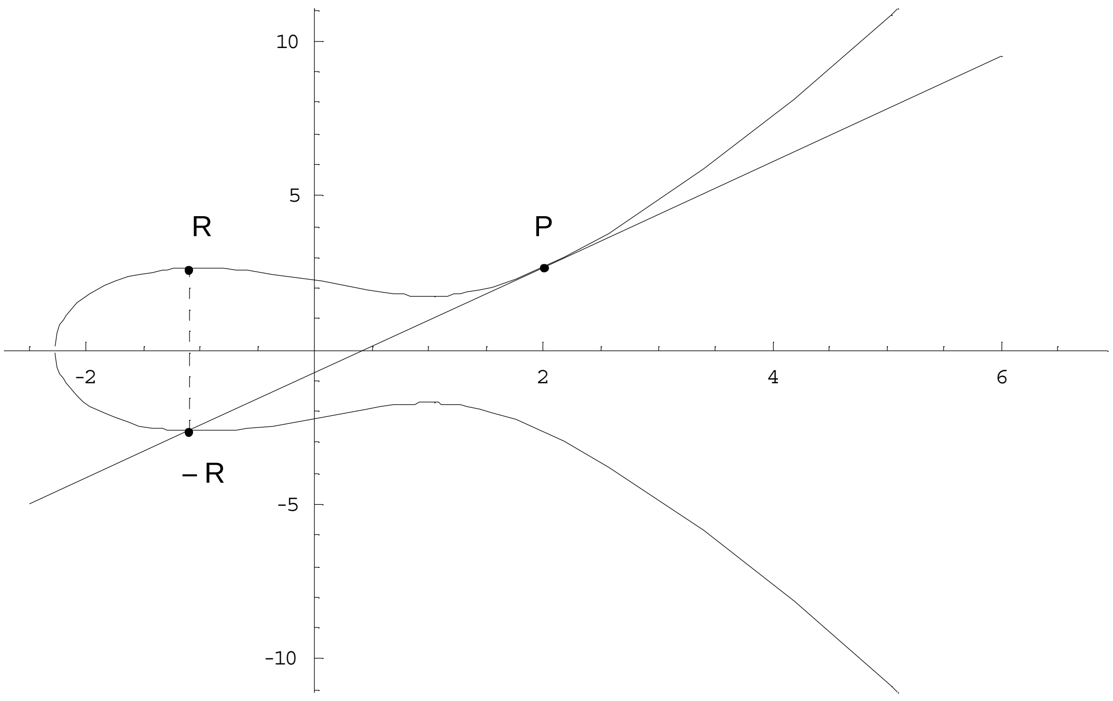


Figure 3-2

Figure 3-2 shows the doubling of point P on an elliptic curve given by $y^2 = x^3 - 3x + 5$. The tangent drawn at the point P intersects the elliptic curve in exactly one more point, the point $-R$. The reflection of $-R$ in the x-axis is the point $R = 2P$.

If the point P is such that $y_P = 0$, then the tangent line to the elliptic curve at the point P is vertical. So it does not intersect the curve at any other point. In this case, the doubling of P results in the point at infinity:

$$2P = O \qquad \text{for } P = (x_P, y_P = 0)$$

Equation 3-6

For finding 3P in this case P is added to the result of the doubling operation:

$$2P + P = O + P = P$$

Equation 3-7

This results in the following, given that $P = (x_P, y_P = 0)$:

$\Rightarrow xP = P$ for odd x; i.e. $x = \{1,3,5,7,...\}$

$\Rightarrow xP = O$ for even x; i.e. $x = \{2,4,6,8,...\}$

3.3 An Algebraic Approach

The geometric description of the operation in elliptic curves was to gain a better understanding and to illustrate the arithmetic. For the implementation of elliptic curve operations this is not practical, however. Algebraic equations are needed to compute the geometric arithmetic efficiently and precisely.

3.3.1 Adding two Distinct Points P and Q

If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, and none is the negative of the other, then:

$$P + Q = R$$

with $\lambda = \dfrac{y_P - y_Q}{x_P - x_Q}$ the slope of the line through *P* and *Q* if $P \neq Q$

and $R = (x_R, y_R)$ given by:

$$x_R = \lambda^2 - x_P - x_Q \quad \text{and} \quad y_R = \lambda(x_P - x_R) - y_P$$

Equation 3-8

3.3.2 Doubling of a Point P

If $P = (x_P, y_P)$ with $y_P \neq 0$

then $2P = R$

with $\lambda = \frac{3{x_P}^2 + a_4}{2y_P}$ the tangent in the point P and a_4 the parameter from the Weierstrass elliptic curve equation $y^2 = x^3 + a_4 x + a_6$.

The point $R = (x_R, y_R)$ is given by:

$x_R = \lambda^2 - 2x_P$ and $y_R = \lambda(x_P - x_R) - y_P$

Equation 3-9

Though very good for understanding the general arithmetic of elliptic curves, calculations over real numbers are not suitable for cryptography. Computing over real numbers normally is much slower than integer arithmetic. Further the calculations over real numbers are not very accurate due to round-off errors. What is needed for cryptography is a fast and precise arithmetic. For this reason, elliptic curve groups over the finite fields of $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ are used in cryptograhic practice. See 2.3.4 for details on finite fields.

3.4 Elliptic Curves over Finite Fields

In 2.3 the terms of field and finite field were discussed. The matter is briefly repeated here. A field is a set of numbers for which certain arithmetic operations, usually addition and subtraction, are defined. For the elliptic curve arithmetic discussed so far, only the infinite field of real numbers $\mathbb{R}$ (which contains all non-imaginary numbers) was of concern. This field is infinite in size, which means it has an infinite number of elements. Numerous other infinite fields exist, including whole numbers $\mathbb{Z}$, fractional numbers $\mathbb{Q}$, and imaginary numbers. As mentioned in section 3.1 infinite fields like the one over real numbers work poorly for cryptography because of infinitely repeating fractions and round-off errors, which is not acceptable for cryptographic purposes.

For cryptography, the curves should be defined over specific finite fields instead. A finite field, as the name says, is a field which is constrained and has a finite number of elements. The number of elements, though finite, can be a very large number. This depends on the field size and is indeed desired for cryptographic purposes. The finite fields which are common in the usage for elliptic curve cryptography are $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. $\mathbb{F}_p$ is the set of integer numbers less than some prime number *p*. $\mathbb{F}_{2^m}$ is a more complex field related to polynomials whose coefficients are all 0 or 1. It is said to have characteristic 2. The number *m* to define the field $\mathbb{F}_{2^m}$, can either be prime or composite. For *m* beeing composite a speedup in the calculations can be achieved.

However, for *m* composite the security is under question by cryptographic experts. The reason for that has to do with the existence of so called "non-trivial subfields" in *m* composite. These are subfields in the larger 2^m sized field which contain 2^n elements, for each divisor *n* of *m*. Thus the discrete logarithm problem for this type of fields reduces to the discrete logarithm problem in the smallest subfield. For this reason it is preferable to choose an *m* that is prime or at least an *m* composite with a small number of big factors when defining a field $\mathbb{F}_{2^m}$.

3.5 Elliptic Curves over the Field F$_P$

$\mathbb{F}_p$ is a finite field commonly used for elliptic curve cryptography. It is defined as the field of integer numbers *n*, $0 \le n \le p-1$, where *p* must be a prime number. Any calculations in $\mathbb{F}_p$ must be reduced modulo *p* so that the result of every field operation will still be in the field of $\mathbb{F}_p$.

As a result the elliptic curve equation (Equation 3-2) changes as follows:

$$y^2 \bmod p = (x^3 + ax + b) \bmod p \qquad \text{with } a, b \in \mathbb{F}_p$$

Equation 3-10

An elliptic curve with the underlying field of $\mathbb{F}_p$ can be formed by choosing the variables *a* and *b* within the field $\mathbb{F}_p$. The elliptic curve includes all points $P = (x, y)$ which satisfy the elliptic curve equation modulo *p* (where *x* and *y* are elements of $\mathbb{F}_p$) together with the point O, the point at infinity.

If $x^3 + ax + b$ contains no common factors, then the elliptic curve can be used to form a group. So likewise, to determine if the term $x^3 + ax + b$ has no common factors Equation 3-3 changes to:

$$(4a^3 + 27b^2) \bmod p \neq 0$$

Equation 3-11

Elliptic curves drawn in $\mathbb{F}_p$ look a lot different from those drawn in the field of real numbers. Instead of curves they only consist of disjoint points. It might not even be obvious how all the points connect together. The mathematical formulas for addition and multiplication still work, however. An elliptic curve over $\mathbb{F}_p$ consists only of a finite number of points, as Figure 3-3 shows for $p = 23$. The number of point on an elliptic curve is called the *order* or sometimes the *cardinality* of the curve. The order of an elliptic curve over a finite field must satisfy *Hasse's Theorem*. Given a field $\mathbb{F}_p$, the order of the curve N will satisfy the following equation:

$$\left| N - (p+1) \right| \leq 2\sqrt{p}$$

Equation 3-12

The *order of the curve* means the total number of points of the finite field, which satisfy the elliptic curve equation. The *order of a point* is defined as the number of times the point can be added to itself until the result is the point at infinity, O. The order of any point on a curve will evenly devide the order of that curve. This is a property which all abelian groups have (see 2.3.1 or the glossary for the definition of an abelian group).

As an example an elliptic curve over $\mathbb{F}_{23}$ is given by the equation $y^2 = x^3 + x$ (with $a = 1$ and $b = 0$). The term $(4a^3 + 27b^2) \bmod p \neq 0$ with the chosen *a* and *b* results in: $(4 + 0) \bmod 23 = 4 \neq 0$.

The elliptic curve equation $y^2 \bmod p = (x^3 + ax + b) \bmod p$ for this case becomes: $y^2 \bmod p = (x^3 + x) \bmod p$.

There are two points for every *x* value. In the field of real number the there were the two points $P = (x, y)$ and $P' = (x, -y)$. In the field $\mathbb{F}_p$ the values are taken modulo *p* which always results in a positive numbers less than *p*. So in the case of this example with $p = 23$ there is still a symmetry, which is shown in Figure 3-3 as a vertical line at $y = 11.5$.

For the *x* value of 11 the are two points $P = (11,10)$ and $P' = (11,13)$ which lie on the curve. To verify this, 11 is substituted for *x* in the right term of the equation:

$$(x^3 + x) \bmod p = (11^3 + 11) \bmod 23 = (1342) \bmod 23 = 8$$

So the y values must become:

$$y^2 \equiv 8(\mathrm{mod}\,23) \rightarrow \begin{cases} y = 10 \rightarrow 100 \equiv 8(\mathrm{mod}\,23) \\ y = 13 \rightarrow 169 \equiv 8(\mathrm{mod}\,23) \end{cases}$$

The 23 points which satisfy the equation $y^2 \equiv x^3 + x(\mathrm{mod}\,p)$ are:

(0, 0), (1, 5), (1, 18), (9, 5), (9, 18), (11, 10), (11, 13), (13, 5), (13, 18), (15, 3), (15, 20), (16, 8), (16, 15), (17, 10), (17, 13), (18, 10), (18, 13), (19, 1), (19, 22), (20, 4), (20, 19), (21, 6), (21, 17)

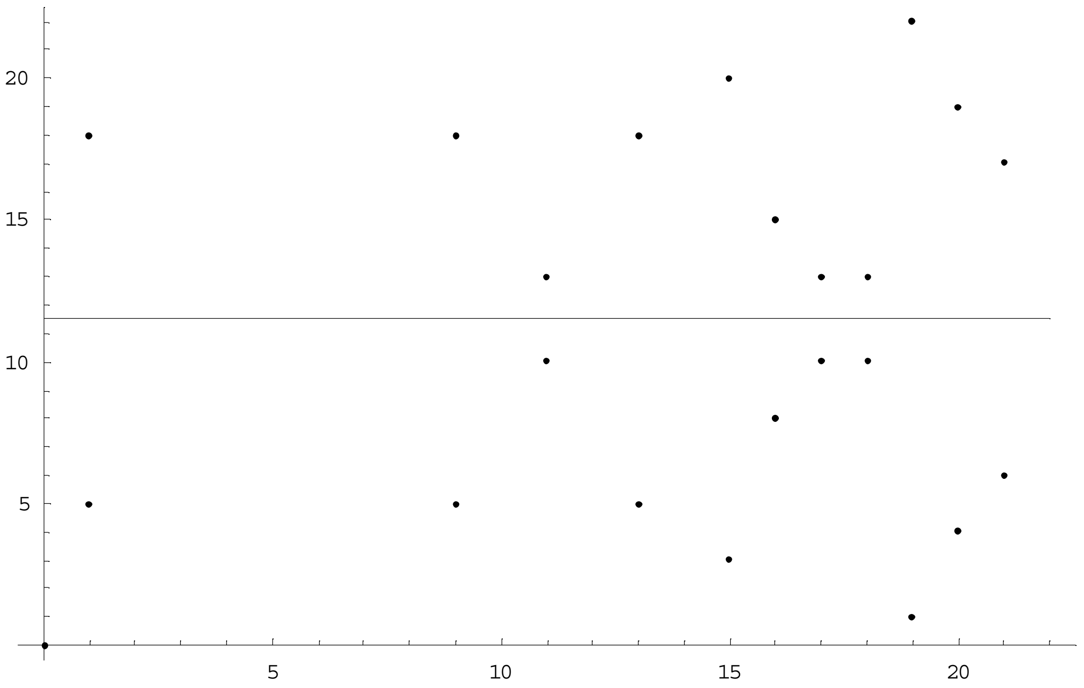


Figure 3-3

Figure 3-3 shows an example of an elliptic curve over $\mathbb{F}_{23}$. The elliptic curve equation is given by $y^2 \bmod 23 = (x^3 + x) \bmod 23$. The elliptic curve group of $y^2 = x^3 + x$ over the field $\mathbb{F}_{23}$ consists of 23 elements including the point at (0,0). A horizontal line is drawn in the diagram to show the symmetry in the x-axis at $y = 11.5$.

3.6 Arithmetic in Elliptic Curve over the Field F_p

There exist some differences in the arithmetic between elliptic curves over $\mathbb{F}_p$ and over the field of real numbers. Elliptic curves over $\mathbb{F}_p$ have a finite number of elements, as mentioned above. Since these curves consist only of discrete points, it is not obvious how to connect them to get a graph. Therefor it is not clear, how geometrical arithmetic can be applied. As a result, the geometric operations of real number elliptic curve groups cannot be used for elliptic curve groups over $\mathbb{F}_p$. However, the arithmetic rules can be adapted for the use in elliptic curves over the finite field $\mathbb{F}_p$.

3.6.1 Adding two Distinct Points P and Q

Suppose that P and Q are two distinct points on an elliptic curve, and $P \neq -Q$. The negativ of the point $P = (x_P, y_P)$ is defined as $-P = (x_P, -y_P \bmod p)$.

So if $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, and none is the negative of the other, then:

$$P + Q = R$$

with $\lambda = \dfrac{y_P - y_Q}{x_P - x_Q} \bmod p$ the slope of the line through P and Q

The point $R = (x_R, y_R)$ is given by:

$x_R = \lambda^2 - x_P - x_Q \bmod p$ and $y_R = \lambda(x_P - x_R) - y_P \bmod p$

Equation 3-13

3.6.2 Doubling of a Point P

If $y_p = 0$, then $2P = O$

If $y_P \neq 0$

$2P = R$

with $\lambda = \dfrac{3{x_P}^2 + a_4}{2y_P} \bmod p$

and a_4 the parameter from the Weierstrass elliptic curve equation $y^2 = x^3 + a_4x + a_6$.

The point $R = (x_R, y_R)$ is given by:

$x_R = \lambda^2 - 2x_P \bmod p$ and $y_R = \lambda(x_P - x_R) - y_P \bmod p$

Equation 3-14

The rules are exactly the same as for elliptic curve groups over real numbers, with the only exception that all results are reduced modulo the prime number *p*.

3.7 Elliptic Curves over the Field F_2^m

$\mathbb{F}_{2^m}$ is another finite field over which elliptic curves are commonly defined. Its definition is fairly complex. $\mathbb{F}_{2^m}$ consists of polynomials of degree less than *m*, whose coefficients fall into the field of $\mathbb{F}_2$. $\mathbb{F}_2$ is a field of the type $\mathbb{F}_p$ which was mentioned in 3.5. It consists of all integer numbers less than 2, i.e. 0 and 1. Any field of type $\mathbb{F}_{2^m}$ has exactly 2^m elements.

As the elements of $\mathbb{F}_{2^m}$ are polynomial with coefficients only 0 or 1, these elements can be represented as bit strings or binary vectors of length *m*. This property makes them very efficient for computer arithmetic. The rules for arithmetic in $\mathbb{F}_{2^m}$ can be defined in *polynomial basis* or *optimal normal basis* representation. This is discussed later in the implementation section.

An elliptic curve with the underlying field $\mathbb{F}_{2^m}$ is formed by choosing the parameters within $\mathbb{F}_{2^m}$. There are two possible versions of the Weierstrass equation for elliptic curves over the finite field $\mathbb{F}_{2^m}$:

$y^2 + xy = x^3 + a_2x^2 + a_6$ *non-supersingular* curve equation

$y^2 + y = x^3 + a_4x + a_6$ *supersingular* curve equation

The second form of the elliptic curve equation is called *supersingular*. These form has some advantages which makes computation faster. However, some special properties of supersingular curves make them unsuitable for cryptography. A prove of this fact is omitted here.

To keep a consistent notation the non-supersingular equation can be rewritten to:

$$y^2 + xy = x^3 + ax^2 + b \qquad \text{with } a, b, x, y \in \mathbb{F}_{2^m}$$

Equation 3-15

The parameters *a* and *b* must be chosen to lie within $\mathbb{F}_{2^m}$. The only constraint for the parameters is that *b* must not be zero, thus $b \neq 0$. However *a* may be zero. The elliptic curve includes all points $P = (x, y)$, with *x* and *y* being elements of $\mathbb{F}_{2^m}$, which satisfy the elliptic curve equation over $\mathbb{F}_{2^m}$, together with the point $\mathcal{O}$, the point at infinity.

As an example the field $\mathbb{F}_{2^4}$ is defined using the irreducible polynomial $f(x) = x^4 + x + 1$. For this field a generator α exists, which is a specific element of $\mathbb{F}_{2^4}$ that can be used to generate all of the other elements of the field by exponentiation. The generator α is given by the binary vector (0010) here.

Note that the generator risen to a specific power can also be used to represent any specific element in a field $\mathbb{F}_{2^m}$. When points are denoted in $\mathbb{F}_{2^m}$, they are usually given as powers of the generator α for the ease of writing. For this example the powers of $\alpha \in \mathbb{F}_{2^4}$ are:

power of α	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7
binary vector	(0001)	(0010)	(0100)	(1000)	(0011)	(0110)	(1100)	(1011)

power of α	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^{15}
binary vector	(0101)	(1010)	(0111)	(1110)	(1111)	(1101)	(1001)	(0001)

In a true application, the parameter *m* must be a large enough number to preclude the efficient generation of such a table. Otherwise the cryptosystem is not strong and can be easily broken. Nowadays, $m = 160$ bit would be a suitable choice to prevent the generation of the table of elements.

With the power of the generator α as a representation for each element of $\mathbb{F}_{2^m}$, the graphical representation could be drawn. The use of the power of α also allows for multiplication of elements without reference to the irreducable polynomial

$f(x) = x^4 + x + 1$. The difference in the calculations is that they tend to be much more complex, because of the fact that each element in $\mathbb{F}_{2^m}$ is a polynomial equation rather than a simple point. But because of the binary nature of those elements, computer arithmetic can be done quite efficiently.

For the example here the elliptic curve equation $y^2 + xy = x^3 + ax^2 + b$ is used. The parameters are chosen to be $a = \alpha^4 = (0011)$ and $b = \alpha^0 = (0001) = 1$.

The point $P = (\alpha^5, \alpha^3)$ satifies the elliptic curve equation and therefor lies on the curve, which can be easily verified:

$$
\begin{array}{l}
y^2 + xy = x^3 + \alpha^4 x^2 + 1 \\
(\alpha^3)^2 + \alpha^5 \alpha^3 = (\alpha^5)^3 + \alpha^4 (\alpha^5)^2 + 1 \\
\alpha^6 + \alpha^8 = \alpha^{15} + \alpha^{14} + 1 \\
(1100) + (0101) = (0001) + (1001) + (0001) \\
(1001) = (1001)
\end{array}
$$

The 15 points which satisfy the equation $y^2 + xy = x^3 + \alpha^4 x^2 + 1$ are:

$(1, \alpha^{13})$, (α^3, α^{13}), (α^5, α^{11}), (α^6, α^{14}), (α^9, α^{13}), (α^{10}, α^8), $(\alpha^{12}, \alpha^{12})$, $(1, \alpha^6)$, (α^3, α^8), (α^5, α^3), (α^6, α^8), (α^9, α^{10}), (α^{10}, α^1), $(\alpha^{12}, 0)$, $(0,1)$

3.8 Arithmetic in Elliptic Curves over the Field F$_2$^{m}

As for elliptic curves over $\mathbb{F}_p$ there exist some differences in the arithmetic, compared to elliptic curves over the field of real numbers. As for curves over $\mathbb{F}_p$ it is not clear, how geometrical arithmetic can be applied. So the geometric operations cannot be used for elliptic curve groups over $\mathbb{F}_{2^m}$. However, there exists an adapted set of arithmetic rules for elliptic curves over the finite field $\mathbb{F}_{2^m}$.

3.8.1 Adding two Distinct Points P and Q

Suppose that P and Q are two distinct points on an elliptic curve, and $P \neq -Q$. The negativ of the point $P = (x_P, y_P)$ is $-P = (x_P, x_P + y_P)$.

So if $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, and none is the negative of the other, then:

$$P + Q = R$$

with $\lambda = \dfrac{y_P - y_Q}{x_P - x_Q}$ the slope of the line through P and Q

Then the point $R = (x_R, y_R)$ is given by:

$$x_R = \lambda^2 + \lambda + x_P + x_Q + a_2 \quad \text{and} \quad y_R = \lambda(x_P + y_R) + x_R + y_P$$

Equation 3-16

As with elliptic curves over real numbers and over the finite field $\mathbb{F}_p$, $P+(-P)=O$, the point at infinity. Furthermore, $P+O=P$ for all points P in the elliptic curve group.

3.8.2 Doubling of a Point P

If $y_p = 0$, then $2P = O$

If $y_p \neq 0$

$2P = R$

with $\lambda = \dfrac{x_P + y_P}{x_P}$

The the point $R=(x_R, y_R)$ is given by:

$x_R = \lambda^2 + \lambda + a_2$ and $y_R = x_P + (\lambda+1)x_R$

Equation 3-17

With a_2 the parameter of the Weierstrass equation for non-supersingular curves.

3.9 Polynomial Basis and Optimal Normal Basis

To make things even more complex, elements in the field $\mathbb{F}_{2^m}$ can be represented in *polynomial basis* or *optimal normal basis*. A normal basis representation exists for all $m > 1$. Furthermore two different types of normal bases exist: referred to as Type I and Type II. This section tries to briefly describe this topic. There exists an enormous amount of literatur about normal basis representation. See [Blake et al 1999] for further details.

3.9.1 Polynomial Basis Representation

The elements of $\mathbb{F}_{2^m}$ are polynomials of degree less than *m*, with coefficients in $\mathbb{F}_2$; that is, $\{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_2x^2 + a_1x + a_0 \mid a_i \in \{0,1\}\}$. These elements can be written in vector form as $(a_{m-1}, a_{m-2}, ..., a_1, a_0)$. The finite field $\mathbb{F}_{2^m}$ has 2^m elements.

The main operations in $\mathbb{F}_{2^m}$ are addition and multiplication. Some computations involve a polynomial $f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + ... + f_2x^2 + f_1x + f_0$, where each f_i is in $\mathbb{F}_2$. The polynomial $f(x)$ must be irreducible; that is, it cannot be factored into two polynomials over $\mathbb{F}_2$, each of degree less than *m*.

Addition

$(a_{m-1},...,a_1,a_0)+(b_{m-1},...,b_1,b_0)=(c_{m-1},...,c_1,c_0)$ where each $c_i = a_i + b_i$ over $\mathbb{F}_2$. Therefore addition is just the componentwise XOR of $(a_{m-1},...,a_1,a_0)$ and $(b_{m-1},...,b_1,b_0)$.

Subtraction

In the field $\mathbb{F}_{2^m}$, each element $(a_{m-1},...,a_1,a_0)$ is its own additive inverse, since $(a_{m-1},...,a_1,a_0)+(a_{m-1},...,a_1,a_0)=(0,...,0,0)$, the additive identity. Thus addition and subtraction are equivalent operations in $\mathbb{F}_{2^m}$.

Multiplication

$(a_{m-1},...,a_1,a_0)(b_{m-1},...,b_1,b_0)=(r_{m-1},...,r_1,r_0)$ where $r_{m-1}x^{m-1}+...+r_1x+r_0$ is the remainder when the polynomial $(a_{m-1}x^{m-1}+...+a_1x+a_0)(b_{m-1}x^{m-1}+...+b_1x+b_0)$ is divided by the polynomial $f(x)$ over $\mathbb{F}_2$. Note that all polynomial coefficients are reduced modulo 2.

Exponentiation

The exponentiation $(a_{m-1},...,a_1,a_0)^k$ is performed by multiplying together k copies of $(a_{m-1},...,a_1,a_0)$.

Multiplicative Inversion

There exists at least one element α in $\mathbb{F}_{2^m}$ such that all non-zero elements in $\mathbb{F}_{2^m}$ can be expressed as a power of α. Such an element α is called a *generator* of $\mathbb{F}_{2^m}$. The multiplicative inverse of an element $x = \alpha^i$ is $x^{-1} = \alpha^{(-i) \bmod (2m-1)}$.

3.9.2 Example for Polynomial Basis

Given the field $\mathbb{F}_{2^4}$ with polynomial basis representation. The elements of $\mathbb{F}_{2^4}$ are the following 16 bit vectors of length 4:

(0000)	(0001)	(0010)	(0011)	(0100)	(0101)	(0110)	(0111)
(1000)	(1001)	(1010)	(1011)	(1100)	(1101)	(1110)	(1111)

The irreducible polynomial used will be $f(x) = x^4 + x + 1$. The following are sample calculations.

Addition

$(0110) + (0101) = (0011)$

Multiplication

$$
\begin{aligned}
&(1101)(1001) \\
&= (x^3 + x^2 + 1)(x^3 + 1) \bmod f(x) = (x^6 + x^5 + 2x^3 + x^2 + 1) \bmod f(x) \\
&= (x^6 + x^5 + x^2 + 1) \bmod f(x) \\
&= (x^4 + x + 1)(x^2 + x) + (x^3 + x^2 + x + 1) \bmod f(x) \\
&= x^3 + x^2 + x + 1 \\
&= (1111)
\end{aligned}
$$

Coefficients are reduced modulo 2. That is why $2x^3$vanishs.

Exponentiation

To compute$(0010)^5$, first find $(0010)^2$:

$$
\begin{aligned}
&(0010)(0010) \\
&= x^2 \bmod f(x) \\
&= x^2 \\
&= (0100)
\end{aligned}
$$

Then find $(0010)^4$:

$$
\begin{aligned}
&(0010)^2(0010)^2 \\
&= (0100)(0100) \\
&= x^4 \bmod f(x) \\
&= x + 1 \\
&= (0011)
\end{aligned}
$$

Finally, find $(0010)^5$:

$$\begin{aligned}
&(0010)^4(0010)\\
&=(0011)(0010)\\
&=(x^2+x)\bmod f(x)\\
&=x^2+x\\
&=(0110)
\end{aligned}$$

Multiplicative Inversion

The element $\alpha=(0010)$ is a generator for the field $\mathbb{F}_{2^4}$. The powers of α are listed in the example in 3.7 above.

The multiplicative identity for the field is $\alpha^0=(0001)$. The multiplicative inverse of $\alpha^7=(1011)$ is $\alpha^{-7 \bmod 15}=\alpha^{8 \bmod 15}=(0101)$.

To verify this, see that:

$$\begin{aligned}
&(1011)(0101)\\
&=(x^3+x+1)(x^2+1)\bmod f(x)\\
&=(x^5+x^2+x+1)\bmod f(x)\\
&=1\\
&=(0001)
\end{aligned}$$

which is the multiplicative identity.

3.9.3 Optimal Normal Basis Representation

For values of $m > 1$, the finite field $\mathbb{F}_{2^m}$ has an *optimal basis representation* as well as the polynomial representation described above. An optimal basis gives an alternative way of defining multiplication on the elements of a field. While optimal normal basis multiplication is less straight forward than polynomial multiplication, it is in practice much more efficient.

Optimal normal basis (ONB) representations are classified as either Type I or Type II. The value of *m* determines which type shall be used. For a Type I normal basis to exist, the following requirements must be fulfilled:

1. $m+1$ must be prime

2. 2 must be primitive in $\mathbb{Z}_{m+1}$; this means that 2 raised to any power in the range $[0, m-1]$ modulo $m+1$ must result in a unique integer in the range $[0, m]$.

For a Type II normal basis can be created when the following requirements are fulfilled:

1. $2m+1$ is prime

2.a 2 is primitive in $\mathbb{Z}_{2m+1}$; this means that 2 raised to any power in the range $[0, 2m-1]$ modulo $2m+1$ must result in a unique integer in the range $[0, 2m]$.

or

2.b $2m+1 \equiv 3 \bmod 4$ and 2 generates the quadratic residue in $\mathbb{Z}_{2m+1}$; this means that the last two bits of the binary representation of the prime $2m+1$ are set and that if 2 is not primitive in $\mathbb{Z}_{2m+1}$, it at least generates the quadratic residue in $\mathbb{Z}_{2m+1}$.

So the optimal normal basis setup is the following:

1. If $\mathbb{F}_{2^m}$ only has a type I ONB then let $f(x) = x^m + x^{m-1} + \ldots + x^2 + x + 1$. Otherwise, if $\mathbb{F}_{2^m}$ has a type II ONB then compute $f(x) = f_m(x)$ using the following recursive formulae:

$$\begin{aligned} &f_0(x) = 1 \\ &f_1(x) = x + 1 \\ &f_{i+1}(x) = x \cdot f_i(x) + f_{i-1}(x) \end{aligned}$$

with $1 \le i \le m$

At each stage, the coefficients of the polynomials $f_i(x)$ are reduced modulo 2; hence $f(x)$ is a polynomial of degree m with coefficients in $\mathbb{F}_2$. The set of polynomials $\{x, x^2, \ldots, x^{2m-1}\}$ forms a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$, called a normal basis.

2. Construct the m by m matrix A whose i^{th} row, $0 \le i \le m-1$, is the bit string corresponding to the polynomial $x^{2^i} \bmod f(x)$. The rows and columns of A are indexed by the integers from 0 to $m-1$. The entries of A are elements of $\mathbb{F}_2$.

3. Determine the inverse matrix A^{-1} of A over $\mathbb{F}_2$.

4. Construct m by m matrix T' whose i^{th} row, $0 \le i \le m-1$, is $x \cdot x^{2^i} \bmod f(x)$. Then compute the matrix $T = T' A^{-1}$ over $\mathbb{F}_2$.

5. Determine the product terms l_{ij}, for $i, j = [0 \ldots m-1]$, as $l_{ij} = T(j-i, -i)$. $T(g,h)$ denotes the (g,h)-entry of T with indices reduced modulo *m*. Each product term l_{ij} is an element of $\mathbb{F}_2$. It should also be the case that $l_{0j} = 1$ for precisely one *j*, $0 \le j \le m-1$, and that for each *i*, $0 \le i \le m-1$, $l_{ij} = 1$ for precisely two distinct *j*, $0 \le j \le m-1$. Hence only $2m-1$ of the m^2 entries of the matrix *T* are 1, the rest being 0. This scarcity of 1's is the reason that the normal basis is called an *optimal normal basis.*

3.9.4 Example for Optimal Normal Basis

Given the field $\mathbb{F}_{2^4}$ with optimal normal basis (ONB) representation. The elements of $\mathbb{F}_{2^4}$ are all binary vectors of length 4. Addition and subtraction are defined as with polynomial basis representation.

1. The field $\mathbb{F}_{2^4}$ has a Type I optimal normal basis; thus $f(x) = x^4 + x^3 + x^2 + x + 1$. The set of polynomials $\{x, x^2, x^4, x^8\}$ forms a normal basis of $\mathbb{F}_{2^4}$ over $\mathbb{F}_2$.

2. The rows of A are constructed as follows:

   Row 0: $x \bmod f(x) = x = (0010)$

   Row 1: $x^2 \bmod f(x) = x^2 = (0100)$

   Row 2: $x^4 \bmod f(x) = x^3 + x^2 + x + 1 = (1111)$

   Row 3: $x^8 \bmod f(x) = x^3 = (1000)$

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

3. And the inverse of A over $\mathbb{F}_2$ is:

$$A^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

4. The rows of T' are constructed as follows:

Row 0: $v = x \cdot x \bmod f(x) = x^2 = (0100)$

Row 1: $v = x \cdot x^2 \bmod f(x) = x^3 = (1000)$

Row 2: $v = x \cdot x^4 \bmod f(x) = x^5 = 1 = (0001)$

Row 3: $v = x \cdot x^8 \bmod f(x) = x^9 = x^3 + x^2 + x + 1 = (1111)$

$$T' = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

5. $$T = T' \cdot A^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The product terms are:

$l_{0,0} = T(0,0) = 0$	$l_{1,0} = T(3,3) = 0$	$l_{2,0} = T(2,2) = 1$	$l_{3,0} = T(1,1) = 0$
$l_{0,1} = T(1,0) = 0$	$l_{1,1} = T(0,3) = 0$	$l_{2,1} = T(3,2) = 1$	$l_{3,1} = T(2,1) = 1$
$l_{0,2} = T(2,0) = 1$	$l_{1,2} = T(1,3) = 1$	$l_{2,2} = T(0,2) = 0$	$l_{3,2} = T(3,1) = 0$
$l_{0,3} = T(3,0) = 0$	$l_{1,3} = T(2,3) = 1$	$l_{2,3} = T(1,2) = 0$	$l_{3,3} = T(0,1) = 1$

Multiplication

Multiplication is defined by $(a_0, a_1, a_2, a_3)(b_0, b_1, b_2, b_3) = (c_0, c_1, c_2, c_3)$, where:

$$c_0 = a_0 b_2 + a_1(b_2 + b_3) + a_2(b_0 + b_1) + a_3(b_1 + b_3)$$

$$c_1 = a_1 b_3 + a_2(b_3 + b_0) + a_3(b_1 + b_2) + a_0(b_2 + b_0)$$

$$c_2 = a_2 b_0 + a_3(b_0 + b_1) + a_0(b_2 + b_3) + a_1(b_3 + b_1)$$

$$c_3 = a_3 b_1 + a_0(b_1 + b_2) + a_1(b_3 + b_0) + a_2(b_0 + b_2)$$

Thus $(0100)(1101) = (c_0, c_1, c_2, c_3)$, where:

$$c_0 = 0(0) + 1(0+1) + 0(1+1) + 0(1+1) = 1$$

$$c_1 = 1(1) + 0(1+1) + 0(1+0) + 0(0+1) = 1$$

$$c_2 = 0(1) + 0(1+1) + 0(0+1) + 1(1+1) = 0$$

$$c_3 = 0(1) + 0(1+0) + 1(1+1) + 0(1+0) = 0$$

and finally: $(0100)(1101) = (1100)$.

Exponentiation using Optimal Normal Bases

The squaring $(a_0, a_1, a_2, a_3)^2 = (a_0, a_1, a_2, a_3)(a_0, a_1, a_2, a_3) = (c_0, c_1, c_2, c_3)$, where:

$$c_0 = a_0 a_2 + a_1(a_2 + a_3) + a_2(a_0 + a_1) + a_3(a_1 + a_3) = a_3^{\ 2} = a_3$$

$$c_1 = a_1 a_3 + a_2(a_3 + a_0) + a_3(a_1 + a_2) + a_0(a_2 + a_0) = a_0^{\ 2} = a_0$$

$$c_2 = a_2 a_0 + a_3(a_0 + a_1) + a_0(a_2 + a_3) + a_1(a_3 + a_1) = a_1^{\ 2} = a_1$$

$$c_3 = a_3 a_1 + a_0(a_1 + a_2) + a_1(a_3 + a_0) + a_2(a_0 + a_2) = a_2^{\ 2} = a_2$$

Thus squaring $(a_0, a_1, a_2, a_3)^2 = (a_3, a_0, a_1, a_2)$ can be calculated with a simple rightwise rotation of (a_0, a_1, a_2, a_3).

Squaring is a very efficient operation when optimal normal basis representation is used. Since exponentiation typically involves many squaring operations, exponentiation is performed far more efficiently using optimal normal basis representation than using polynomial representation.

4 Elliptic Curve Cryptography

4.1 The Elliptic Curve Discrete Logarithm Problem

Defining curves and fields is not sufficient without having a problem to solve. In the case of cryptography a trap-door or one-way function is needed so that it is easy to encrypt data, but difficult to decrypt it without the knowledge of a specific secret.

The Elliptic Curve Discrete Logarithm Problem (ECDLP) is based upon the Discrete Logarithm Problem (DLP) which is the basis of the Digital Signature Algorithm (DSA) and the ElGamal cryptosystem. Because of the use of elliptic curves groups, the ECDLP offers the advantage of smaller keys, however. The ECDLP is built upon the difficulty to reverse scalar multiplication, which is a type of multiplication defined over a elliptic curve. The defined operations over elliptic curve groups described in this paper were point adding and point doubling. A point P on an elliptic curve can be doubled to obtain $2P$. After that addition of P to the point $2P$ results in the point $3P$ and so on. The determination of a point kP (where k is an integer) in this manner is refered to as scalar multiplication of a point P. The problem to reverse this scalar multiplication is written as:

Given $kP = Q$, and given P and Q, find k. Both P and Q are points on the elliptic curve, while k is some large integer number, typically consisting of 160 bits or more.

As mentioned in 2.4.2 the discrete logarithm problem in the multiplicative group $\mathbb{Z}_p^*$ is the following:

Given a prime *p*, the elements α, β with α a generator and β an arbitrary element of $\mathbb{Z}_p^*$, find the integer *k*, $0 \le k \le p-2$, such that $\alpha^k \equiv \beta (\bmod p)$.

In the elliptic curve group the discrete logarithm problem can be statet as:

Given points *P* and *Q* which are elements of the group, find the integer number *k* such that $P^k = Q$. So *k* is called the discrete logarithm of *Q* to the base *P*.

When the elliptic curve group is defined using additive notation, the elliptic curve discrete logarithm problem is:

Given points *P* and *Q*, which are elements of the group, find a integer *k* such that $kP = Q$.

4.2 Elliptic Curve Group Operations

The multiplicative group of a finite field, the finite abelian group $\mathbb{F}_q^*$, can be used to build public key cryptosystems. The aim is to create cryptosystems based on elliptic curves, that is the finite abelian group of an elliptic curve E defined over $\mathbb{F}_q$. Elliptic curve public key cryptosystems are based on the intractability of the discrete logarithm problem in this group, which is the elliptic curve discrete logarithm problem (ECDLP) as mentioned in 4.1.

The elliptic curve analogy of multiplying two elements of $\mathbb{F}_q^*$ is adding two points on E, where E is the elliptic curve over some finite field $\mathbb{F}_q$. Thus the analogy of raising to the k'th power in $\mathbb{F}_q^*$ is multiplication of a point $P \in E$ by an integer k. The multiple $kP \in E$ can be found very efficiently by repeated doubling and adding. To find the point 100P, this can be written as $100P = 2(2(P + 2(2(2(P + 2P)))))$. This results in 6 point doubling operations and 2 point addition operations.

4.3 Imbedding Plaintext

To encrypt or sign plaintext it must be represented as elements of the group in which the arithmetic is done. So in the case of an elliptic curve cryptosystem the data must be encoded as points on some elliptic curve *E* defined over a finite field $\mathbb{F}_q$ (where $\mathbb{F}_q$ can be either $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$). This should be done in a simple and systematic way, so that the plaintext *m* (which can be regarded as an integer in some range) can be readily determined from the knowledge of the coordinates of the corresponding point P_m. This process in usually called *encoding*. Note that *encoding* is not the same as *encryption*. The point P_m, which can be referred to as the message point, can then be transformed into a point P_e using some encryption algorithm defined over the elliptic curve. So P_e is the ciphertext point. An authorised user of the cryptosystem must be able to recover the message *m* from the point P_m after deciphering the ciphertext point.

According to [Koblitz 1994] there is no polynomial time (in $\log q$) *deterministic* algorithm known for writing down a large number of points on an arbitrary elliptic curve *E* over $\mathbb{F}_q$. However, there are probabilistic algorithms for which the chance of failure is very small. Further it is not enough to generate random points of the elliptic curve *E*. In order to encode a large number of plaintext messages *m*, a systematic way is needed to generate points that are related to *m* in some way. This can be the relationship of only the x-coordinate to the integer *m*, for example. To be able to imbed plaintext in the point coordinates, there must be a number of *don't care bits,* so the fields size should be somewhat bigger than the units of data to imbed. A probabilistic method to do this is given in [Koblitz 1994]:

Given an elliptic curve *E* over $\mathbb{F}_q$, where $q = 2^r$ is assumed to be large and odd. Let κ be a large enough integer so that the failure probability will be 1 out of 2^κ when a plaintext unit *m* is tried to imbed. In practice it may be assumed that $30 \le \kappa \le 50$ should suffice. It is further assumed that the message units *m* are integers with $0 \le m < M$. It is also supposed that the finite field is chosen so that $q > M\kappa$. The integers are written from 1 to $M\kappa$ in the form $m\kappa + j$, where $1 \le j \le \kappa$, and a 1-to-1 correspondence is set up between such integers and a set of elements of $\mathbb{F}_q$. For example, such an integer is written as an *r*-digit integer to the base *p*. The *r* digits, considered as elements of $\mathbb{Z}/p\mathbb{Z}$, can be taken as coeffcients of a polynomial of degree *r*-1 corresponding to an element of $\mathbb{F}_q$. That is, the integer $(a_{r-1}, a_{r-2}, ..., a_1, a_0)_p$ corresponds to the polynomial $\sum_{j=0}^{r-1} a_j X^j$, which, considered modulo some fixed degree-*r* irreducible polynomial over $\mathbb{F}_p$, gives an element of $\mathbb{F}_q$.

Thus, given *m,* for each $j = 1,2,...,\kappa$ an element *x* of $\mathbb{F}_q$ can be obtained which correspopnds to $m\kappa + j$. For such an *x*, the right side of the equation $y^2 = f(x) = x^3 + ax + b$ is computed and tried to find a square root of $f(x)$. If a *y* is found such that $y^2 = f(x)$, then $P_m = (x, y)$ is a point on the given curve. If it turns out the $f(x)$ is a nonsquare, then *j* is incremented by 1 and tried again with the corresponding *x*. Provided an *x* can be found for which $f(x)$ is a square, before *j* gets bigger than κ, then *m* can be recovered from the point $P_m = (x, y)$ by $m = [(\overline{x} - 1)/\kappa]$, where $\overline{x}$ is the integer corresponding to *x* under the 1-to-1 correspondence between integers and elements of $\mathbb{F}_q$. Since $f(x)$ is a square for approximately 50% of all *x*, there is only about a $2^{-\kappa}$ probability that the method will fail to produce a point P_m whose x-coordinates correspond to an integer $\overline{x}$ between $m\kappa + 1$ and $m\kappa + \kappa$. To be precise, the probability that $f(x)$ is a square is essentially equal to $N/2q$, which is very close to $1/2$.

4.4 Elliptic Curve ElGamal

As mentioned in 2.5.4 the generalised version of the ElGamal public key encryption and signature scheme can be defined over the group of points on an elliptic curve *E* defined over a finite field $\mathbb{F}_q$. The advantage of the ElGamal scheme is that the order of the group, that is the number of points of the elliptic curve in this case, is not needed to be known.

Given a fixed publicly known finite field $\mathbb{F}_q$ and an elliptic curve *E* defined over it. Given further and a base point $B \in E$. Each entity chooses a random integer *a*, which must be kept secret, and computes a point aB which is published as the public key.

So for some entity A, the private key would be the integer a_A and the public key would be the point $a_A B$. Note that the finite field $\mathbb{F}_q$, the chosen elliptic curve *E* defined over that field and the base point $B \in E$ must be known to each communicating party or must be published with the public key.

If entity B wishes to encrypt a message *m* for A, B must do the following:

1. Obtain the (authentic) public key of A, which is the point $a_A B$.

2. Represent the message *m* as an point $P_m \in E$.

3. Select a random integer *k*.

4. Compute the points kB and $P_m + k(a_A B)$

5. Send the pair of points $c = \{kB, P_m + k(a_A B)\}$ as the ciphertext to A.

To recover the plaintext *m* from the ciphertext *c*, A must do the following:

1. Use the private key a_A to compute $a_A(kB)$ from the first point.

2. Then take the second point and compute $P_m + k(a_A B) - a_A(kB) = P_m$ which recovers the plaintext *m* from the point $P_m \in E$.

Thus B sends a encryption of the point P_m, which is $P_m + k(a_A B)$, together with some *clue* how to decrypt it, which is kB. If one knows the private key a_A, the computation of $a_A kB$ and therefor the recovery of P_m is easy. An eavesdropper will only learn the points kB and $P_m + k(a_A B)$. However, if an eavesdropper can solve the discrete logarithm problem on *E,* the private key a_A can be retrieved from the publicly known information *B* and $a_A B$.

4.5 Elliptic Curve DSA

There is also an elliptic curve version of the DSA. As mentioned in 2.6 the generalised version of the DSA signature scheme can be (as is the case for ElGamal) defined over the group of points on an elliptic curve *E* defined over a finite field $\mathbb{F}_p$. The elliptic curve version of DSA is also mentioned in the IEEE P1363 standard draft which proposes methods for elliptic curve cryptography. It is called ECDSA.

Given a fixed publicly known finite field $\mathbb{F}_p$ and an elliptic curve *E* defined over it. Given further and a base point $B \in E$ with order *n*. Each entity chooses a random integer *a*, which must be kept secret, and computes a point aB which is published as the public key.

So, equally to elliptic curve ElGamal, for some entity A, the private key would be the integer a_A and the public key would be the point $a_A B$. Note that the finite field $\mathbb{F}_p$, the chosen elliptic curve *E* defined over that field and the base point $B \in E$ with order *n* must be known to each communicating party or must be published with the public key.

To generate a key pair, entity A must do the following:

1. Select an elliptic curve *E* defined over the finite field $\mathbb{F}_p$. The number of points of *E*, i.e. the order of the curve should be devisible by a large prime *n*.

2. Select a point $B \in E$ of order *n*.

3. Select a unique and unpredictable integer *a* at random with $1 \le a \le n-1$.

4. Compute the point $Q = aB$

5. A's public key is (*E, B, n, Q*); A's private key is *a*.

If entity A wishes to sign a message *m*, A must do the following:

1. Select a unique and unpredictable random integer *k* with $1 \le k \le n-1$ and compute the point $R = kB$.

2. Use hash function $h(m)$ with $h(m) < n$ to calculate the hash for the message *m*. The integer *n* is the order of the base point *B*.

3. Compute $r = x_R \bmod n$ with x_R the x component of point $R = (x_R, y_R) = kB$. If $r = 0$ then start again with step 1. This is because of the next operation requires $r \neq 0$ or the private key would not be involved in the signature calculation!

4. Use the private key *a* to compute $s = k^{-1}(h(m) + ar) \bmod n$. $h(m)$ is the hash algorithm. For the DSA standard, it is required to be the SHA-1 secure hash algorithm! If $s = 0$ start again with step 1. For the case $s = 0$ there exists no $k^{-1} \bmod n$ which is required in the signature verification process.

5. Send the pair $\{r, s\}$ as the signature together with the message *m*.

To verify the signature for *m*, B must do the following:

1. Obtain the (authentic) public key of A, which is (*E*, *B*, *n*, *Q*). Verify that the signature values *r* and *s* are integers in the interval $[1, n-1]$.

2. Use the hash function, called $h'(m)$ here with $h'(m) < n$ to calculate the hash for the message *m*. Note that $h'(m) = h(m)$; the different notation is chosen just for telling apart the operations of A and B.

3. Compute $w = s^{-1} \bmod n$

4. Compute $u_1 = (h'(m) \cdot w) \bmod n$ and $u_2 = rw \bmod n$

5. Compute the point $R' = u_1 B + u_2 Q = (x_{R'}, y_{R'})$ and $v = x_{R'} \bmod n$

5. The signature is valid if and only if $v = r$.

As can be seen the only significant difference between the ECSA and DSA is the generation of the signature value *r*. The DSA does this by taking a random element $\alpha^k \bmod p$ with α a generator of the cyclic group $\mathbb{Z}_p^*$ and reducing it modulo *q*. Thus a integer in the interval $[1, q-1]$ is obtained. The ECDSA generates the integer *r* in the interval $[1, n-1]$ by taking the x coordinate of the random point $R = kB$ and reducing it modulo *n*.

4.6 Computational Complexity

In this section it is tried to compare elliptic curve cryptography with coventional public key cryptography in terms of computational complexity. To be able to compare the security of conventional DLP based public key cryptosystems with elliptic curve cryptosystems the computational complexity of the DLP in both systems must be determined. The intention is to make an statement about the practical implications of using the group of point on an suitable chosen elliptic curve defined over some finite field $\mathbb{F}_q$, which is noted $E(\mathbb{F}_q)$ (where q can be either a prime number p or the binary polynomial of grade 2^m), instead of the conventional multiplicative group $\mathbb{F}_p^*$.

For a well chosen curve (a curve with good cryptographic properties), the best known method for solving the DLP on $E(\mathbb{F}_q)$ is of complexity exponential in the size $n = \lceil \log_2 q \rceil$ of the field elements. For the DLP in $\mathbb{F}_p^*$ there are algorithms available that are sub-exponential in $N = \lceil \log_2 p \rceil$. So the best known algorithms for the elliptic curve DLP are of complexity proportional to:

$$C_{EC}(n) = 2^{n/2}$$

Define the function $L_p(v,c) = \exp(c(\log p)^v (\log\log p)^{(1-v)})$ where log without a base specification denotes the real natural logarithm. When $v = 1$, the function L_p is exponential in $\log p$, while for $v = 0$ it is polynomial in $\log p$. When $0 < v < 1$, the behaviour is strictly between polynomial and exponential, and is referred to as *sub-exponential*.

Discrete logarithms in $\mathbb{F}_p$ can be found in time proportional to $L_p(\frac{1}{3}, c_0)$ where $c_0 = (\frac{64}{9})^{1/3} \approx 1.92$, using a general number field sieve method. In terms of *N*, and neglecting constant factors, the complexity is:

$$C_{CONV}(N) = \exp(c_0 N^{1/3} (\log(N \log 2))^{2/3)}$$

Here C_{CONV} stands for the complexity function for conventional public key cryptography. It is worth noting that the best known algorithms for *integer factorisation* are of roughly the same asymptotic complexity. This makes the discussion and comparision even applicable to public key cryptosystems based on the integer factoring problem, e.g. the RSA algorithm.

Equating C_{EC} and C_{CONV} neglecting constant factors in the complexities it follows that for similar levels of security:

$n = \beta \cdot N^{1/3} (\log(N \log 2))^{2/3}$, where $\beta = 2c_0 / (\log 2)^{2/3} \approx 4.91$

The parameters *n* and *N* in the equation above can be interpreted as the key sizes in bits for the respective cryptosystems. Therefore, with today‘s algorithmic knowledge, the key size in an elliptic curve cryptosystem grows slightly faster than the cube root of the corresponding key size of the conventional public key cryptosystem for similar cryptographic strenght (or computational complexity).

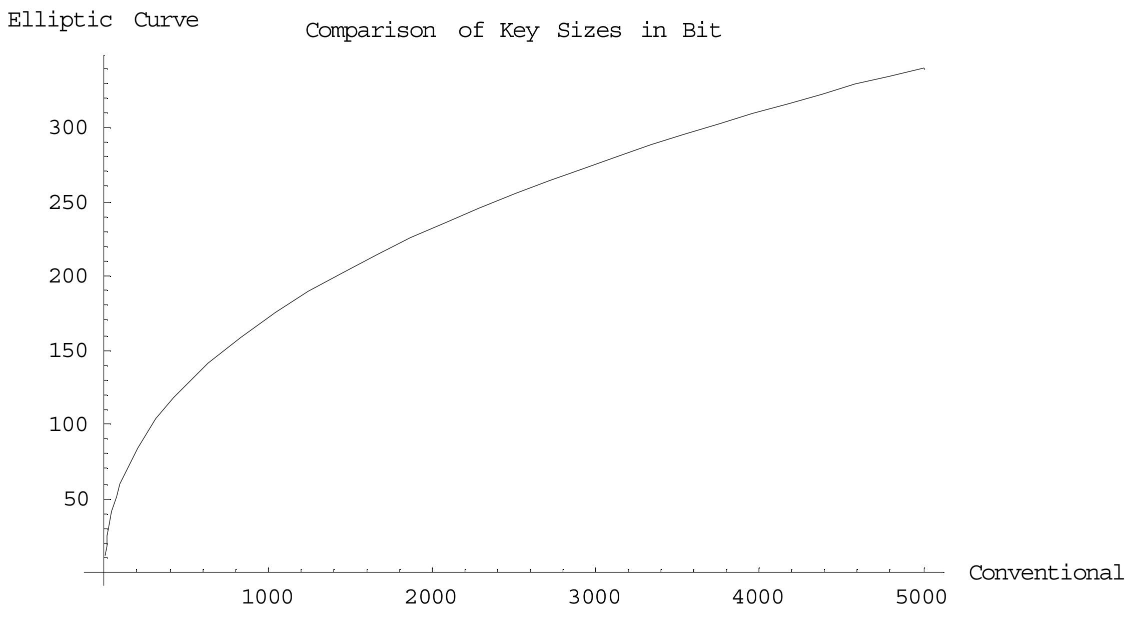


Figure 4-1

The relation can be plotted in a diagram as shown in Figure 4-1. Common key sizes for DSA are 512 and 1024 bit and a further possible key size for RSA is 2048 bit. This translates into elliptic curve cryptosystem key sizes of 128 bit, 174 bit and 234 bit respectively. Given that various approximations have been used, and various constants neglegted, such figures are approximate and give only general trends. A fair comparison must also take into account the complexity of implementing the cryptosystem. While implementation of group exponentiation is about the same complexity in both cases, in terms of elementary group operations, the group operations themselves are more complex in the elliptic curve case for the same field size. Nevertheless, the plot helps to explain the recent interest in elliptic curve cryptography as an alternative to conventional crypto systems. In practice it can be assumed that shorter key length translates in possible faster implementation, less power consumption (e.g. for mobile devices) and less usage of computer memory.

5 Design and Implementation

The aim of this thesis‘ work was to implement a cryptographic scheme based on the arithmetic of elliptic curves in the Java programming language. Therefore an object oriented approach is adopted for the design and implementation of the software. As the Java programming language (or the JDK) provides an architecture for cryptographic functionality, the software should be designed according to that architecture and build in a way to seamless fit into the provided framework.

5.1 The Java Cryptography Architecture (JCA)

The cryptographic functionality in Java is separated in the JCA, the Java Cryptography Architecture and the JCE, the Java Cryptography Extension. The splitting has mainly to do with U.S. export restrictions for cryptographic products. While the JCA provides the general framework for the cryptographic core classes in Java, the JCE provides additional support for encryption, the cipher engine classes and some concrete cipher implementations with an extended security provider. Note that secure hash algorithms, pure signature schemes (which can not be used to encrypt data) and algorithms for secure pseudo random number generation are not subject of U.S. export restricitions. Only cryptographic algorithms which provide strong data encryption functionality are put under export control and usually are not shipped by U.S. software vendors.

The concept of security providers was introduced in the JDK 1.1 and allows for multiple and interoperable cryptography implementations from different vendors. The Java 2 platform (JDK 1.2 and later) significantly extended the JCA. The international available versions of the JDK and JRE still ship without the JCE. This means that without a third party implementation of the JCE functionality there is no support for data encryption and ciphers.

The JCA can be described as a provider architecture. The primary principal in the design of the JCA has been to separate the cryptographic concepts from their algorithmic implementations. It is designed to allow different vendors to provide their own implementation of the cryptographic tools and other administrative functions. This makes it a very flexible framework which is open to future requirements and allows for vendor independence.

The architecture defines a series of classes, called engine classes. The engine classes are representations of general cryptographic functions. So, for example, there are several different standards for digital signatures, which differ in their detailed implementation but which, at a high level, are very similar. For this reason, a single engine class, `java.security.Signature`, has been created that represents all of the variations in a digital signature. The actual implementation of the different signature algorithms is done by a provider class which may be offered by a number of vendors.

Figure 5-1 shows a schematic of the JCA. The providers with their implementation of particular algorithms must be registered in the runtime environment. The user's (or programmer's) code does not refer to the algorithms directly. Instead an secondary layer of abstraction is used to instanciate a particular object. E.g. a signature object is created in the following way:

```
Signature sig = Signature.getInstance("Algorithm","Provider");
```

The parameter `Provider` is optional. The call returns a `Signature` object, if the given algorithm is implemented by at least one registered provider. If no provider is specified, an available implementation of the first provider in the list of installed providers is taken. However, a particular provider may be given as a parameter (as shown above), which causes the system to return the implementation of this specified provider. An Exception is thrown in case that the algorithm is not implemented by any provider in the system (if no specific provider was given) or it is not available from the explicitly specified provider.

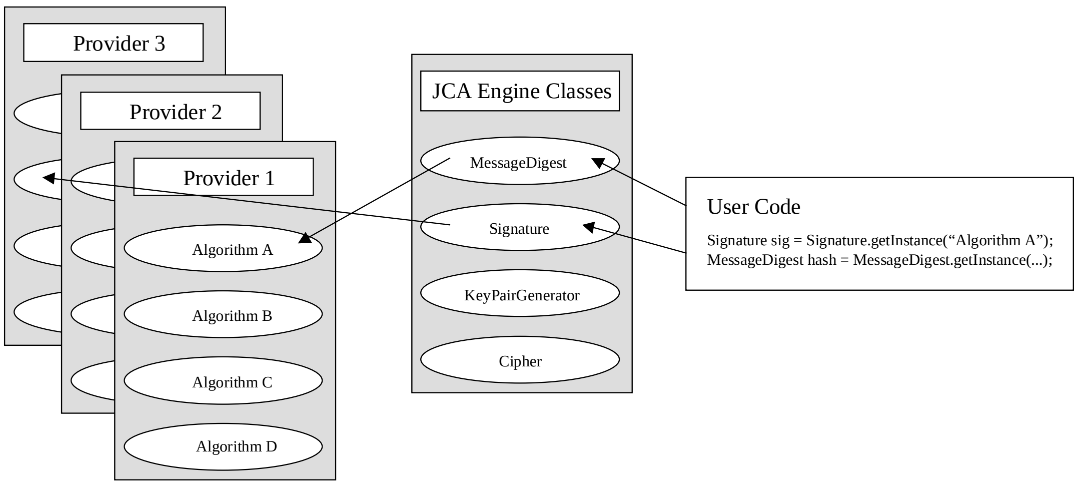


Figure 5-1

Currently, Sun Microsystems' version of the JDK and JRE come with a default provider, named SUN. Other Java Runtime Environments (JREs) may not necessarily supply the SUN provider. The SUN provider includes an implementation of the following algorithms:

- Digital Signature Algorithm (DSA)
- SHA-1 and MD5 message digest (also known as secure hash) algorithms

- A SHA-1 based pseudo-random number generation algorithm

Just to complete the picture, the SUN provider implements a DSA *key factory*, a *certificate factory* for X.509v3 certificates and certificate revocation list (CRLs), and a *keystore* implementation for the proprietary keystore type named Java Keystore (JKS).

5.2 Requirements

The following requirement are given for the system:

1. The implemented elliptic curve algorithm must fit in the Java Cryptography Architecture. Thus, a own provider for the JCA should be implemented, which is fully compliant and implements the needed algorithm.

2. It should be ensured, that the software runs with general available components, i.e. the JDK (Java Development Kit) or JRE (Java Runtime Environment) from Sun Microsystems, which can be obtained for free in the standard version.

   As there is no support for encryption in the international version of Sun Microsystems‘ JDK or JRE, a scheme for digital signatures based on elliptic curves should be implemented. This ensures that it runs without a JCE, which otherwise must be obtained from a third party vendor.

3. The implemented algorithm should allow for benchmarking against an available algorithm which is comparable to the elliptic curve variant on the algorithmic level, i.e. is similar except for the finite field used. This means comparing ElGamal with the elliptic curve version of ElGamal (ECElGamal) or DSA with ECDSA.

5.3 Choice of an Algorithm

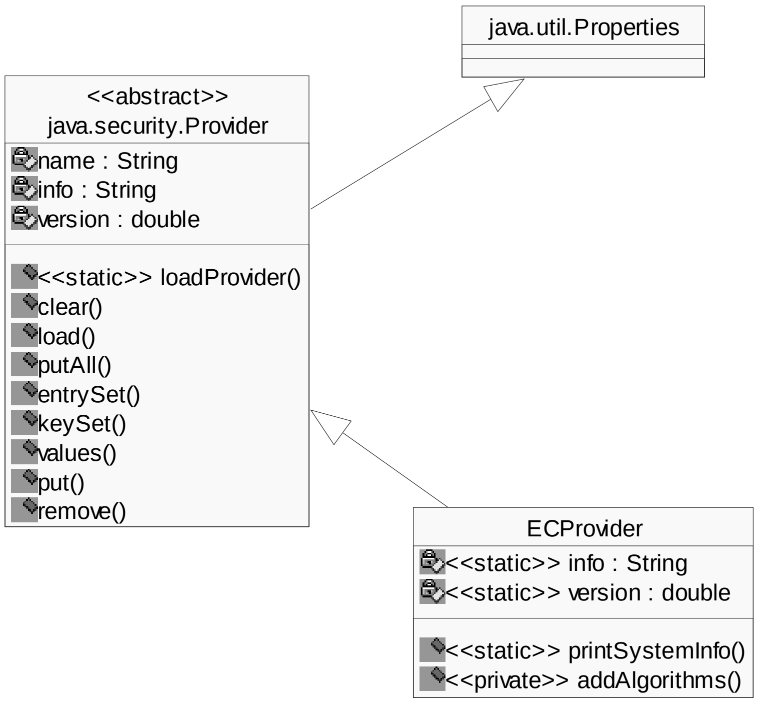
Because of the availability of the SHA-1 secure hash function and the DSA signature algorithm in the SUN provider, the choice for an elliptic curve algorithm to implement fell on the specific variant of DSA, called ECDSA. The SHA-1 hash algorithm is needed for both variants of DSA and the standard version of the DSA algorithm is ready to use with the availability of Sun's JDK or JRE. As mentioned earlier, the standard version of the JDK and JRE from Sun come with the SUN provider and can be obtained for free. Further this allows for the direct comparison of the DSA and the ECDSA on the algorithmic level for benchmarking. The implementation of a public key encryption scheme like ElGamal seemed not feasible as it would have required a JCE which is not part of the international version of the Sun Microsystems' JDK. An own implementation of the JCE has been considered, but was rejected because of the heavy implementation effort which would have been infeasible in the given time. With an available JCE the implementation of the elliptic curve ElGamal encryption scheme would have been easily possible as the mathematical and algorithmic background is already described in this document. As the order of an elliptic curve must not be known for the EC ElGamal algorithm, the implementation effort would have been even less than for ECDSA.

5.4 Design

The design of the components was mainly driven by the given JCA framework. The core classes which must be available are the specific *provider* class implementation and the classes for the *signature* and *keypair generator*.

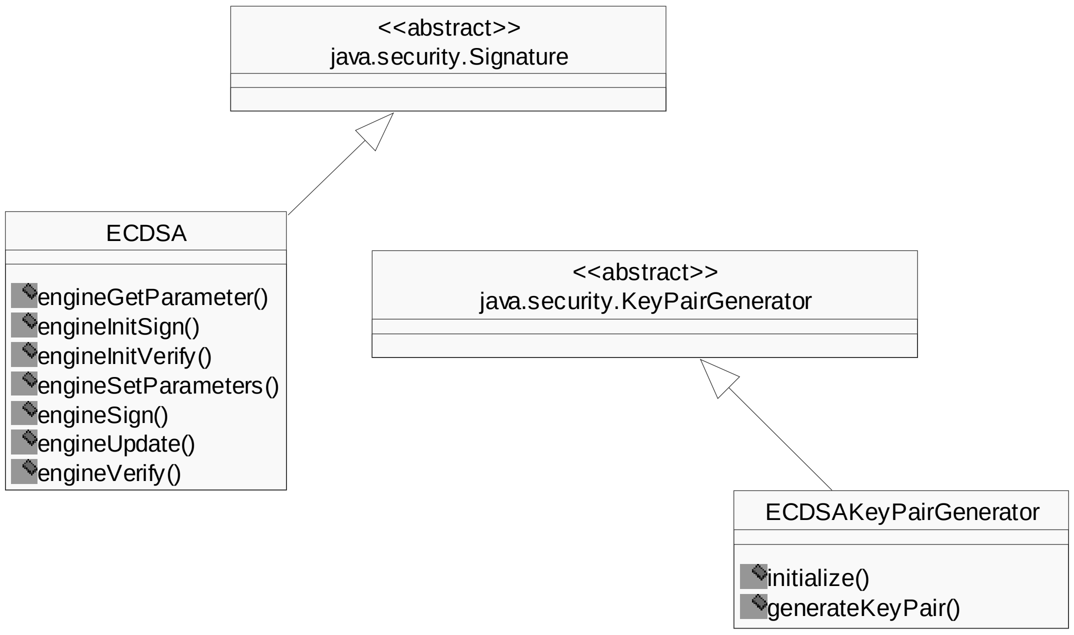
5.4.1 The Provider Class

The specific `ECProvider` class extends the abstract class `java.security.Provider` of the JCA framework. It is just a container for some information about the algorithms which are implemented by this provider.

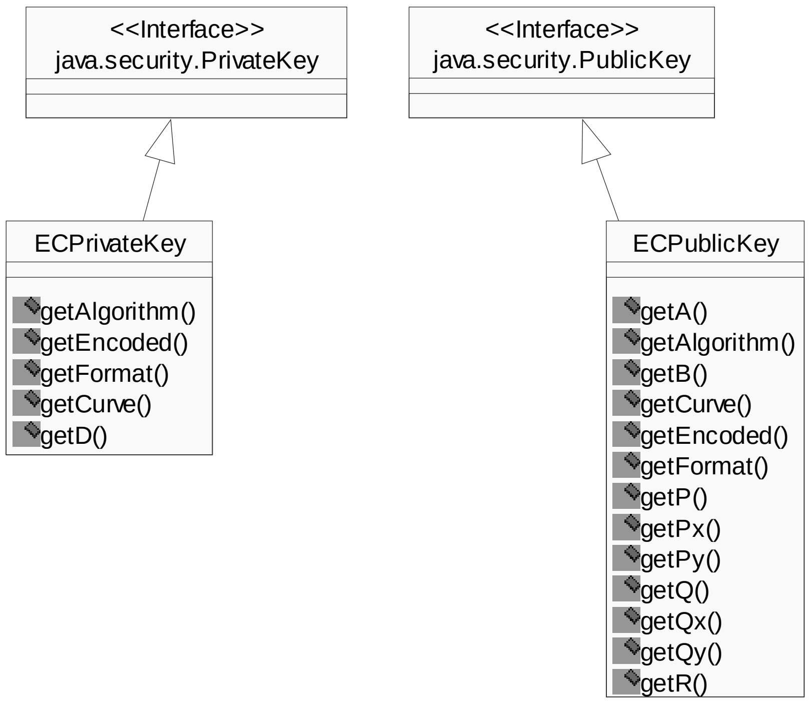


5.4.2 Signature and KeyPairGenerator

For the signature scheme the abstract classes `java.security.Signature` and `java.security.KeyPairGenerator` of the JCA must be extended and filled with life.



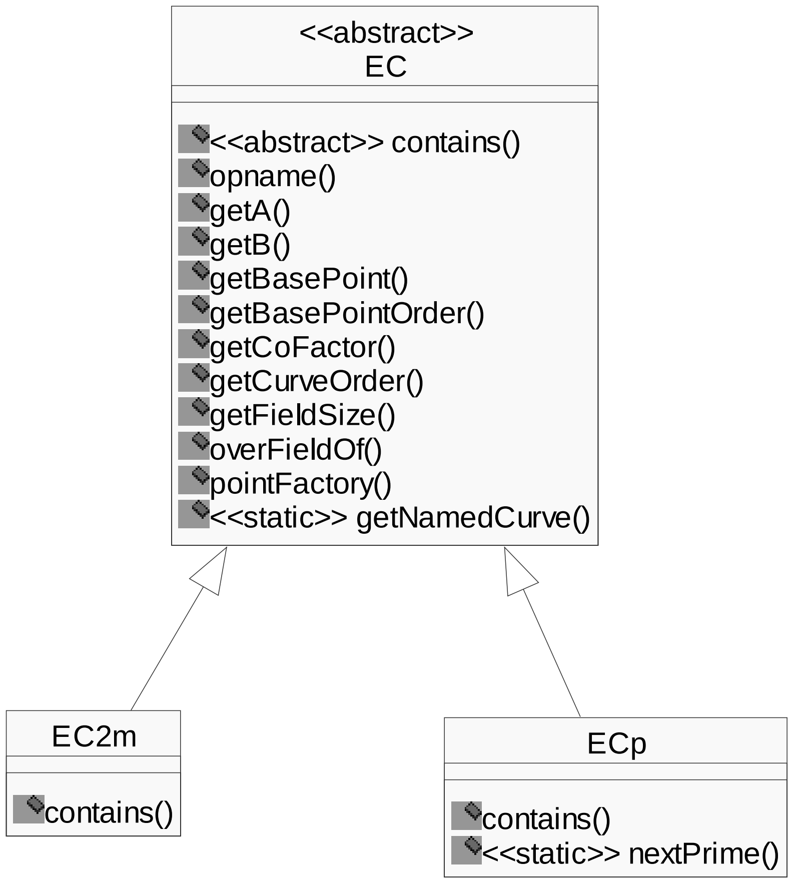
The `KeyPairGenerator` class provides the functionality of generation of the public and private key for the specific public key cryptoscheme. A call of the method `generateKeyPair()` returns a `KeyPair` object, which itself is just a container for a `PrivateKey` and `PublicKey` object. The specific versions of the private key and public key must implement the interfaces `java.security.PrivateKey` and `java.security.PublicKey` respectively.



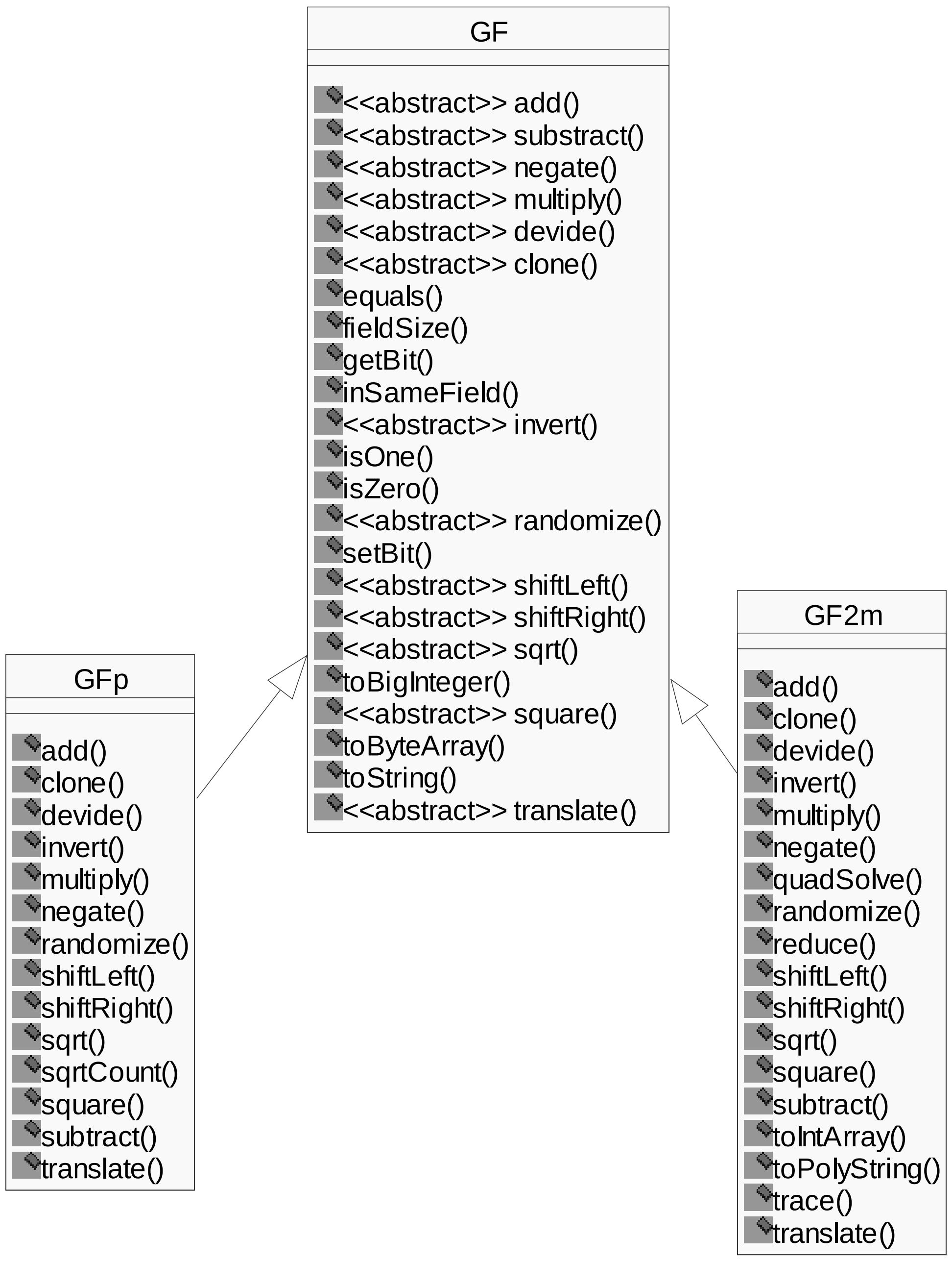
5.4.3 Further Classes

The design so far was driven by the given context, the Java Cryptography Architecture. Further classes for the modelling of elliptic curves, points on an elliptic curve and Galois fields are specific for this implementation. Furthermore some additionally helper classes are defined. There are two types of elliptic curves considered here: curves defined over the finite field $\mathbb{F}_p$ and curves defined over the finite field $\mathbb{F}_{2^m}$. To take care of this fact, there is one base class which defines members and methods which are equal for both types of curves. Methods which are specific to one particular type are left abstract and are implemented in the extending class.

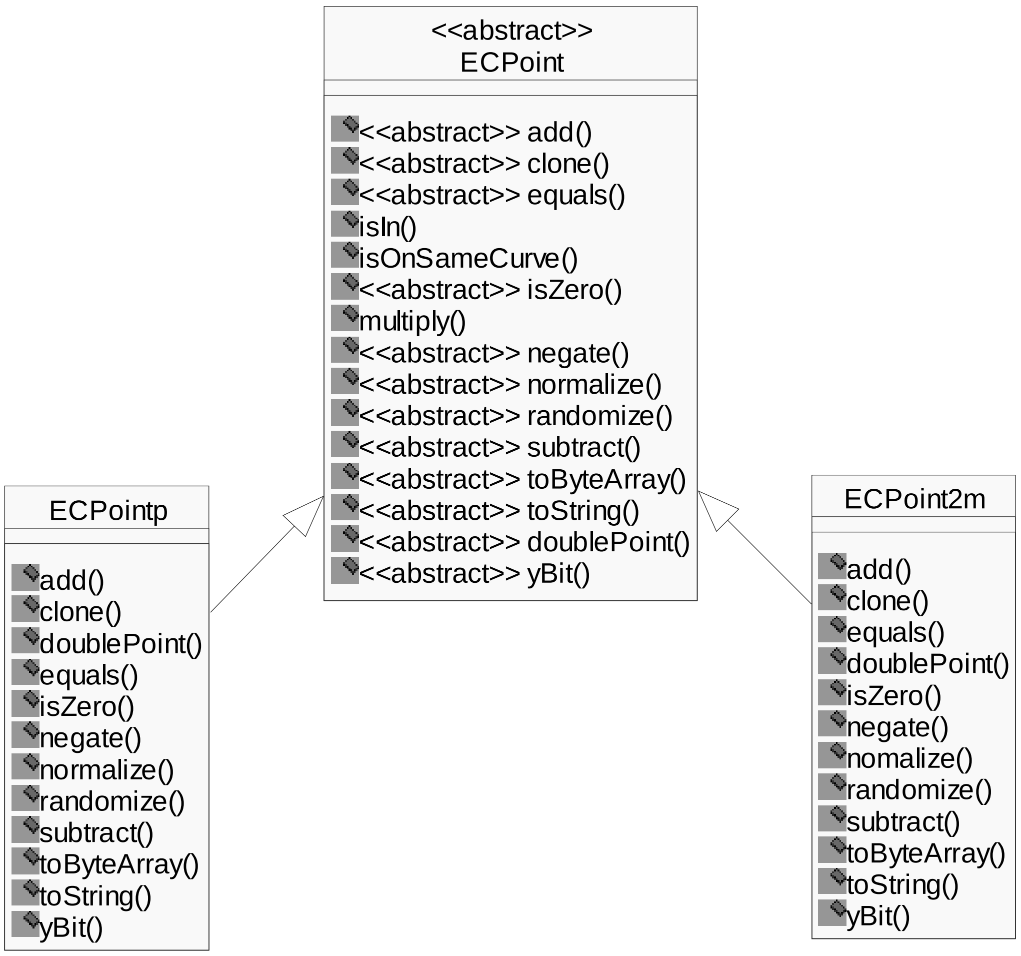
This results in the following class hierarchy for the elliptic curve implementation. The base class is named `EC`:



The finite field elements and operations defined for these elements are modelled by the base class `GF` (which stands for Galois field). Subclasses for the particular fields $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ are the classes `GFp` and `GF2m` respectively.



The points on a specific elliptic curve are modelled using the base class `ECPoint` and the extending classes `ECPointp` and `ECPoint2m`.



5.5 Implementation Issues

5.5.1 Choice of the Basis

The choice of the basis representation can be either polynomial or normal (ONB). The tradeoff between these is a tradeoff between speed and space. Polynomial basis implementation is considered to be faster in point addition and multiplication. But it takes twice as much space, because multiplication doubles the number of bits stored before being reduced modulo the basis function. Optimal normal basis uses less space but takes longer to compute the point addition and multiplication operations. The inversion operation for field elements is slower in polynomial basis than in optimal normal basis. A combination of both would optimise the operations but makes implementation more complicated.

The implementation of ONB algorithms takes a lot of time effort. So for this project there is only polynomial basis representation implemented at the moment. This is slower than it could be with ONB. Furthermore shift and rotate operations on a bit basis are not very efficiently to implement in Java. This is believed to be mainly due to the abstraction of the virtual maschine (VM).

5.5.2 Finding suitable Elliptic Curves

Finding good elliptic curves at random is not easy. One definite rule is to avoid supersingular curves, as mentioned elsewhere in this paper. In these curves the discrete logarithm problem can be reduced to the discrete logarithm problem over an extension field K of small degree.

As the number of points on a curve determines the security level, it is useful to know this number of points (the order) for a generated curve. Furthermore for some cryptographic protocols (e.g. DSA) the order of the curve must definitely be known. Calculating the order for a random elliptic curve is quite difficult and very time consuming.

Three main techniques are presently known to determine elliptic curves which are suitable for cryptography [Blake et al 1999]:

1. Generate random curves and compute their group orders, until an appropriate curve is found.

2. Generate curves with given group order using the theory of complex multiplication (CM). Such curves are usually called CM-curves.

3. Use the group of $\mathbb{F}_{q^n}$-rational points, $E(\mathbb{F}_{q^n})$, of a curve E defined over $\mathbb{F}_q$, for q relatively small. Taking a view centered on the field over which the rational points of interest are defined, the curve in this case is often referred to as a *subfield curve*, or a curve of *Koblitz type* (also known as *Koblitz curve*).

The CM method is considered less computationally taxing than general point counting. In case both CM and point counting on an random elliptic curve are considered too complex, a reasonable compromise can be found in the use of subfield curves.

As state already, it is very difficult and time consuming to find a curve with good cryptographic properties at random. Furthermore the algorithms are very hard to implement. However, for the concrete implementation it is possible to take curves which are already known to have good cryptographic properties and for which the curve order is already determined. Such curves can be found in the literature about the subject [Blake et al 1999, Koblitz 1992]. For algorithms as EC ElGamal, the order of the curve doesn't matter. If not a very high security level is needed here, it is sufficient to choose a curve at random without knowledge of its order. In the case of the ECDSA algorithm, the curve order must be known to make the algorithm work. Here it is appropriate to take a known curve as basis for the cryptosystem, as generating a good random curve and determining the curve order might be too time consuming.

5.5.3 Finding a Base Point at Random

To find a point at random which is a generator of a given elliptic curve is not trivial. One method to guarantee that the choice of the point P_B is suitable - and in fact generates the elliptic curve - is to choose the curve and finite field so that the number N of points is itself a prime number. If so, then every point $P_B \neq O$ will be a generator of the curve. This, for a given fixed $\mathbb{F}_q$, requires to choose pairs (E, P_B) until one is found for which the number of points on *E* is a prime number (which can be determined by the known primality tests). It is not clear how long this will take, so an assumption about the running time of the algorithms is hard to make.

5.6 Performance Evaluation

The performance figures were taken on a Pentium III-600 Mhz with 512 MB of random access memory, running the Windows 2000 operating system. The DSA implementation is the one which comes with Sun Microsystems‘ JDK and JRE. The possible key length for DSA is specified to be between 512 bit and 1024 bit and to be a multiple of 64 bit. So for the benchmarking, three key sizes were chosen: 512 bit, 768 bit and 1024 bit.

The maximum time precision of the Java VM from Sun running on a Microsoft Windows platform is about 10 ms. This is a real drawback and made absolute performance measures for standard DSA somewhat useless. However, the performance measures related to ECDSA have still some value in direct comparison. The performance values might seem to be with 1 ms precision as there are values of e.g. 21 ms. This is due to round-off errors. However, Sun‘s specification of the VM for Microsoft Windows operating systems states a maximum accuracy of 10 ms for time measurement.

5.6.1 DSA Signature Scheme

512 bit key

Key generation time (ms)	5047	10	0	10	10	10	0	10	10	10
Signing time (ms)	10	10	10	0	10	10	0	10	10	0
Verification time (ms)	20	20	20	20	10	20	10	20	20	10

768 bit key

Key generation time (ms)	5297	10	20	21	10	20	20	10	20	20
Signing time (ms)	20	20	10	20	20	10	20	20	10	20
Verification time (ms)	30	30	40	30	30	40	30	40	40	40

1024 bit key

Key generation time (ms)	5067	20	30	40	30	30	40	30	30	20
Signing time (ms)	20	30	30	30	30	20	20	30	30	30
Verification time (ms)	60	50	50	50	60	51	60	50	50	60

Worth noting is the first measured value for each key generation. It tends to be very high, compared to the rest of the values. This stems from the setup of the internal secure random number generator. The first time seed is automatically generated from a number of system parameters (see Sun's spec for details of the algorithm) which takes some time. After the inital setup, the generation of pseudo random numbers is less time consuming, as it just follows a mathematical function.

5.6.2 ECDSA Signature Scheme

There are no specifical restrictions in the key length for elliptic curve DSA. However, to be sure to have an security level which equals the key size, the size of the field (the key length) is chosen to be a prime number of bits. Note that 131 bit ECDSA is comparable to 512 bit in DSA from a security point of view. This means that the computational complexity of the Discrete Logarithm Problem is equal in both systems for the given key length. As it implies from the section about the computational complexity of elliptic curve cryptography versus conventional public key schemes, 163 bit ECDSA can be compared to DSA with a key length somewhere between 768 bit and 1024 bit. Further 197 bit ECDSA gives a security level which is somewhat greater than 1024 bit DSA. The measure for 239 bit ECDSA was made out of curiosity. It is especially interesting because of the a complexity level which could be compared to (a fictitious) 2048 bit DSA.

97 bit key

Key generation time (ms)	5487	330	1452	1342	931	421	561	4537	3906	410
Signing time (ms)	71	60	51	50	40	40	40	50	50	61
Verification time (ms)	100	100	110	110	110	100	90	120	100	100

131 bit key

Key generation time (ms)	22642	17014	661	1222	9193	5948	10715	8622	831	962
Signing time (ms)	90	90	110	90	80	90	90	111	100	90
Verification time (ms)	191	181	190	190	170	191	190	170	180	190

163 bit key

Key generation time (ms)	13329	32226	14401	3856	12147	1382	7831	1712	941	27039
Signing time (ms)	140	150	150	150	140	160	151	161	131	170
Verification time (ms)	291	270	280	281	281	280	290	270	280	301

197 bit key

Key generation time (ms)	6019	1051	2985	11907	3905	74497	2563	2664	9614	1142
Signing time (ms)	240	221	200	220	190	221	211	210	210	220
Verification time (ms)	421	450	401	421	411	430	420	440	421	411

239 bit key

Key generation time (ms)	11807	2223	32347	76099	10956	7481	3415	2633	17405	26068
Signing time (ms)	321	3210	330	331	320	360	331	331	330	320
Verification time (ms)	671	651	641	671	631	651	621	631	661	641

The most noteworthy result is the time for key generation. The time to generate a key pair may vary a lot even for the same key size, depending on the chosen elliptic curve. As the elliptic curve is chosen at random, the number of points on this curve must be calculated for the ECDSA algorithm to work. The calculation of the curve order and base point is a quite time consuming process and may even result in the rejection of the chosen curve. The time for determining the curve order depends on the field size, but not entirely. So for a field size of 97 bit the best value for key pair genration was 330 ms, compared to the worst with 5487 ms. This can be also compared to a 239 bit field size, where the best case for a row of measured values was 2223 ms (which is less than half of the time taken for the worst case in the 97 bit field) and the worst case was 76099 ms. Significant improval in time for the key generation process may be obtained by the use of known curves with good security properties (e.g. the so called *Koblitz Curves* which were found by Neal Kobliz and are published in [Koblitz 1992]). For those known curves, the curve order is already determined.

6 Conclusion

To quote the mathematician Serge Lang, *“It is possible to write endlessly about elliptic curves”*. And indeed there is an enormous amount of literature on the subject. Let alone the other aspects of applications for elliptic curves in number theory, factoring and algebraic geometry, the field of cryptographic application is huge and complex. To cover it as a whole would have required to write a book and possibly more than one. This paper tries to explain parts of the subject to an extend which should be sufficient to get an principal understanding of the subject and the implementation of elliptic curve cryptography.

Elliptic curve cryptography is still in its infany. Even known for quite some time now, the subject is still a field with a lot of research going on. Standards for elliptic curve cryptography are under development as the proposed IEEE P1363. The possiblity of shorter keys for a given security level allows for more efficient implementation in environments where computer memory is limited. However, the Java programming language might not be the best choice for the efficient implementation of elliptic curve algebra. This is mainly due to the impossibility of the efficient implementation of bit operations, as shift and rotate for example. A better choice for an efficient implementation of elliptic curve cryptography might be the C programming language, with critical parts of the software (e.g. elliptic curve point inversion) even better written in the assembly language.

To be fair it should be noted that the existing implementation is far from optimal. A lot of operations are very fast in optimal normal basis representation (ONB) for elliptic curve elements, others are faster in polynomial basis. ONB implementation is considered a little faster over all. Time was to short to fully implement functional ONB arithmetic. So only polynomial basis arithmetic could be used for the performance benchmarking. So with the possibility of ONB arithmetic and some other algorithmic enhancements proposed in the actual literature there is quite a potential for performance improvement.

There was a lot of research in the recent years in this particular area of cryptography, which is still going on. And as recent publications have shown, there is still a lot of room for improventments in the algorithms for elliptic curve cryptography. One problem left is the enormous complexity of the used algorithms, which in fact might have prevented elliptic curves to be used widely in cryptography today. It is far easier to implement the RSA scheme or ElGamal and DSA over $\mathbb{Z}_p^*$ than it is to implement the same using elliptic curves over $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$. So the advantage of shorter key length for the same security level today is bought by a very high implementation effort and higher complexity and longer run time of the algorithms.

But there is application for elliptic curve cryptography, even today. Mainly hand held devices as mobile phones, PDAs or smart cards are very limited in the use of computer memory. Here, the use of smaller key size without loosing security is a real advantage. And with research going on we might reasonably expect faster algorithms and easier implementation in the coming years.

7 Glossary

Abelian group	An arithmetic operation is said to be commutative if the order of its arguments is insignificant. With ordinary numbers, addition and multiplication are commutative operations; for example, $2 \cdot 9 = 9 \cdot 2$ and $2 + 9 = 9 + 2$. However, subtraction and division are not commutative since $2 - 9 \neq 9 - 2$ and $2/9 \neq 9/2$. A group is called *abelian* if its main operation is commutative. Thus an additive group is abelian if $a + b = b + a$ for all elements *a*, *b* in the group. A multiplicative group is abelian if $ab = ba$ for all *a*, *b* in the group. The additive group $\mathbb{Z}_n$ and the multiplicative group $\mathbb{Z}_p^*$ are both *abelian groups*.
Adaptive chosen ciphertext attack	A version of the *chosen ciphertext attack* where the cryptanalyst can choose ciphertexts dynamically. A cryptanalyst can mount an attack of this type in a scenario in which he or she has free use of a piece of decryption hardware, but is unable to extract the decryption key from it.

Adaptive chosen plaintext attack	A special case of the *chosen plaintext attack* in which the cryptanalyst is able to choose plaintexts dynamically, and alter his or her choices based on the results of previous encryptions.
Attack	Either a successful or unsuccessful attempt of breaking a part of a cryptosystem or the cryptosystem as a whole. See *birthday attack*, *brute force attack*, *chosen ciphertext attack*, *chosen plaintext attack*, *dictionary attack*, *known plaintext attack*.
Authentication	The action of verifying information such as identity, ownership or authorisation.
Birthday attack	A brute force attack used to find collisions. It got its name from the surprising result that the probability of two or more people in a group of 23 sharing the same birthday is greater than 0.5.
Brute force attack	This attack requires trying all (or a large fraction of all) possible values till the right value is found; also called an *exhaustive search*.

Chosen ciphertext attack	An attack where the cryptanalyst may choose the ciphertext to be decrypted.
Chosen plaintext attack	A form of cryptanalysis where the cryptanalyst may choose the plaintext to be encrypted.
Cipher	An encryption and decryption algorithm.
Ciphertext	Encrypted data.
Ciphertext only attack	A form of cryptanalysis where the cryptanalyst has some ciphertext for analysis, but nothing else.
Collision	Two values x and y form a *collision* of a (supposedly) one-way function F if $x \neq y$ but $F(x) = F(y)$.
Collision free	A one-way or hash function is called *collision free* if collisions are hard to find in general. The function is *weakly collision free* if it is computationally hard to find a collision for a given message x. That is, it is computationally infeasible to find a message y for a given x such that $H(x) = H(y)$. A hash function is *strongly collision free* if it is computationally infeasible to find *any* messages x, y such that $x \neq y$ and $H(x) = H(y)$.

Collision search	The search for a collision of a one-way function.
Commutative	When a mathematical operator yields the same result regardless of the order the objects are operated on. For example if *a*, *b* are integers then $a+b=b+a$, that is, the addition operator acting on integers is commutative.
Commutative group	A group where the operator is commutative, also called an *abelian group*.
Computational complexity	Refers to the amount of space (memory) and computation time required to solve a problem.
Cryptanalysis	The art and science of breaking encryption or any form of *cryptography*. See also *attack*.
Cryptography	The art and science of using mathematics to secure information and create a high degree of trust in the electronic world.
Cryptology	The branch of mathematics concerned with *cryptography* and *cryptanalysis*.
Cryptosystem	An encryption and decryption algorithm (*cipher*), together with all possible *plaintexts*, *ciphertexts* and *keys*.

Decryption	The inverse or reverse operation of *encryption*. Decryption takes *ciphertext* and transforms it into *plaintext*.
Dictionary attack	A brute force attack that tries passwords, keys or any other cryptographic information from a precompiled list (a dictionary) of possible values. This is often done as a precomputation attack.
Digest	Commonly used to refer to the output of a *hash function*, e.g. *message digest* refers to the *hash* of a message.
Digital signature	The encryption of a *message digest* with a *private key*.
Discrete logarithm	Given two elements *d, g*, in a group such that there is an integer *r* satisfying $g^r = d$, *r* is called the discrete logarithm.
Discrete logarithm problem	The problem of given *d* and *g* in a group, to find *r* such that $g^r = d$. For some groups, the discrete log problem is a hard problem that can be used in public key cryptography.
DSA	Digital Signature Algorithm. DSA is a public key signature method based on the discrete logarithm problem.

DSS	Digital Signature Standard. The DSS is in fact equivalent to the *DSA*.
ECC	Elliptic Curve Cryptosystem; A public key cryptosystem based on the properties of elliptic curves.
ECDL	Elliptic curve discrete logarithm.
ECDLP	Elliptic curve discrete logarithm problem. The problem of given two points *P* and *Q* on an elliptic curve, to find *m* satisfying $mP = Q$, assuming such an *m* exists.
Elliptic curve	The set of points (x, y) satisfying an equation of the form $y^2 = x^3 + ax + b$, for variables *x*, *y* and constants *a*, *b*.
Elliptic curve discrete logarithm problem	See *ECDLP*.
Elliptic curve factoring method	A special purpose factoring algorithm that attempts to find a prime factor *p* of an integer *n* by finding an elliptic curve whose number of points modulo *p* is divisible by only small primes.

Encryption	The transformation of *plaintext* into an apparently less readable form (called *ciphertext*) through a mathematical process. The *ciphertext* may be read by anyone who has the key that decrypts (undoes the encryption) the *ciphertext*. Briefly speaking, encryption takes *plaintext* and transforms it into *ciphertext*.
Exhaustive search	Checking every possible value until the right value is found. See also *brute force attack*.
Exponential function	A function where the variable is in the exponent of some base. For example take $f(x) = b^x$, where x is the variable, and b is some constant.
Exponential running time	If the running time, given as a function of the length of the input, is an exponential function, the algorithm is said to have exponential running time.
Factor	Given an integer *N*, any number that divides it is called a factor.
Factoring	The breaking down of an integer into its prime factors. This is regarded a hard problem.

Factoring method	A special method or algorithm for factoring an integer. Known methods are the *elliptic curve factoring method*, the *quadratic sieve*, the *number field sieve*, and the *Pollard rho method*.
Field	A mathematical structure with multiplication and addition that behave as they do with the real numbers.
Flat keyspace	See *Linear Key Space*.
Galois field	A finite field.
General purpose factoring algorithm	A factoring algorithm which is efficient of effective for *all* numbers. See also *factoring* and *prime factors*.
Group	A mathematical structure in which elements are combined.
Hard problem	A computationally intensive problem. This is a problem that is computationally difficult to solve.
Hash function	A function that takes a variable sized input and has a fixed size output. See also *digest*.
Index calculus	The most efficient method published today to solve the discrete logarithm problem.

Intractable	In complexity theory, referring to a problem with no efficient means of deriving a solution.
Key	A string of bits used widely in *cryptography*, allowing entities to encrypt and decrypt data; a key can be used to perform other mathematical operations as well. Given a *cipher*, a key determines the mapping of the *plaintext* to the *ciphertext*.
Key agreement	A process used by two or more parties to agree upon a secret symmetric key.
Key exchange	A process used by two more parties to exchange keys in cryptosystems.
Key generation	The process of creating a key in a cryprosystem.
Key life cycle	The length of time a key can be kept in use while still providing an appropriate level of security.
Key management	The various processes that deal with the creation, distribution, authentication, and storage of keys.
Key pair	The full key information in a public-key cryptosystem, consisting of the *public key* and *private key*.

Keyspace	The collection of all possible keys for a given cryptosystem. See also *linear key space, nonlinear key space,* and *reduced key space*.
Known plaintext attack	A form of *cryptanalysis* where the cryptanalyst knows both the *plaintext* and the associated *ciphertext*.
Linear complexity	Referring to a sequence of 0‘s and 1‘s, the size of the smallest linear feedback shift register (LFSR) that would replicate the sequence.
Linear cryptanalysis	A *known plaintext attack* that uses linear approximations to describe the behavior of the block cipher. See *known plaintext attack*.
Linear keyspace	A key space where each key is equally strong.
LSFR	Linear feedback shift register. Used in many keystream generators because of its ability to produce sequences with certain desirable properties.
Message digest	The result of applying a *hash function* to a message.

Modular arithmetic	A form of arithmetic where integers are considered equal if they leave the same remainder when divided by the *modulus*.
Modulus	The integer used to divide out by in *modular arithmetic*.
Nondeterministic	Not determined or decided by previous information.
Nondeterministic computer	A theoretical computer capable of choosing a next step out of a set of possible steps by guessing during each state of computation.
Nondeterministic polynomial running time (NP)	Problems that are, in the worst case, verifiable through a certificate of correctness which runs in polynomial time.
Nonlinear keyspace	A key space comprised of strong and weak keys.
Non-repudiation	A property of a *cryptosystem*. Non-repudiation cryptosystems are those in which the users cannot deny actions they performed.

NP	*Nondeterministic Polynomial*. Refers to the running time of the best known algorithm for solving a particular problem. A set of problems where the best-known algorithm solving it would run in polynomial time on a nondeterministic computer; such problems are said to be *NP* or *in NP*.
NP-complete	A problem is *NP-complete* if any *NP* problem can be reduced (transformed) to it, and it is itself *NP*.
Number field sieve	A method of factoring, currently the fastest general purpose factoring algorithm published.
Number theory	A branch of mathematics that investigates the relationships and properties of numbers.
One-way function	A function that is easy to compute in one direction but quite difficult to reverse compute (compute in the opposite direction).
One-way hash function	A *one-way function* that takes a variable sized input and creates a fixed size output. See also *hash function* or *message digest*.

P	*Polynomial*. Refers to the running time of the best known algorithm for solving a particular problem. A set of problems where the best known algorithm solving it runs in polynomial time; such a problem is said to be *P* or *in P*.
Plaintext	The raw data to be encrypted.
Pollard rho method	A possible and very effective method for solving the *discrete logarithm* and *elliptic curve discrete logarithm*.
Polynomial	An algebraic expression written as a sum of constants multiplied by different powers of a variable. For example the expression $a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$, where the a_i are the constants and *x* is the variable.
Polynomial running time (P)	A polynomial running time algorithm is an algorithm whose worst case running time, given as a function of the length of the input, is a polynomial.
Precomputation attack	An attack where the adversary precomputes a look-up table (a dictionary) of values used to break the encryption, keys or passwords. See also *dictionary attack*.

Primality testing	A test that determines, with varying degree of probability, whether or not a particular number is prime.
Prime factor	A prime number that is a factor of another number is called a prime factor of that number.
Prime number	Any integer greater than 1 that is divisible only by 1 and itself. The integer 2 is a special case as it is the only prime number which is even. It has some very special properties as a prime number, which makes it the oddest of all primes (as it is sometime referred to).
Private key	In public key cryptography, this key is the secret key of some entity. It is used for decryption but is also used for computation of digital signatures.
Protocol	In cryptography this refers to a series of steps that two or more parties agree upon to complete a specific task.
Pseudo random number	A number extracted from a pseudo random sequence.

Pseudo random sequence	A deterministic function which produces a sequence of bits with qualities similar to that of a truly random sequence. There are methods to messure the quality of a pseudo random sequence.
Public key	In public key cryptography this key is made public. This allows for encryption of messages to the owner of the key but can also be used for verifying digital signatures.
Public key cryptography	Cryptography based on methods involving a *public key* and a *private key*.
Quadratic sieve	An algorithm for factoring an integer, developed by Carl Pomerance.
Random number	As opposed to a pseudo-random number, a truly random number is a number produced independently of its generating criteria. For cryptographic purposes, numbers based on physical measurements, such as based on radioactivity, are considered truly random.

Reduced keyspace	When using an *n* bit key, some implementations may only use $r < n$ bits of the key; the result is a smaller (reduced) key space.
Relatively prime	Two integers are relatively prime if they have no common factors. For example, 14 and 25 are relatively prime, while 14 and 91 are not; 7 is a common factor.
RSA algorithm	A public key cryptosystem based on the factoring problem. RSA stands for Rivest, Shamir and Adleman, the developers of the RSA public key cryptosystem.
Running time	A measurement of the time required for a particular algorithm to run as a function of the input size. See also *exponential running time*, *nondeterministic polynomial running time*, *polynomial running time*, *sub-exponential running time*.
Secret key	In symmetric (or secret key) cryptography, this is the key used both for encryption and decryption.

Secret sharing	Splitting a secret (e.g. a private key) into many pieces such that any specified subset of *k* pieces may be combined to form the secret, but *k*-1 pieces are not enough.
Shared key	The secret key two (or more) users share in a symmetric-key cryptosystem. See also *shared secret*.
Shared secret	A piece of information (such as a cryptographic key) which is shared between the communicating parties but which is kept secret in general.
Special purpose factoring algorithm	A factoring algorithm which is efficient or effective only for *some* numbers. See also *factoring* and *prime factors*.
Strong prime	A *prime number* with certain properties chosen to defend against specific factoring techniques.
Sub-exponential running time	The running time is less than exponential. Polynomial running time algorithms are sub-exponential, but not all sub-exponential algorithms are polynomial running time.
Symmetric cipher	An encryption algorithm that uses the same key for encryption as for decryption.
Symmetric key	See *secret key*.

Tractable	A property of a problem, stating that it can be solved in a reasonable amount of time using a reasonable amount of space.
Trapdoor one-way function	A *one-way function* that has an inverse which is easy to compute, given some certain secret information. This secret information is called the trapdoor.

8 Bibliography

8.1 Books

[Blake et al 1999] Ian Blake, Gladiel Seroussi, Nigel Smart (1999), Elliptic Curves in Cryptography, Cambridge University Press, ISBN 0-521-65374-6

[Berlekamp 1984] Elwyn R. Berlekamp (1984), Algebraic Coding Theory, Revised 1984 Edition, Aegean Park Press, ISBN 0-89412-063-8

[Knuth 1997] Donald E. Knuth, (1997), The Art of Computer Programming: Seminumerical Algorithms, 3rd Edition, Addison-Wesley, ISBN 0-201-89684-2

[Koblitz 1994] Neal Koblitz, (1994), A Course in Number Theory and Cryptography, 2nd Edition, Springer New York, ISBN 0-387-94293-9

[Koblitz 1993] Neal Koblitz, (1993), Introduction to Elliptic Curves and Modular Forms, 2nd Edition, Springer New York, ISBN 0-387-97966-2

[Menezes et al 1997]	Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone (1997), Handbook of Applied Cryptography, CRC Press, Inc., ISBN 0-8493-8523-7
[Silverman 1986]	Joseph H. Silverman, (1986), The Arithmetic of Elliptic Curves, 1st Edition, Springer New York, ISBN 0-387-96203-4
[Singh 1997]	Simon Singh (1997), Fermat's Last Theorem, Fourth Estate, London, ISBN 1-85702-669-1
[Stinson 1995]	Douglas R. Stinson (1995), Cryptography – Theory and Practice, CRC Press, Inc., ISBN 0-8493-8521-0
[Schneier 1996]	Bruce Schneier (1996), Applied Cryptography, 2nd Edition, John Wiley & Sons, Inc., ISBN 0-471-11709-9

8.2 Articles

[Johnson, Menezes]	Don B. Johnson, Alfred J. Menezes, Elliptic Curve DSA (ECDSA): An Enhanced DSA
[Jurisic, Menezes]	Aleksandar Jurisic, Alfred J. Menezes, Elliptic Curves and Cryptography
[Koblitz 1992]	Neal Kobitz, CM—Curves with Good Cryptographic Properties, CRYPTO ’91 (Springer New York 1992)
[Lenstra et al 1999]	Arjen K. Lenstra, Eric R. Verheul (1999), Selecting Cryptographic Key Sizes
[Schroeppel et al 1995]	Richard Schroeppel, Hilarie Orman, Sean O’Malley (1995), Fast Key Exchange with Elliptic Curve Systems