Memorandum

To: P1363 working group
From: Alfred Menezes, Minghua Qu and Scott Vanstone
Subject: IEEE P1363, *Part 4: Elliptic Curve Systems*
Date: November 6 1995.

Enclosed is a copy of IEEE P1363, Part 4, November 6 1995, for your review.

I have taken into consideration comments made at the last meeting. The following is a list of major changes made to the August 25, 1995 draft.

1. Improved the accessibility of the introduction as per Terry Arnold's suggestions.
2. Elliptic curves now form Part 4. As suggested at the last meeting, the sections have been organized into 6 main sections: Basic algorithms, Key generation and key syntax, ECES, ECSS, ECDSA, and ECKEP.
3. The sections of the Appendix were rearranged as suggested at the last meeting.
4. Incorporated numerous suggestions made by Burt Kaliski and Lisa Yin.
5. Changed the key agreement protocol (ECKEP) from Protocol 2' to Protocol 1' (see the paper "Some new key agreement protocols providing implicit authentication", by Menezes, Qu and Vanstone that was distributed at the August meeting). Protocol 1' is more efficient than Protocol 2'.
6. ECES was simplified as discussed at the last meeting.
7. The order n of the point P is now required to be a prime number.
8. As discussed at the last meeting, ASN.1 specification of encrypted message and signatures was removed.
9. Included ASN.1 for private key syntax.
10. Included an example of ECDSA.

To be completed:

1. Key Length Considerations (to be supplied by Jerry Solinas).
2. Include test vectors.
3. Translation of LaTeX source to a Microsoft Word document.
4. Include the detailed description of ECKEP.

Some possible discussion topics for the November 15 meeting:

1. Discuss Jerry Solinas' suggestion of setting $a = -3$ for elliptic curves over $\mathbb{F}_p$ (Key Option 2).
2. Need some help with ASN.1 for the key syntax.

WORKING DRAFT

IEEE P1363 STANDARD

STANDARD FOR RSA, DIFFIE-HELLMAN AND RELATED PUBLIC-KEY CRYPTOGRAPHY

PART 4: ELLIPTIC CURVE SYSTEMS (Draft 6)

Notice – Warning to readers of this document

This document is in the working document stage. It has not yet been processed through the consensus procedures of the IEEE.

Many changes which may greatly affect the contents can occur before this document becomes an IEEE Standard. The developmental committee may not be held responsible for the contents of this document as it currently exists.

Implementation or design based on this working paper is at the risk of the user. No advertisement implying compliance with this "Standard" should appear as it is erroneous and misleading to so state.

Dr. Alfred J. Menezes
Dr. Minghua Qu
Dr. Scott A. Vanstone

MÖBIUS ENCRYPTION TECHNOLOGIES
200 MATHESON BOULEVARD, WEST
MISSISSAUGA, ONTARIO,
CANADA, L5R 3L7

November 6 1995

Outline

Part 4

Elliptic Curve Systems

Abstract.[1] *This standard describes a method for data encryption and for digital signatures using the elliptic curve analogue of the ElGamal public-key cryptosystem, and a method for key agreement using a variant of the Diffie-Hellman key agreement protocol. Elliptic curve systems are public-key (asymmetric) cryptographic algorithms, typically used in conjunction with a hash algorithm to create digital signatures, and for the secure distribution of secret keys for use in symmetric-key cryptosystems. Elliptic curve systems may also be used to transmit confidential information.*

Introduction

Many public-key cryptographic systems are based on exponentiation operations in large finite mathematical groups. The cryptographic strength of these systems is derived from the believed computational intractability of computing logarithms in these groups. The most commonly seen groups are the multiplicative group of $\mathbb{Z}_p$ (the integers modulo a prime p) and $\mathbb{F}_{2^m}$ (characteristic 2 finite fields). The primary advantages of these groups have been their rich theory, easily understood structure, and straight-forward implementation. However, they are not the only groups that have the requisite properties. In particular the mathematical structures known as elliptic curves have the requisite mathematical properties, a rich theory, and are especially amenable to efficient implementation in hardware or software.

The algebraic system defined on the points of an elliptic curve provides an alternate means to implement the ElGamal and ElGamal-like public key encryption and signature protocols. These protocols are typically described in the literature in the algebraic system $\mathbb{Z}_p$, the integers modulo p, where p is a prime. For example, the NIST Digitial Signature Algorithm (DSA) is an ElGamal-like signature scheme defined over $\mathbb{Z}_p$. Precisely the same protocol for signing could be defined over the points on an elliptic curve.

Elliptic curve systems as applied to ElGamal protocols were first proposed in 1985 independently by Neil Koblitz of the University of Washington, and Victor Miller, who was

[1] This section will be moved to a "rationale" section, probably in the Appendix.

then at IBM, Yorktown Heights. Elliptic curves as algebraic/geometric entities have been studied extensively for the past 150 years, and from these studies has emerged a rich and deep theory.

The security of the cryptosystems using elliptic curves hinges on the intractability of the discrete logarithm problem in the algebraic system defined on the elliptic curve points. Over the past ten years this problem has received considerable attention from leading mathematicians around the world. Unlike the case of the discrete logarithm problem in finite fields, or the problem of factoring integers, there is no general subexponential-time algorithm known for the elliptic curve discrete logarithm problem. The best algorithm known to date takes exponential time.

The primary advantage of elliptic curve systems is thus their high cryptographic strength relative to key size. This will tend to increase the attractiveness of the elliptic curve cryptosystems relative to other public-key cryptosystems as computing power improvements warrant general key size increases. The significantly shorter key size results in significantly shorter certificates and system parameters. These advantages manifest themselves in many ways including storage efficiencies, bandwidth savings and computational efficiencies. The computational efficiencies lead in turn to higher speeds and power efficiency.

If the elliptic curve systems are implemented using the finite field $\mathbb{F}_{2^m}$, further computational efficiencies are available. This is particularly true in hardware implementations where very compact architectures exist for finite field multiplication.

The computational efficiencies available with the use of these implementations combined with the efficiencies of elliptic curves in general are particularly beneficial in applications where bandwidth, processing capacity, power availability and storage are constrained. Such applications include wireless communications, handheld computing, broadcast and smart card applications.

Elliptic curves systems have been implemented by various groups around the world including Siemens (Germany), Matsushita (Japan), Thompson (France) and MÖBIUS Encryption Technologies (Canada). An ISO/IEC SC27 standard for elliptic curve systems is currently being drafted.

This standard describes the implementation of various cryptographic protocols defined by elliptic curves over a finite field $\mathbb{F}_q$ where q is either a prime number or $q = 2^m$ for some positive integer m.

Associated with any finite field $\mathbb{F}_q$ there are on the order of q different elliptic curves that can be formed and used for the cryptosystems; thus, for a fixed finite field with q elements and q large, there are many choices for the elliptic curve group. Since one elliptic curve operation requires a number of more basic operations in the underlying finite field $\mathbb{F}_q$, the preceding relation between a finite field and the number of associated elliptic curves permits one to select a finite field so that either a software or hardware realization is very efficient but retaining the desirable feature that one will still have an enormous number of choices for the cryptosystem used.

Although the finite field $\mathbb{F}_p = \mathbb{Z}_p$, the integers modulo p a prime number, is more well known and is typically the one which comes to most people's minds when the term finite field is used, it may not be the most appropriate for the purposes of elliptic curve cryptosystems. It appears currently that the most efficient implementations are realized in the finite field $\mathbb{F}_{2^m}$. This is particularly so if the implementation is in hardware. Circuit designs for VLSI devices to do $\mathbb{F}_{2^m}$ operations are very regular and give rise to low power and high speed chips. Elliptic curve cryptosystems over finite fields require the computation of multiplicative inverses in the underlying field. These are most efficiently computed in fields of the form $\mathbb{F}_{2^m}$ and lead to substantial savings in hardware implementations.

Definitions and Abbreviations

binary string	A *binary string* is a sequence of 0's and 1's. The leftmost bit is the *most significant bit* of the string. The rightmost bit is the *least significant bit* of the string.
octet	An *octet* is a binary string of length 8. An octet is represented by a hexadecimal string of length 2. The first hexadecimal digit represents the four most significant bits of the octet. The second hexadecimal digit represents the four least significant bits of the octet. For example, 9d represents the binary string 10011101.
octet string	An *octet string* is a sequence of octets.
PS	Padding string.
optimal normal basis	A special kind of representation for the elements of the finite field $\mathbb{F}_{2^m}$.
elliptic curve	An elliptic curve is specified by 2 parameters a and b, which are elements of a field $\mathbb{F}_q$. The elliptic curve is said to be *defined over* $\mathbb{F}_q$, and $\mathbb{F}_q$ is sometimes called the *underlying field.* If q is a prime (so the field is $\mathbb{F}_p$), then the equation defining the curve is of the form $y^2 = x^3 + ax + b$, where $(4a^3 + 27b^2 \bmod p) \neq 0$. If q is a power of 2 (so the field is $\mathbb{F}_{2^m}$), then the equation defining the curve is of the form $y^2 + xy = x^3 + ax^2 + b$, where $b \neq 0$.
elliptic curve point	If E is an elliptic curve defined over a field $\mathbb{F}_q$, an elliptic curve point is a pair of field elements (x_P, y_P), $x_P \in \mathbb{F}_q$, $y_P \in \mathbb{F}_q$, such that the values $x = x_P$ and $y = y_P$ satisfy the equation defining E.
x-coordinate	The x-coordinate of an elliptic curve point $P = (x_P, y_P)$ is x_P.
y-coordinate	The y-coordinate of an elliptic curve point $P = (x_P, y_P)$ is y_P.
addition rule	There is an addition rule which allows the addition of two elliptic curve points P_1 and P_2 to produce a third elliptic curve point P_3. (This is described in Sections A.4 and A.5.)
exponentiation	If k is a positive integer, then kP denotes the point obtained by adding together k copies of the point P. The process of computing kP from P and k is called *exponentiation.*
order of a point	The order of a point P is the smallest positive integer such that $nP = \mathcal{O}$.
order of a curve	The order of an elliptic curve E defined over the field $\mathbb{F}_q$ is the number of points defined over $\mathbb{F}_q$, and including the point at infinity $\mathcal{O}$.
supersingular	An elliptic curve E defined over $\mathbb{F}_p$ is supersingular if $\#E(\mathbb{F}_p) = p+1$. An elliptic curve defined over $\mathbb{F}_{2^m}$ is *supersingular* if its defining equation is of the form $y^2 + cy = x^3 + ax + b$, where a, b, $c \in \mathbb{F}_{2^m}$ and $c \neq 0$ — these curves are not considered in this standard.
non-supersingular	If the elliptic curve is not supersingular, it is called *non-supersingular.*

point compression	Let P be a point (x_P, y_P) on an elliptic curve E defined over a field $\mathbb{F}_q$. Point compression allows the point P to be represented using x_P and a single additional bit derived from x_P and y_P. If point compression is used and q is a prime, then $\widetilde{y_P}$ is equal to the least significant bit of y_P. If point compression is used and q is a power of 2, then $\widetilde{y_P}$ is 0 if $x_P = 0$; if $x_P \neq 0$, then $\widetilde{y_P}$ is equal to the least significant bit of the field element $y_P \cdot {x_P}^{-1}$. The point compression technique is described in Section 4.2.4 If point compression is not used then $\widetilde{y_P}$ is just y_P.
PC hline ECES	Point compression octet. If point compression is used then the least significant bit of PC is 1, and the second least significant bit is equal to the value of $\widetilde{y_P}$. If point compression is not used then the least significant bit of PC 0. Elliptic Curve Encryption Scheme.
ECSS	Elliptic Curve Signature Scheme.
ECDSA	Elliptic Curve analogue of the NIST Digital Signature Algorithm (DSA).
ECKEP	Elliptic Curve Key Establishment Protocol.
SHA-1	The Secure Hash Algorithm.
hash value	When a message of length less than 2^{64} bits is input, SHA-1 produces a 160-bit representation of the message called the *message digest* or *hash value*. Any change of the message will, with very high probability, result in a different message digest.
ISO	International Organization for Standardization.
IEC	International Electrotechnical Commission.
ANSI	American National Standards Institute.
ASN.1	Abstract Syntax Notation One. A notation for describing abstract types and values that is described in standard ISO/IEC 8824.
BER	Basic Encoding Rules. A set of rules for representing or encoding the values of each ASN.1 type as a string of octets. There is usually more than one way to encode a given value using BER encoding rules. BER is defined in standard ISO/IEC 8825.
DER	Distinguished Encoding Rules. A subset of BER which gives a unique way to represent any ASN.1 value as an octet string. DER is defined in standard ISO/IEC 8825.

Symbols and Notation

$\lceil x \rceil$	The smallest integer $\geq x$. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.
$\lfloor x \rfloor$	The largest integer $\leq x$. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.
$X \oplus Y$	Bitwise *exclusive or* of two binary strings X and Y.
$X \Vert Y$	Concatenation of two strings X and Y. X and Y are either both binary strings, or both octet strings.
$\Vert X \Vert$	Length in octets of the octet string X.
$\log_2 x$	The logarithmic function to the base 2.
$a \bmod n$	The unique remainder r, $0 \leq r \leq n-1$, when integer a is divided by n. For example, $(23 \bmod 7) = 2$.
$\mathbb{Z}_p$ or $\mathbb{F}_p$	The integers modulo p, where p is a prime number.
$\mathbb{F}_{2^m}$	The finite field containing 2^m elements.
$\mathbb{F}_q$	The finite field containing q elements. For this standard, q will either be a prime number (p) or a power of 2 (2^m).
t	A field element of $\mathbb{F}_q$ will be represented as a binary string of length $t = \lceil \log_2 q \rceil$. In particular, if $q = 2^m$, then a field element in $\mathbb{F}_{2^m}$ can be represented as a binary string of length $t = m$.
E	An *elliptic curve*
P	A point on an elliptic curve.
$\mathcal{O}$	A special point on an elliptic curve, called the *point at infinity*.
n	The order of the point P. For this standard, n shall always be a prime number.
$E(\mathbb{F}_q)$	The set of all points on the elliptic curve E defined over $\mathbb{F}_q$, and including the point at infinity $\mathcal{O}$.
$\#E(\mathbb{F}_q)$	If E is defined over $\mathbb{F}_q$, then $\#E(\mathbb{F}_q)$ denotes the order of E.
$\widetilde{y_P}$	The representation of the y-coordinate of a point P.

Conventions for Data Representations

binary string to integer conversion	Let m be a binary string of length k. Let $m_1, m_2, \ldots, m_k$ be the bits of m from first (most significant) to last (least significant). Then m shall be converted to an integer x satisfying $x = \sum_{i=1}^{k} 2^{(k-i)} m_i$.
octet string to binary string conversion	Let M be an octet string of length k. Let $M_1, M_2, \ldots, M_k$ be the octets of M from first to last. Then M shall be converted to a binary string m of length $8k$ as follows. Let the bits of m be $m_1, m_2, \ldots, m_{8k}$ from first bit to last bit. For each i, $1 \leq i \leq k$, the bits $m_{8(i-1)+j}$, $1 \leq j \leq 8$, shall be the bits of octet M_i.

4.1 Basic Algorithms

This section gives a high-level overview of the elliptic curve encryption scheme (ECES), two elliptic curve signature schemes (ECSS and ECDSA), and an elliptic curve key establishment protocol (ECKEP). The cryptosystems are described using an arbitrary elliptic curve E over an arbitrary finite field $\mathbb{F}_q$. Complete details, refinements and options are provided in Sections 4.2–4.6. Appendix A contains an introduction to the theory of finite fields and elliptic curves. Numerical examples of ECES, ECSS, ECDSA and ECKEP are given in Section A.6.

4.1.1 System Setup and Key Generation

System Setup

An underlying finite field $\mathbb{F}_q$ is chosen. An elliptic curve E defined over $\mathbb{F}_q$, and a point P of prime order n on E are chosen.
The field $\mathbb{F}_q$, curve E, point P, and order n, comprise the system parameters, and are public information.

Key Generation

Each entity shall perform the following operations:

1. Select a random integer d in the interval $[1, n-1]$.
2. Compute the point $Q := dP$.
3. The entity's public key consists of the point Q.
4. The entity's private key is the integer d.

4.1.2 Elliptic Curve Encryption Scheme (ECES)

Encryption Process

(Entity B sends data M to entity A)

Entity B performs the following steps:

1. Look up A's public key Q.
2. Represent the data M as a field element $m \in \mathbb{F}_q$.
3. Select a random integer k in the interval $[1, n-1]$.
4. Compute the point $(x_1, y_1) := kP$.
5. Compute the point $(x_2, y_2) := kQ$. If $x_2 = 0$ then go to step 3.
6. Compute $c := m \cdot x_2$.
7. Transmit the encrypted data (x_1, y_1, c) to A.

Decryption Process

(Entity A decrypts ciphertext (x_1, y_1, c) received from B)

Entity A performs the following steps:

1. Compute the point $(x_2, y_2) := d(x_1, y_1)$, using its private key d.
2. Recover the data m by computing $m := c \cdot x_2^{-1}$.

4.1.3 Elliptic Curve Signature Schemes (ECSS and ECDSA)

Two signature schemes are described in this standard. Both are examples of signature schemes with appendix. In these schemes the message to be signed is first hashed to a message digest of fixed length, and then this digest is signed. Verification of the signature requires both the signature and the original message. ECDSA is the elliptic curve analogue of the NIST Digital Signature Algorithm (DSA).

Signature Generation for ECSS

(Entity A signs a message M for entity B)

A performs the following steps:

1. Represent the message M as a binary string.
2. Use a hash algorithm to compute the hash value $e := H(M)$.
3. Select a random integer k in the interval $[1, n-1]$.
4. Compute the point $(x_1, y_1) := kP$.
5. Compute $r := x_1 + e \bmod q$.
6. Use the private key d to compute $s := k - dr \bmod n$.
7. A sends to B the message M and the signature (r, s).

Signature Verification for ECSS

(Entity B verifies A's signature (r, s) for a message M.)

B performs the following steps:

1. Look up A's public key Q.
2. Compute the point $(x_1, y_1) := sP + rQ$.
3. Compute the hash value $e := H(M)$.
4. Compute $r' := x_1 + e \bmod q$.
5. Accept A's signature for message M if and only if $r = r'$.

Notes

The hash value e is reduced modulo q and then modulo n. Hence for security reasons it is necessary that the hash function H, q and n be chosen so that (H mod q) mod n is also a cryptographically secure hash function.

Signature Generation for ECDSA

(Entity A signs a message M for entity B)

A performs the following steps:

1. Represent the message M as a binary string.
2. Use a hash algorithm to compute the hash value $e := H(M)$.
3. Select a random integer k in the interval $[1, n-1]$.
4. Compute the point $(x_1, y_1) := kP$ and set $r := x_1 \bmod n$.
5. Use the private key d to compute $s := k^{-1}(e + rd) \bmod n$.
6. A sends to B the message M and its signature (r, s).

Notes

If $r = 0$ or $s = 0$ then signature verification will fail. However, if k is chosen randomly, the probability that either $r = 0$ or $s = 0$ is negligibly small.

Signature Verification for ECDSA

(Entity B verifies A's signature (r, s) for a message M.)

B performs the following steps:

1. Look up A's public key Q.
2. If $(r \bmod n) = 0$ then reject the signature.
3. Compute the hash value $e := H(M)$.
4. Compute $s^{-1} \bmod n$.
5. Compute $u := s^{-1}e \bmod n$ and $v := s^{-1}r \bmod n$.
6. Compute the point $(x_1, y_1) := uP + vQ$.
7. Accept A's signature for the message M if and only if $(x_1 \bmod n) = r$.

Notes

(a) The hash value e is reduced modulo n. Hence for security reasons it is necessary that the hash function H and n be chosen so that H mod n is also a cryptographically secure hash function.

(b) The condition $r \neq 0$ imposed in step 4 of signature generation is a security condition – if $r = 0$ then the signature equation $s = k^{-1}(e + rd)$ does not involve the private key d.

(c) If k is chosen at random then the probability that either $r = 0$ or $s = 0$ is negligibly small.

4.1.4 Elliptic Curve Key Establishment Protocol (ECKEP)

This section describes a protocol whereby two parties A and B establish a shared secret key K called the *session key*. The session key may subsequently used to achieve some cryptographic goal, such as privacy or authentication.

System Setup

It is assumed that A and B are using the same elliptic curve parameters $\mathbb{F}_q$, E, P and n. A has private key d_A and public key $Q_A = d_A P = (x_A, y_A)$. B has private key d_B and public key $Q_B = d_B P = (x_B, y_B)$.

1. Entity A does the following:
   (a) Select a random integer k_A, $1 \leq k_A \leq n - 1$.
   (b) Compute the point $(x_1, y_1) = R_A := k_A P$.
   (c) A sends R_A to B.
2. Entity B does the following:
   (a) Select a random integer k_B, $1 \leq k_B \leq n - 1$.
   (b) Compute the point $(x_2, y_2) = R_B := k_B P$.
   (c) B sends R_B to A.
3. A does the following:
   (a) Compute the integer $s_A := k_A + x_1 d_A x_A \bmod n$.
   (b) Compute the session key $K := s_A(R_B + x_2 x_B Q_B)$.
4. B does the following:
   (a) Compute the integer $s_B := k_B + x_2 d_B x_B \bmod n$.
   (b) Compute the session key $K := s_B(R_A + x_1 x_A Q_A)$.

Notes

(a) The protocol provides mutual implicit key authentication.

(b) The protocol does *not* provide key confirmation. The means by which key confirmation may be achieved are not specified in this standard.

(c) Some other features of this protocol are that it is *non-interactive* (the messages transmitted between the two parties are independent of each other), *role-symmetric* (the 2 messages transmitted have the same structure), does not require encryption, hash functions or timestamping, offers *perfect forward secrety*, and has low bandwidth requirements.

4.2 Key Generation and Key Syntax

4.2.1 Key Generation

This section describes ECES, ECSS, ECDSA, and ECKEP key generation.

4.2.1.1 System Setup

The underlying field $\mathbb{F}_q$ is selected first and is common to all users.

There are two basic options available for generating the remaining parameters.

1. *Key Option 1: Short Public Keys*
   The elliptic curve parameters, namely the field elements a and b (which define the elliptic curve), and the point P of prime order n, are selected by the system administrator. The parameters a, b, $P = (x_P, \widetilde{y_P})$, n, are public information and are common to all users.

2. *Key Option 2: Long Public Keys*
   Each user selects its own elliptic curve parameters, namely the field elements a and b (which define the elliptic curve), and the point P of prime order n. To make the public key shorter, the parameter a is restricted to be $a = 0$ if the underlying field is $\mathbb{F}_{2^m}$, and $a = 1$ if the underlying field is $\mathbb{F}_p$. The parameters b, $P = (x_P, \widetilde{y_P})$, must be part of the user's public key. If ECDSA is to be used, then the public key must furthermore include the order n of P.

Notes

(a) See Section 4.2.5 for guidance on the selecting the sizes of parameters q and n in order to achieve a desired level of security.

(b) See Section B.2 for a discussion on how to select a curve of appropriate order.

(c) See Section B.3 for a method of computing the order of a point.

4.2.1.2 Key Generation

After system setup, each entity performs the following operations.

1. Select a random integer d in the interval $[1, n-1]$.
2. Compute the point $Q := dP$.
3. Let $Q = (x_Q, y_Q)$, and compute $\widetilde{y_Q}$.
4. (a) For Key Option 1 (short public keys), the entity's public key consists of the point Q, which is represented by x_Q and $\widetilde{y_Q}$.

   (b) For Key Option 2 (long public keys), the entity's public key consists of the point P which is represented by x_P and y_P, and the point Q which is represented by x_Q and $\widetilde{y_Q}$. (Note that the elliptic curve parameter b can be easily computed from x_P and y_P as follows: if $q = p$ then $b = y_P^2 - x_P^3 - x_P$; if $q = 2^m$ then $b = y_P^2 + x_P y_P + x_P^3$).
5. The entity's private key is the integer d.

4.2.2 Key Syntax

This section describes the syntax for ECES, ECSS, ECDSA and ECKEP public and private keys.

If the field used is $\mathbb{F}_p$ then the following ASN.1 type is defined:

```
FieldElement ::= INTEGER (0..p-1)
```

Section 4.2.3.1 describes the representation that shall be used for the elements of $\mathbb{F}_{2^m}$.

If the field used is $\mathbb{F}_{2^m}$ then the following ASN.1 type is defined:

```
FieldElement ::= BIT STRING (SIZE(m))
```

Section 4.2.3.2 describes the representation that shall be used for the elements of $\mathbb{F}_p$.

An elliptic curve point is represented by two field elements, the x-coordinate and the y-coordinate. If point compression is used (Section 4.2.4) then the y-coordinate is represented by a single bit; otherwise the y-coordinate is a field element.

```
ECPoint ::= SEQUENCE  {
        x  FieldElement,   -- x coordinate
        y  CHOICE {        -- y coordinate
               fullYCoordinate [0] IMPLICIT FieldElement,
               compYCoordinate [1] IMPLICIT BOOLEAN }
}
```

4.2.2.1 Public Key Syntax

An elliptic curve public key shall have ASN.1 type `ECPublicKey`. The public key is a choice between the short form of the public key and the long form.

```
ECPublicKey ::= CHOICE  {
        ecpublickeyshort [0] IMPLICIT ECPublicKeyShort,
        ecpublickeylong  [1] IMPLICIT ECPublicKeyLong
}

ECPublicKeyShort ::= SEQUENCE  {
        pubPoint   ECPoint  -- Q
}

ECPublicKeyLong  ::= SEQUENCE  {
        basePointx       FieldElement,    -- Px
        basePointy       FieldElement,    -- Py
        pubPoint         ECPoint,         -- Q
        order            INTEGER OPTIONAL -- n
}
```

This is the public key as described in Section 4.2.1.2.

4.2.2.2 Private Key Syntax

An elliptic curve private key shall have ASN.1 type `ECPrivateKey`:

```
ECPrivateKey ::= CHOICE  {
        ecprivatekeyshort [0] IMPLICIT ECPrivateKeyShort,
        ecprivatekeylong  [1] IMPLICIT ECPrivateKeyLong
}

ECPrivateKeyShort ::= SEQUENCE {
        privateExponent INTEGER,  -- d
        pubPoint        ECPoint,  -- Q
}

ECPrivateKeyLong ::= SEQUENCE {
        privateExponent INTEGER,          -- d
        basePoint       ECPoint,          -- P
        curveParameter  FieldElement,     -- b
        pubPoint        ECPoint,          -- Q
        order           INTEGER           -- n
}
```

This is the private key as described in Section 4.2.1.2. The private key logically consists only of the private exponent, d, which is an integer in the interval 0 to $n-1$ where n is the order of P. The other parameters are included in the private key in order to make it simple to extract a public key from the private one.

4.2.3 Field Representation

This section describes the representation that shall be used for the elements of the underlying finite field $\mathbb{F}_q$.

4.2.3.1 The finite field $\mathbb{F}_p$

The finite field $\mathbb{F}_p$ is comprised of the set of integers $\{0, 1, 2, \ldots, p-1\}$. Each such integer can be represented by a binary string of length exactly $t = \lceil \log_2 p \rceil$ consisting of the binary representation of the integer padded on the left with the appropriate number of 0's.

4.2.3.2 The finite field $\mathbb{F}_{2^m}$

The elements of the finite field $\mathbb{F}_{2^m}$ shall be represented using an *optimal normal basis representation* as described in this section. Optimal normal bases (ONB) only exist in $\mathbb{F}_{2^m}$ for certain values of m. Table 4.1 lists all the values of $m \leq 2000$ for which the field $\mathbb{F}_{2^m}$ has an ONB. There are two kinds of ONB, called *type I ONB* and *type II ONB*. The difference between the two types of optimal normal bases is in the mathematical formulae which define them. In the table, $\star$ indicates the existence of a type I ONB, $\dagger$ indicates the existence of both type I and type II ONB, otherwise there exists only a type II ONB. In the case that both type I and type II ONB exist, the type II ONB shall be used.

The finite field $\mathbb{F}_{2^m}$ is comprised of the set of all binary strings of length exactly m. A typical element $a \in \mathbb{F}_{2^m}$ is represented by the binary string $a = (a_0 a_1 a_2 \cdots a_{m-1})$. The multiplicative identity element (1) is represented by the string of all 1's. The zero element (0) is represented by the string of all 0's. Addition of two field elements is accomplished simply by XORing the binary representations. Multiplication of field elements is described next.

Multiplication in $\mathbb{F}_{2^m}$ using an optimal normal basis representation

Setup

In what follows a polynomial $g(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0$, where each a_i is in the binary field $\mathbb{F}_2$, is represented by binary string $a_{m-1}a_{m-2}\cdots a_1 a_0$ of length m. The binary string can equivalently be viewed as a binary vector of length m.

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

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

2†	113	293	473	676⋆	873	1110	1310	1533	1790
3	119	299	483	683	876⋆	1116⋆	1323	1539	1791
4⋆	130⋆	303	490⋆	686	879	1118	1329	1541	1806
5	131	306	491	690	882⋆	1119	1331	1548⋆	1811
6	134	309	495	700⋆	891	1121	1338	1559	1818
9	135	316⋆	508⋆	708⋆	893	1122⋆	1341	1570⋆	1821
10⋆	138⋆	323	509	713	906⋆	1133	1346	1583	1829
11	146	326	515	719	911	1134	1349	1593	1835
12⋆	148⋆	329	519	723	923	1146	1353	1601	1838
14	155	330	522⋆	725	930	1154	1355	1618⋆	1845
18†	158	338	530	726	933	1155	1359	1620⋆	1850
23	162⋆	346⋆	531	741	935	1166	1370	1626	1854
26	172⋆	348⋆	540⋆	743	938	1169	1372⋆	1636⋆	1859
28⋆	173	350	543	746	939	1170⋆	1380⋆	1649	1860⋆
29	174	354	545	749	940⋆	1178	1394	1653	1863
30	178⋆	359	546⋆	755	946⋆	1185	1398	1659	1866†
33	179	371	554	756⋆	950	1186⋆	1401	1661	1876⋆
35	180⋆	372⋆	556⋆	761	953	1194	1409	1666⋆	1883
36⋆	183	375	558	765	965	1199	1418	1668⋆	1889
39	186	378†	561	771	974	1211	1421	1673	1898
41	189	378⋆	562⋆	772⋆	975	1212⋆	1425	1679	1900⋆
50	191	386	575	774	986	1218	1426⋆	1685	1901
51	194	388⋆	585	779	989	1223	1430	1692⋆	1906⋆
52⋆	196⋆	393	586⋆	783	993	1228⋆	1439	1703	1923
53	209	398	593	785	998	1229	1443	1706	1925
58⋆	210†	410	606	786⋆	1013	1233	1450⋆	1730	1926
60⋆	221	411	611	791	1014	1236⋆	1451	1732⋆	1930⋆
65	226⋆	413	612⋆	796⋆	1018⋆	1238	1452⋆	1733	1931
66⋆	230	414	614	803	1019	1251	1454	1734	1938
69	231	418⋆	615	809	1026	1258⋆	1463	1740⋆	1948⋆
74	233	419	618†	810	1031	1265	1469	1745	1953
81	239	420⋆	629	818	1034	1269	1478	1746⋆	1955
82⋆	243	426	638	820⋆	1041	1271	1481	1749	1958
83	245	429	639	826⋆	1043	1274	1482⋆	1755	1959
86	251	431	641	828⋆	1049	1275	1492⋆	1758	1961
89	254	438	645	831	1055	1276⋆	1498⋆	1763	1965
90	261	441	650	833	1060⋆	1278	1499	1766	1972⋆
95	268⋆	442⋆	651	834	1065	1282⋆	1505	1769	1973
98	270	443	652⋆	846	1070	1289	1509	1773	1978⋆
99	273	453	653	852⋆	1090⋆	1290⋆	1511	1778	1983
100⋆	278	460⋆	658⋆	858⋆	1103	1295	1518	1779	1986⋆
105	281	466⋆	659	866	1106	1300⋆	1522⋆	1785	1994
106⋆	292⋆	470	660⋆	870	1108⋆	1306⋆	1530⋆	1786⋆	1996⋆

Table 4.1: Values of $m \leq 2000$ for which the field $\mathbb{F}_{2^m}$ has an ONB.

$f(x)$ is a polynomial of degree m with coefficients in $\mathbb{F}_2$. The set of polynomials $\{x, x^2, x^{2^2}, \ldots, x^{2^{m-1}}\}$ form a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$, called a *normal basis.*

2. Construct the $m \times m$ matrix A whose i^{th} row, $0 \leq i \leq m-1$, is the binary vector 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.
4. Construct the $m \times m$ matrix T whose i^{th} row, $0 \leq i \leq m-1$, is obtained as follows. First compute the polynomial $x \cdot x^{2^i} \bmod f(x)$ and let the corresponding binary vector be denoted v. Then the i^{th} row of T is the binary vector $v \cdot A^{-1}$.
5. Determine the quantities λ_{ij}, for $0 \leq i, j \leq m-1$ as follows: $\lambda_{ij} = T(j-i, -i)$. Here, $T(i,j)$ denotes (i,j)-entry of T with indices reduced modulo m.
   Each λ_{ij} is an element of $\mathbb{F}_2$. It should be the case that $\lambda_{0j} = 1$ for precisely one j, $0 \leq j \leq m-1$. It should also be the case that for each i, $1 \leq i \leq m-1$, $\lambda_{ij} = 1$ for precisely two distinct j, $0 \leq j \leq m-1$. Hence exactly $2m-1$ of the m^2 entries of the matrix T are 1, the remainder being 0. (It is for this reason that the normal basis is called an *optimal normal basis.*)

Multiplication

Let $a = (a_0 a_1 a_2 \cdots a_{m-1})$ and $b = (b_0 b_1 b_2 \cdots b_{m-1})$ be two elements in $\mathbb{F}_{2^m}$. Then the product of a and b is the field element $c = (c_0 c_1 c_2 \cdots c_{m-1})$, where the coefficients c_k are computed using the formula

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij}, \quad 0 \leq k \leq m-1,$$

where all subscripts are reduced modulo m.

Notes

(a) Representing the elements of $\mathbb{F}_{2^m}$ using an optimal normal basis leads to efficient hardware implementation of the field arithmetic. If one chooses to use any other representation for $\mathbb{F}_{2^m}$, such as a polynomial basis representation, then it is easy to convert between the two representations by utilizing the appropriate change-of-basis matrices.

(b) An example of field arithmetic using an optimal normal basis representation is given in Section A.3.

4.2.4 Representing an Elliptic Curve Point

An elliptic curve point P (which is not the point at infinity $\mathcal{O}$) is represented by two field elements, the x-coordinate of P and the y-coordinate of P: $P = (x_P, y_P)$. The point can be represented more compactly by storing only the x-coordinate x_P and a certain bit $\widetilde{y_P}$

derived from the x-coordinate x_P and the y-coordinate y_P. The next two subsections show how the full y-coordinate y_P can be recovered from x_P and $\widetilde{y_P}$ if this point compression technique is used.

4.2.4.1 Point compression (elliptic curves over $\mathbb{F}_p$)

Let $P = (x_P, y_P)$ be a point on the elliptic curve $E : y^2 = x^3 + ax + b$ defined over a prime field $\mathbb{F}_p$. Then $\widetilde{y_P}$ is defined to be the least significant bit of y_P.

Suppose that we are given the x-coordinate x_P of P and the bit $\widetilde{y_P}$. Then y_P can be recovered as follows.

1. Compute the field element $\alpha = {x_P}^3 + ax_P + b \bmod p$.
2. Compute a square root β of α modulo p (see Section B.4).
3. If the least significant bit of β is equal to $\widetilde{y_P}$ then set $y_P \longleftarrow \beta$. Otherwise, set $y_P \longleftarrow p - \beta$.

4.2.4.2 Point compression (elliptic curves over $\mathbb{F}_{2^m}$)

The technique described in this subsection works only if the elements of the field $\mathbb{F}_{2^m}$ are represented with respect to an optimal normal basis representation.

Let $P = (x_P, y_P)$ be a point on the elliptic curve $E : y^2 + xy = x^3 + ax^2 + b$ defined over a field $\mathbb{F}_{2^m}$. Then $\widetilde{y_P}$ is defined to be 0 if $x_P = 0$; if $x_P \neq 0$ then $\widetilde{y_P}$ is defined to be the least significant bit of the field element $y_P \cdot {x_P}^{-1}$.

Suppose that we are given the x-coordinate x_P of P and the bit $\widetilde{y_P}$. Then y_P can be recovered as follows.

1. If $x_P = 0$ then y_P is obtained by cyclically shifting the vector representation of the field element b one position to the left. That is, if $b = b_{m-1}b_{m-2}\ldots b_1b_0$, then $y_P \longleftarrow b_{m-2}\ldots b_1b_0b_{m-1}$.
2. If $x_P \neq 0$ then do the following:

   2.1 Compute the field element $\alpha = x_P + a + b{x_P}^{-2}$ in $\mathbb{F}_{2^m}$.

   2.2 Let the vector representation of α be $\alpha = \alpha_{m-1}\alpha_{m-2}\ldots\alpha_1\alpha_0$.

   2.3 Construct a field element $z = z_{m-1}z_{m-2}\ldots z_1z_0$ by setting

$$
\begin{aligned}
z_0 &= \widetilde{y_P}, \\
z_1 &= \alpha_0 \oplus z_0, \\
z_2 &= \alpha_1 \oplus z_1, \\
&\vdots \\
z_{m-2} &= \alpha_{m-3} \oplus z_{m-3}, \\
z_{m-1} &= \alpha_{m-2} \oplus z_{m-2}.
\end{aligned}
$$

2.4 Finally, compute $y_P \longleftarrow x_P \cdot z$.

4.2.5 Key Length Considerations

4.3 Elliptic Curve Encryption Scheme (ECES)

4.3.1 Encryption

This section describes the ECES encryption process.

The encryption process consists of four steps: encryption-block formatting, elliptic curve computations, data inclusion, and point-to-octet-string conversion.

The input to the encryption process is:

- An octet string M, the data. The length of the data M shall not be more than $l-2$ octets; l is the length of the field size (q) in octets, that is,
$$l = \left\lceil \frac{t}{8} \right\rceil, \quad \text{where } t = \lceil \log_2 q \rceil.$$
- Two field elements a and b which describe the elliptic curve equation. A field element is represented by a binary string of length t bits.
- Two field elements x_P and y_P which together describe the point $P = (x_P, y_P)$ of order n.
- Two field elements x_Q and y_Q which together describe the public-key point $Q = (x_Q, y_Q)$.

The output from the encryption process shall be an octet string EM, the encrypted data. The length of EM is $2l+1$ octets if point compression is used, and $3l+1$ octets if point compression is not used.

4.3.1.1 Encryption-block formatting

1. Pad the data M on the left with a padding string PS consisting of $l-2-\|M\|$ octets each having the value ff, followed by the 00 octet, to form the octet string M' of length $l-1$:
$$M' = \mathrm{PS} \,\|\, 00 \,\|\, M.$$
Note that PS may be null if the length of M is $l-2$ octets.

Notes

(a) In typical applications of this standard t will be at least 100. Thus the value of l is at least 13, and so the length of the data M can always be up to 11 octets.

(b) Since the padding string PS contains no 00 octets, and the padding string is separated from the data M by a 00 octet, the encryption block can be parsed unambiguously.

4.3.1.2 Elliptic curve computations

1. Select a random integer k in the interval $[1, n-1]$.
2. Compute the elliptic curve point $(x_1, y_1) := kP$, where P is the point (x_P, y_P).
3. Compute the elliptic curve point $(x_2, y_2) := kQ$, where Q is the point (x_Q, y_Q).
4. If $x_2 = 0$ then go to step 1.

4.3.1.3 Data inclusion

1. Convert M' to a binary string, and then append a zero binary string of length $8-8l+t$ to the left of this binary string to form a field element m.
2. Perform a field multiplication to obtain $c := m \cdot x_2$.

Notes

(a) The value of $8 - 8l + t$ is either 1, 2, 3, 4, 5, 6, 7, or 8.

(b) The most significant bit of the field element m is 0. The reason for doing this is to ensure, in the case where q is equal to a prime p, that the integer represented by m is less than the modulus p.

4.3.2 Point-to-octet-string conversion

1. Append a zero binary string of length $8l - t$ to the left of the field element x_1, and convert the resulting $8l$-bit binary string to an octet string X_1 of length l.
2. Compute the value $\widetilde{y_1}$.
3. If point compression is used then assign the single octet PC the value 01 if $\widetilde{y_1}$ is 0, or the value 03 if $\widetilde{y_1}$ is 1.
   If point compression is not used then do the following:
   (a) Assign the single octet PC the value 00.
   (b) Append a zero binary string of length $8l - t$ to the left of the field element y_1, and convert the resulting $8l$-bit binary string to an octet string Y_1 of length l.
4. Append a zero binary string of length $8l - t$ to the left of the field element c, and convert the resulting $8l$-bit binary string to an octet string C of length l.
5. Finally, obtain the ciphertext EM by concatenating the four octet strings X_1, PC, Y_1, and C (if point compression is used then Y_1 is the null string):

$$\mathrm{EM} = X_1 \,\|\, \mathrm{PC} \,\|\, Y_1 \,\|\, C.$$

   EM is $2l + 1$ octets in length if point compression is used, and $3l + 1$ octets long otherwise.

4.3.3 Decryption

This section describes the ECES decryption process.

The decryption process consists of four steps: Octet-string-to-point coversion, elliptic curve computations, data extraction, and encryption-block parsing.

The input to the decryption process is:

- An octet string EM, of length either $2l+1$ or $3l+1$, the encrypted data.
- Two field elements a and b which describe the elliptic curve equation.
- An integer d, the private key.

The output from the encryption process shall be an octet string M of length at most $l-2$, the data.

4.3.3.1 Octet-string-to-point conversion

1. Parse the encrypted message EM to obtain octet strings X_1, PC, Y_1, and C.

   $$\mathrm{EM} = X_1 \,\|\, \mathrm{PC} \,\|\, Y_1 \,\|\, C.$$

   X_1 and C are each l octets in length. PC is a single octet. If the least significant bit of PC is 1 then Y_1 is the null octet; otherwise Y_1 is l octets in length.
2. Convert X_1 to a binary string, and then discard the $8l-t$ most significant bits to obtain a field element x_1.
3. If the least significant bit of PC is 1 then set the bit $\widetilde{y_1}$ to be equal to the second least significant bit of PC.
   If the least significant bit of PC is 0 then convert Y_1 to a binary string, and then discard the $8l-t$ most significant bits to obtain the binary string $\widetilde{y_1}$.
4. Convert C to a binary string, and then discard the $8l-t$ most significant bits to obtain a field element c.

4.3.3.2 Elliptic curve computations

1. Use x_1 and $\widetilde{y_1}$ to obtain the elliptic curve point (x_1, y_1).
2. Compute the elliptic curve point $(x_2, y_2) := d(x_1, y_1)$.

4.3.3.3 Data extraction

1. Compute $m := c \cdot x_2^{-1}$.
2. Discard the $8-8l+t$ most significant bits of m, and convert the resulting binary string to an octet string M' of length $l-1$.

4.3.3.4 Encryption-block parsing

1. Parse M' to obtain the data M:

$$M' = \mathrm{PS} \,\|\, 00 \,\|\, M.$$

4.4 Elliptic Curve Signature Scheme (ECSS)

4.4.1 Signature Generation

This section describes the ECSS (elliptic curve signature scheme) signature generation process.

The signature generation process consists of four steps: message digesting, elliptic curve computations, modular computations, and integer-to-octet-string conversion.

The input to the signature process is:

- An octet string M, the message, of arbitrary length.
- Two field elements a and b which describe the elliptic curve equation. A field element is represented by a binary string of length t bits, or by an octet string of length $l = \lceil \frac{t}{8} \rceil$.
- Two field elements x_P and y_P, which together describe the point $P = (x_P, y_P)$.
- An integer n, the order of P. An integer in the interval $[0, n-1]$ is represented by a binary string of length $h = \lceil \log_2 n \rceil$ bits, or by an octet string of length $f = \lceil \frac{h}{8} \rceil$.
- An integer d, the private key.
- A cryptographic hash function H.

The output from the signature generation process shall be an octet string SI of length $l + f$, the signature for M.

4.4.1.1 Message digesting

1. Convert M to a binary string M'.
2. Compute the hash value $e := H(M')$. It is assumed that e is represented as a binary string. If H is the Secure Hash Algorithm (SHA-1), then the hash value e is a binary string of length 160.

4.4.1.2 Elliptic curve computations

1. Select a random integer k in the interval $[1, n-1]$.
2. Compute the elliptic curve point $(x_1, y_1) := kP$, where P is the point (x_P, y_P).

4.4.1.3 Modular computations

1. Convert the binary string e to an integer $\overline{e}$.
2. Convert the binary string x_1 of length t bits to an integer $\overline{x}_1$.
3. Compute $r := \overline{x}_1 + \overline{e} \bmod q$.
4. Compute $s := k - dr \bmod n$.

4.4.1.4 Integer-to-octet-string conversion

1. Convert r to a binary string of length t. Append a zero binary string of length $8l - t$ to its left, and convert the resulting $8l$-bit binary string to an octet string R of length l.
2. Convert s to a binary string of length h. Append a zero binary string of length $8f - h$ to its left, and convert the resulting $8f$-bit binary string to an octet string S of length f.
3. The signature of M is the octet string
$$\mathrm{SI} := R \,\|\, S$$
of length $l + f$.

4.4.2 Signature Verification

This section describes the ECSS signature verification process.

The signature verification process consists of 4 steps: octet-string-to-integer conversion, message digesting, elliptic curve computations, and signature checking.

The input to the signature verification process is:

- An octet string M, the message.
- An octet string SI, the signature for M.
- Two field elements a and b which describe the elliptic curve equation.
- Two field elements x_P and y_P, which together describe the point $P = (x_P, y_P)$.
- Two field element x_Q and y_Q, which together describe the public-key point $Q = (x_Q, y_Q)$.

4.4.2.1 Octet-string-to-integer conversion

1. Parse the octet string SI to obtain on octet string R of length l and an octet string S of length f
$$\mathrm{SI} := R \,\|\, S.$$
2. Convert R to a binary string of length $8l$. Delete its $8l - t$ most significant bits and convert the result to an integer r.
3. Convert S to a binary string of length $8f$. Delete its $8f - h$ most significant bits and convert the result to an integer s.

4.4.2.2 Message digesting

1. Convert M to a binary string M'.
2. Compute the hash value $e = H(M')$. It is assumed that the hash value e is represented as a binary string; if the hash function used is SHA-1, then e is a binary string of length 160.

4.4.2.3 Elliptic curve computations

1. Compute the elliptic curve point $(x_1, y_1) := sP + rQ$ where $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$.

4.4.2.4 Signature checking

1. Convert the binary string e to an integer $\overline{e}$.
2. Convert the binary string x_1 to an integer $\overline{x}_1$.
3. Compute $r' := \overline{x}_1 + \overline{e} \bmod q$.
4. Accept the signature as valid if and only if the integers r and r' are equal.

4.5 Elliptic Curve Digital Signature Algorithm (ECDSA)

4.5.1 Signature Generation

This section describes the ECDSA signature generation process.

The signature generation process consists of four steps: message digesting, elliptic curve computations, modular computations, and integer-to-octet string conversion.

The input to the signature process is:

- An octet string M, the message, of arbitrary length.
- Two field elements a and b which describe the elliptic curve equation. A field element is represented by a binary string of length t.
- Two field element x_P and y_P, which together describe the point $P = (x_P, y_P)$.
- An integer n, the order of P. An integer in the interval $[0, n-1]$ is represented by a binary string of length $h = \lceil \log_2 n \rceil$ bits, or by an octet string of length $f = \lceil \frac{h}{8} \rceil$.
- An integer d, the private key.
- A cryptographic hash function H.

The output from the signature generation process shall be an octet string SI length $2f$, the signature for M.

4.5.1.1 Message digesting

1. Convert M to a binary string M'.
2. Compute the hash value $e := H(M')$. It is assumed that e is represented as a binary string. If H is the Secure Hash Algorithm (SHA-1), then the hash value e is a binary string of length 160.

4.5.1.2 Elliptic curve computations

1. Select a random integer k in the interval $[1, n-1]$.
2. Compute the elliptic curve point $(x_1, y_1) := kP$, where P is the point (x_P, y_P).
3. Convert the binary string x_1 of length t to an integer $\overline{x_1}$.
4. Set $r := \overline{x_1} \bmod n$.

Notes

If $r = 0$ then signature verification will fail. However, if k is chosen randomly, the probability that $r = 0$ is negligibly small.

4.5.1.3 Modular computations

1. Convert the binary string e to an integer $\overline{e}$.
2. Compute $s := k^{-1}(\overline{e} + rd) \bmod n$. computations.

Notes

If $s = 0$ then signature verification will fail. However, if k is chosen randomly, the probability that $s = 0$ is negligibly small.

4.5.1.4 Integer-to-octet-string conversion

1. Convert r to a binary string of length h. Append a zero binary string of length $8f - h$ to its left, and convert the resulting $8f$-bit binary string to an octet string S of length f.
2. Convert s to a binary string of length h. Append a zero binary string of length $8f - h$ to its left, and convert the resulting $8f$-bit binary string to an octet string S of length f.
3. The signature of M is the octet string

$$\mathrm{SI} := R \,\|\, S$$

of length $2f$.

4.5.2 Signature Verification

This section describes the ECDSA signature verification process.

The signature verification process consists of 4 steps: octet-string-to-integer conversion, message digesting, elliptic curve computations, and signature checking.

The input to the signature verification process is:

- An octet string M, the message.
- An octet string SI, the signature for M.
- Two field elements a and b which describe the elliptic curve equation.
- Two field elements x_P and y_P, which together describe the point $P = (x_P, y_P)$.
- An integer n, the order of P.
- Two field elements x_Q and y_Q, which together describe the public-key point $Q = (x_Q, y_Q)$.

4.5.2.1 Octet-string-to-integer conversion

1. Parse the octet string SI to obtain two octet string R and S each of length f:
$$\mathrm{SI} := R \,\|\, S.$$
2. Convert R to a binary string of length $8f$. Delete its $8f - h$ most significant bits and convert the result to an integer r.
3. Convert S to a binary string of length $8f$. Delete its $8f - h$ most significant bits and convert the result to an integer s.

4.5.2.2 Message digesting

1. Convert M to a binary string M'.
2. Compute the hash value $e = H(M')$. It is assumed that the hash value e is represented as a binary string; if the hash function used is SHA-1, then e is a binary string of length 160.

4.5.2.3 Elliptic curve computations

1. Compute $r \bmod n$; if this is 0 then reject the signature.
2. Convert the binary string e to an integer $\overline{e}$.
3. Compute $s^{-1} \bmod n$.
4. Compute $u := s^{-1}\overline{e} \bmod n$ and $v := s^{-1}r \bmod n$.
5. Compute the elliptic curve point $(x_1, y_1) := uP + vQ$ where $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$.

4.5.2.4 Signature checking

1. Convert the binary string x_1 of length t to an integer $\overline{x_1}$.
2. Set $r' := \overline{x_1} \bmod n$.
3. Accept the signature as valid if and only if the integers r and r' are equal.

4.6 Elliptic Curve Key Agreement Protocol (ECKEP)

Appendix A

Mathematical Background (Informative)

A.1 The Finite Field $\mathbb{F}_p$

Let p be a prime number. The *finite field* $\mathbb{F}_p$ is comprised of the set of integers $\{0, 1, 2, \ldots, p-1\}$ with the following arithmetic operations:

- *Addition:* If $a, b \in \mathbb{F}_p$, then $a + b = r$ where r is the remainder when the integer $a + b$ is divided by p, $0 \leq r \leq p - 1$.
- *Multiplication:* If $a, b \in \mathbb{F}_p$, then $a \cdot b = s$ where s is the remainder when the integer ab is divided by p, $0 \leq s \leq p - 1$.

Let $\mathbb{F}_p^*$ denote all the non-zero elements in $\mathbb{F}_p$. In $\mathbb{F}_p$, there exists at least one element g such that any non-zero element of $\mathbb{F}_p$ can be expressed as a power of g. Such an element g is called a *generator* (or *primitive element*) of $\mathbb{F}_p$. That is

$$\mathbb{F}_p^* = \{g^i : 0 \leq i \leq p - 2\}.$$

The *multiplicative inverse* of $a = g^i \in \mathbb{F}_p^*$ is

$$\frac{1}{a} = g^{-i} = g^{(-i) \bmod (p-1)}.$$

Example A.1 (*The finite field* $\mathbb{F}_2$)
$\mathbb{F}_2$=$\{$0,1$\}$. The addition and multiplication tables for $\mathbb{F}_2$ are

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Example A.2 (*The finite field* $\mathbb{F}_{23}$)
$\mathbb{F}_{23} = \{0, 1, 2, \ldots, 22\}$. Examples of the arithmetic operations in $\mathbb{F}_{23}$ are $12 + 20 = 32 = 9$, $8 \cdot 9 = 72 = 3$. The element 5 is a generator of $\mathbb{F}_{23}$. The powers of 5 are:

$5^0 = 1$	$5^1 = 5$	$5^2 = 2$	$5^3 = 10$	$5^4 = 4$	$5^5 = 20$
$5^6 = 8$	$5^7 = 17$	$5^8 = 16$	$5^9 = 11$	$5^{10} = 9$	$5^{11} = 22$
$5^{12} = 18$	$5^{13} = 21$	$5^{14} = 13$	$5^{15} = 19$	$5^{16} = 3$	$5^{17} = 15$
$5^{18} = 6$	$5^{19} = 7$	$5^{20} = 12$	$5^{21} = 14$	$5^{22} = 1.$	

A.2 The Finite Field $\mathbb{F}_{2^m}$

There are many ways to construct a finite field with a prime power number of elements. The method described in this section uses polynomial addition, multiplication, division and remainder.

Let $f(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_2x^2 + f_1x + f_0$, $f_i \in \mathbb{F}_2$, be an irreducible polynomial of degree m over $\mathbb{F}_2$, i.e., $f(x)$ cannot be factored into two polynomials over $\mathbb{F}_2$ each of degree less than m. The *finite field* $\mathbb{F}_{2^m}$ is comprised of all polynomials over $\mathbb{F}_2$ of degree less than m:

$$\mathbb{F}_{2^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0 : a_i \in \{0, 1\}\}.$$

The field element $(a_{m-1}x^{m-1} + \cdots + a_1x + a_0)$ is usually denoted by the binary string $(a_{m-1} \cdots a_1a_0)$ of length m, so that

$$\mathbb{F}_{2^m} = \{(a_{m-1} \cdots a_1a_0) \ : \ a_i \in \{0, 1\}\}.$$

Thus the elements of $\mathbb{F}_{2^m}$ can be represented by the set of all binary strings of length m.

Field elements are added and multiplied as follows:

- *Field addition:* $(a_{m-1} \cdots a_1a_0) + (b_{m-1} \cdots b_1b_0) = (c_{m-1} \cdots c_1c_0)$, where $c_i = a_i + b_i$ in the field $\mathbb{F}_2$. That is, field addition is performed componentwise.
- *Field multiplication:* $(a_{m-1} \cdots a_1a_0) \cdot (b_{m-1} \cdots b_1b_0) = (r_{m-1} \cdots r_1r_0)$, where the polynomial $(r_{m-1}x^{m-1} + \cdots + r_1x + r_0)$ is the remainder when the polynomial $(a_{m-1}x^{m-1} + \cdots + a_1x + a_0) \cdot (b_{m-1}x^{m-1} + \cdots + b_1x + b_0)$ is divided by $f(x)$ over $\mathbb{F}_2$.

This method of representing $\mathbb{F}_{2^m}$ is called a *polynomial basis representation*.

Note that $\mathbb{F}_{2^m}$ contains exactly 2^m elements. Let $\mathbb{F}^*_{2^m}$ denote the set of all non-zero elements in $\mathbb{F}_{2^m}$. There exists at least one element g in $\mathbb{F}_{2^m}$ such that any non-zero element of $\mathbb{F}_{2^m}$ can be expressed as a power of g. Such an element g is called a *generator* (or *primitive element*) of $\mathbb{F}_{2^m}$. That is

$$\mathbb{F}^*_{2^m} = \{g^i : 0 \leq i \leq 2^m - 2\}$$

The *multiplicative inverse* of $a = g^i \in \mathbb{F}^*_{2^m}$ is

$$\frac{1}{a} \ = \ g^{-i} \ = \ g^{(-i) \bmod (2^m - 1)}.$$

Example A.3 (*The finite field $\mathbb{F}_{2^4}$ using a polynomial basis representation*)
Take $f(x) = x^4 + x + 1$ over $\mathbb{F}_2$; it can be verified that $f(x)$ is irreducible over $\mathbb{F}_2$. Then the elements of $\mathbb{F}_{2^4}$ are:

$$\begin{array}{cccccccc} (0000) & (0001) & (0010) & (0011) & (0100) & (0101) & (0110) & (0111) \\ (1000) & (1001) & (1010) & (1011) & (1100) & (1101) & (1110) & (1111). \end{array}$$

As examples of field arithmetic, we have $(1011) + (1001) = (0010)$, and

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

i.e., the remainder when $(x^3 + x^2 + 1)(x^3 + 1)$ is divided by $f(x)$ is $x^3 + x^2 + x + 1$.

The multiplicative identity is (0001).

$\mathbb{F}_{2^4}$ can be generated by one element $\alpha = x$. The powers of α are:

$$\begin{array}{llll} \alpha^0 = (0001) & \alpha^1 = (0010) & \alpha^2 = (0100) & \alpha^3 = (1000) \\ \alpha^4 = (0011) & \alpha^5 = (0110) & \alpha^6 = (1100) & \alpha^7 = (1011) \\ \alpha^8 = (0101) & \alpha^9 = (1010) & \alpha^{10} = (0111) & \alpha^{11} = (1110) \\ \alpha^{12} = (1111) & \alpha^{13} = (1101) & \alpha^{14} = (1001) & \alpha^{15} = \alpha^0 = (0001). \end{array}$$

More generally, the field $\mathbb{F}_{2^m}$ can be viewed as a vector space of dimension m over $\mathbb{F}_2$. That is, there exists a set of m elements $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ in $\mathbb{F}_{2^m}$ such that each $\alpha \in \mathbb{F}_{2^m}$ can be written uniquely in the form

$$\alpha = a_0\alpha_0 + a_1\alpha_1 + \cdots + a_{m-1}\alpha_{m-1}, \quad \text{where } a_i \in \{0, 1\}.$$

We can then represent α as the 0-1 vector $(a_0 a_1 \ldots a_{m-1})$. Addition of field elements is performed by bitwise XOR-ing the vector representations.

In general, there are many different bases of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. A *normal basis* of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is a basis of the form

$$\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\},$$

where $\beta \in \mathbb{F}_{2^m}$; it is a well-known fact that such a basis always exists. Given any element $\alpha \in \mathbb{F}_{2^m}$, we can write $\alpha = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in \{0, 1\}$. Since squaring is a linear operator in $\mathbb{F}_{2^m}$, we have

$$\alpha^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1} \beta^{2^i} = (a_{m-1} a_0 \ldots a_{m-2}),$$

with indices reduced modulo m. Hence a normal basis representation of $\mathbb{F}_{2^m}$ is advantageous because squaring a field element can then be accomplished by a simple rotation of the vector representation, an operation that is easily implemented in hardware.

Multiplication can be easily implemented in hardware when the normal basis used is of a special type called an *optimal normal basis*.

A.3 Optimal Normal Bases in $\mathbb{F}_{2^m}$

In Example A.3, the field $\mathbb{F}_{2^4}$ was described using polynomial multiplication, division and remainders. A less insightful way to construct the field $\mathbb{F}_{2^4}$, but one which is computationally nicer, is an optimal normal basis representation (see Section 4.2.3.2).

Example A.4 (*The finite field $\mathbb{F}_{2^4}$ using an optimal normal basis representation*)
As in the example A.3, the elements of $\mathbb{F}_{2^4}$ will be the set of all binary 4-tuples

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

Field elements are added and multiplied as follows:

- *Field addition:* $(a_0a_1a_2a_3) + (b_0b_1b_2b_3) = (c_0c_1c_2c_3)$, where $c_i = a_i + b_i$ in the field $\mathbb{F}_2$ (i.e., modulo 2). In other words, field addition is performed by simply XORing the vector representations.

- *Field multiplication:* The setup for multiplication is done as follows (cf. Section 4.2.3.2):

  1. $f(x) = x^4 + x^3 + x^2 + x + 1$.
  2. The matrix A is computed as:
  $$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$
  3. The inverse of A is:
  $$A^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
  4. The matrix T is computed as:
  $$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$
  5. The λ_{ij} terms which are 1 are: $\lambda_{0,2}$, $\lambda_{1,2}$, $\lambda_{1,3}$, $\lambda_{2,0}$, $\lambda_{2,1}$, $\lambda_{3,1}$, and $\lambda_{3,3}$.

  Hence multiplication is defined to be $(a_0a_1a_2a_3) \cdot (b_0b_1b_2b_3) = (c_0c_1c_2c_3)$, where
  $$\begin{aligned} c_0 &= a_0b_2 + a_1(b_2 + b_3) + a_2(b_0 + b_1) + a_3(b_1 + b_3) \\ c_1 &= a_1b_3 + a_2(b_3 + b_0) + a_3(b_1 + b_2) + a_0(b_2 + b_0) \\ c_2 &= a_2b_0 + a_3(b_0 + b_1) + a_0(b_2 + b_3) + a_1(b_3 + b_1) \\ c_3 &= a_3b_1 + a_0(b_1 + b_2) + a_1(b_3 + b_0) + a_2(b_0 + b_2). \end{aligned}$$

With these definitions of addition and multiplication the 16 binary 4-tuples form a field, although this fact is not obvious.

Notes

(a) The multiplicative identity is (1111). (In Example A.3 it was (0001).)

(b) Notice that

$$\begin{aligned}(a_0a_1a_2a_3)\cdot(a_0a_1a_2a_3) &= (a_0a_1a_2a_3)^2\\ &= (a_3a_0a_1a_2),\end{aligned}$$

and so the square of a field element is simply a right cyclic shift of its vector representation.

(c) The formula for c_1 in the multiplication can be obtained by adding a 1 to each subscript in the formula for c_0 (where the subscripts are reduced modulo 4). The formula for c_2 can be obtained by adding 2 to each subscript in the formula for c_0 and reducing the subscripts modulo 4. The formula for c_3 can likewise be obtained.

(d) From the preceding remark we see that a circuit to compute c_0 from $(a_0a_1a_2a_3)$ and $(b_0b_1b_2b_3)$ can be used to compute c_1 by applying $(a_1a_2a_3a_0)$ and $(b_1b_2b_3b_0)$ to it. Similarly, c_2 and c_3 can be computed by shifting to the left the input vectors.

As an example of how to compute in this representation of $\mathbb{F}_{2^4}$, we have

$$(1001)\cdot(1101) = (c_0c_1c_2c_3)$$

where

$$\begin{aligned}c_0 &= (1)(0)+(0)(0+1)+(0)(1+1)+(1)(1+1)=0\\ c_1 &= (0)(1)+(0)(1+1)+(1)(1+0)+(1)(0+1)=0\\ c_2 &= (0)(1)+(1)(1+1)+(1)(0+1)+(0)(1+1)=1\\ c_3 &= (1)(1)+(1)(1+0)+(0)(1+1)+(0)(1+0)=0.\end{aligned}$$

That is, $(1001)\cdot(1101) = (0010)$.

Also,

$$\begin{aligned}(1010)^{10} &= (1010)^2\cdot(1010)^8\\ &= (0101)\cdot(0101)\\ &= (1010),\end{aligned}$$

and

$$\begin{aligned}(1101)^{9} &= (1101)\cdot(1101)^8\\ &= (1101)\cdot(1011)\\ &= (0001).\end{aligned}$$

In this representation, $\mathbb{F}_{2^4}$ can be generated by the powers of $\alpha = (1100)$:

$\alpha^0 = (1111)$	$\alpha^1 = (1100)$	$\alpha^2 = (0110)$	$\alpha^3 = (0100)$
$\alpha^4 = (0011)$	$\alpha^5 = (1010)$	$\alpha^6 = (0010)$	$\alpha^7 = (0111)$
$\alpha^8 = (1001)$	$\alpha^9 = (1000)$	$\alpha^{10} = (0101)$	$\alpha^{11} = (1110)$
$\alpha^{12} = (0001)$	$\alpha^{13} = (1101)$	$\alpha^{14} = (1011)$	$\alpha^{15} = \alpha^0 = (1111)$.

This method of representing $\mathbb{F}_{2^4}$ is called an *optimal normal basis representation.*

Arithmetic in the finite field $\mathbb{F}_{2^m}$ can be performed efficiently both in hardware and in software when the field elements are represented with respect to an optimal normal basis.

For further information on optimal normal bases, consult the references on page 49.

A.4 Elliptic Curves over $\mathbb{F}_p$

Let $p > 3$ be a prime number. Let $a, b \in \mathbb{F}_p$ be such that $4a^3 + 27b^2 \neq 0$ in $\mathbb{F}_p$. An *elliptic curve* $E(\mathbb{F}_p)$ over $\mathbb{F}_p$ defined by the parameters a and b is the set of solutions (x, y), $x \in \mathbb{F}_p$, $y \in \mathbb{F}_p$, to the equation

$$y^2 = x^3 + ax + b,$$

together with an extra point $\mathcal{O}$, the *point at infinity.* The number of points in $E(\mathbb{F}_p)$ is denoted by $\#E(\mathbb{F}_p)$. The Hasse Theorem tells us that

$$p + 1 - 2\sqrt{p} \;\leq\; \#E(\mathbb{F}_p) \;\leq\; p + 1 + 2\sqrt{p}.$$

The set of points $E(\mathbb{F}_p)$ form a group with respect to the following addition rules:

(i) $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

(ii) $(x, y) + \mathcal{O} = (x, y)$ for all $(x, y) \in E(\mathbb{F}_p)$.

(iii) $(x, y) + (x, -y) = \mathcal{O}$ for all $(x, y) \in E(\mathbb{F}_p)$ (i.e., the inverse of the point (x, y) is the point $(x, -y)$).

(iv) (Rule for adding two distinct points that are not inverses of each other)
Let $(x_1, y_1) \in E(\mathbb{F}_p)$ and $(x_2, y_2) \in E(\mathbb{F}_p)$ be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{and} \quad \lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

(v) (Rule for doubling a point)
Let $(x_1, y_1) \in E(\mathbb{F}_p)$ be a point with $y_1 \neq 0$. Then $2(x_1, y_1) = (x_3, y_3)$, where

$$x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{and} \quad \lambda = \frac{3x_1^2 + a}{2y_1}.$$

The group $E(\mathbb{F}_p)$ is *abelian*, which means that $P + Q = Q + P$ for all points P and Q in $E(\mathbb{F}_p)$. The curve is said to be *supersingular* if $\#E(\mathbb{F}_p) = p + 1$; otherwise it is *non-supersingular.*

Example A.5 (*An elliptic curve over $\mathbb{F}_{23}$*)
Let $y^2 = x^3 + x + 1$ be an equation over $\mathbb{F}_{23}$ (i.e., $a = 1$ and $b = 1$). Then the solutions over $\mathbb{F}_{23}$ to the equation of the elliptic curve are:

(0,1)	(0,22)	(1,7)	(1,16)	(3,10)	(3,13)	(4,0)	(5,4)	(5,19)
(6,4)	(6,19)	(7,11)	(7,12)	(9,7)	(9,16)	(11,3)	(11,20)	(12,4)
(12,19)	(13,7)	(13,16)	(17,3)	(17,20)	(18,3)	(18,20)	(19,5)	(19,18).

The group $E(\mathbb{F}_{23})$ has 28 points (including the point at infinity $\mathcal{O}$). The following are examples of the group operation.

1. Let $P_1 = (x_1, y_1) = (3, 10)$, $P_2 = (x_2, y_2) = (9, 7)$, $P_1 + P_2 = (x_3, y_3)$. Compute

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7 - 10}{9 - 3} = \frac{-3}{6} = \frac{-1}{2} = 11 \in \mathbb{F}_{23},$$

$$x_3 = \lambda^2 - x_1 - x_2 = 11^2 - 3 - 9 = 6 - 3 - 9 = -6 = 17,$$

$$y_3 = \lambda(x_1 - x_3) - y_1 = 11(3 - 17) - 10 = 11(9) - 10 = 89 = 20.$$

   Therefore $P_1 + P_2 = (17, 20)$.

2. Let $P_1 = (x_1, y_1) = (3, 10)$, $2P_1 = (x_3, y_3)$. Compute

$$\lambda = \frac{3x_1^2 + a}{2y_1} = \frac{3(3^2) + 1}{20} = \frac{5}{20} = \frac{1}{4} = 6,$$

$$x_3 = \lambda^2 - 2x_1 = 6^2 - 6 = 30 = 7,$$

$$y_3 = \lambda(x_1 - x_3) - y_1 = 6(3 - 7) - 10 = -24 - 10 = -11 = 12.$$

   Therefore $2P_1 = (7, 12)$.

A.5 Elliptic Curves over $\mathbb{F}_{2^m}$

A non-supersingular *elliptic curve* $E(\mathbb{F}_{2^m})$ over $\mathbb{F}_{2^m}$ defined by the parameters $a, b \in \mathbb{F}_{2^m}$, $b \neq 0$, is the set of solutions (x, y), $x \in \mathbb{F}_{2^m}$, $y \in \mathbb{F}_{2^m}$, to the equation

$$y^2 + xy = x^3 + ax^2 + b$$

together with an extra point $\mathcal{O}$, the *point at infinity*. The number of points in $E(\mathbb{F}_{2^m})$ is denoted by $\#E(\mathbb{F}_{2^m})$. The Hasse Theorem tells us that

$$q + 1 - 2\sqrt{q} \ \le\ \#E(\mathbb{F}_{2^m}) \ \le\ q + 1 + 2\sqrt{q},$$

where $q = 2^m$. Furthermore, $\#E(\mathbb{F}_{2^m})$ is even.

The set of points $E(\mathbb{F}_{2^m})$ is a group with respect to the following addition rules:

(i) $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

(ii) $(x,y)+\mathcal{O}=(x,y)$ for all $(x,y)\in E(\mathbb{F}_{2^m})$.

(iii) $(x,y)+(x,x+y)=\mathcal{O}$ for all $(x,y)\in E(\mathbb{F}_{2^m})$ (i.e., the inverse of the point (x,y) is the point $(x,x+y)$).

(iv) (Rule for adding two distinct points that are not inverses of each other)
Let $(x_1,y_1)\in E(\mathbb{F}_{2^m})$ and $(x_2,y_2)\in E(\mathbb{F}_{2^m})$ be two points such that $x_1\neq x_2$. Then $(x_1,y_1)+(x_2,y_2)=(x_3,y_3)$, where

$$x_3=\lambda^2+\lambda+x_1+x_2+a, \quad y_3=\lambda(x_1+x_3)+x_3+y_1, \quad \text{and} \quad \lambda=\frac{y_1+y_2}{x_1+x_2}.$$

(v) (Rule for doubling a point)
Let $(x_1,y_1)\in E(\mathbb{F}_{2^m})$ be a point with $x_1\neq 0$. Then $2(x_1,y_1)=(x_3,y_3)$, where

$$x_3=\lambda^2+\lambda+a, \quad y_3=x_1^2+(\lambda+1)x_3, \quad \text{and} \quad \lambda=\left(x_1+\frac{y_1}{x_1}\right).$$

The group $E(\mathbb{F}_{2^m})$ is *abelian*, which means that $P+Q=Q+P$ for all points P and Q in $E(\mathbb{F}_{2^m})$.

We now give two examples of elliptic curves over $\mathbb{F}_{2^4}$ to illustrate. The first uses a polynomial basis representation for the field, and the second uses an optimal normal basis representation.

Example A.6 (*An elliptic curve over $\mathbb{F}_{2^4}$. A polynomial basis representation is used for the elements of $\mathbb{F}_{2^4}$ – see Example A.3.*)
Consider the field $\mathbb{F}_{2^4}$ generated by the root $\alpha=x$ of the irreducible polynomial $f(x)=x^4+x+1$. Consider the non-supersingular elliptic curve over $\mathbb{F}_{2^4}$ with defining equation

$$y^2+xy=x^3+\alpha^4x^2+1.$$

(Note that to be absolutely precise with our notation, we should write this equation as

$$(0001)y^2+(0001)xy=(0001)x^3+(0011)x^2+(0001).$$

Since (0001) is the multiplicative identity, we simplify our notation as above.)

Then the solutions over $\mathbb{F}_{2^4}$ to the equation of the elliptic curve are:

$$\begin{array}{cccccccc} & (0,1) & (1,\alpha^6) & (1,\alpha^{13}) & (\alpha^3,\alpha^8) & (\alpha^3,\alpha^{13}) & (\alpha^5,\alpha^3) & (\alpha^5,\alpha^{11}) \\ (\alpha^6,\alpha^8) & (\alpha^6,\alpha^{14}) & (\alpha^9,\alpha^{10}) & (\alpha^9,\alpha^{13}) & (\alpha^{10},\alpha^1) & (\alpha^{10},\alpha^8) & (\alpha^{12},0) & (\alpha^{12},\alpha^{12}). \end{array}$$

The group $E(\mathbb{F}_{2^4})$ has 16 points (including the point at infinity $\mathcal{O}$). The following are examples of the group operation.

1. Let $P_1=(x_1,y_1)=(\alpha^6,\alpha^8)$, $P_2=(x_2,y_2)=(\alpha^3,\alpha^{13})$, and let $P_1+P_2=(x_3,y_3)$. Then

$$\lambda=\frac{y_1+y_2}{x_1+x_2}=\frac{\alpha^8+\alpha^{13}}{\alpha^6+\alpha^3}=\alpha,$$
$$x_3=\lambda^2+\lambda+x_1+x_2+a=\alpha^2+\alpha+\alpha^6+\alpha^3+\alpha^4=1,$$
$$y_3=\lambda(x_1+x_3)+x_3+y_1=\alpha(\alpha^6+1)+1+\alpha^8=\alpha^{13}.$$

2. If $2P_1 = (x_3, y_3)$, then

$$\lambda = x_1 + \frac{y_1}{x_1} = \alpha^6 + \frac{\alpha^8}{\alpha^6} = \alpha^3,$$

$$x_3 = \lambda^2 + \lambda + a = \alpha^6 + \alpha^3 + \alpha^4 = \alpha^{10},$$

$$y_3 = x_1^2 + (\lambda + 1)x_3 = \alpha^{12} + (\alpha^3 + 1)\alpha^{10} = \alpha^8.$$

Example A.7 (*An elliptic curve over $\mathbb{F}_{2^4}$. An optimal normal basis representation is used for the elements of $\mathbb{F}_{2^4}$.*)
Consider the field $\mathbb{F}_{2^4}$ given by the optimal normal basis representation in Example A.4. That is, the elements of $\mathbb{F}_{2^4}$ are the set of all binary 4-tuples with multiplication given by the formulae. Recall that $\alpha = (1100)$ is a generator for the non-zero elements and (1111) is the multiplicative identity.

Consider the non-supersingular curve over $\mathbb{F}_{2^4}$ defined by the equation

$$E: \; y^2 + xy = x^3 + \alpha^3.$$

(Note that to be absolutely precise with our notation, we should write this equation as

$$(1111)y^2 + (1111)xy = (1111)x^3 + (0100).$$

Since (1111) is the multiplicative identity, we simplify our notation as above.)

The solutions over $\mathbb{F}_{2^4}$ to the elliptic curve equation are:

$$\begin{array}{lllllll}
(0, \alpha^9) & (\alpha, 0) & (\alpha, \alpha) & (\alpha^3, \alpha^5) & (\alpha^3, \alpha^{11}) & (\alpha^4, \alpha^3) & (\alpha^4, \alpha^7) \\
(\alpha^5, \alpha^3) & (\alpha^5, \alpha^{11}) & (\alpha^6, 0) & (\alpha^6, \alpha^6) & (\alpha^8, \alpha^3) & (\alpha^8, \alpha^{13}) & \\
(\alpha^{11}, 0) & (\alpha^{11}, \alpha^{11}) & (\alpha^{12}, \alpha^8) & (\alpha^{12}, \alpha^9) & (\alpha^{13}, \alpha^2) & (\alpha^{13}, \alpha^{14}). &
\end{array}$$

Since there are 19 solutions to the equation in $\mathbb{F}_{2^4}$, the group $E(\mathbb{F}_{2^4})$ has $19 + 1 = 20$ elements. This group turns out to be a cyclic group of order 20. If we take $G = (\alpha^3, \alpha^5)$ and use the addition formulae, we find that

$$\begin{array}{lllll}
1G = (\alpha^3, \alpha^5) & 2G = (\alpha^4, \alpha^3) & 3G = (\alpha^{13}, \alpha^2) & 4G = (\alpha, 0) & 5G = (\alpha^{12}, \alpha^8) \\
6G = (\alpha^8, \alpha^3) & 7G = (\alpha^{11}, 0) & 8G = (\alpha^5, \alpha^{11}) & 9G = (\alpha^6, 0) & 10G = (0, \alpha^9) \\
11G = (\alpha^6, \alpha^6) & 12G = (\alpha^5, \alpha^3) & 13G = (\alpha^{11}, \alpha^{11}) & 14G = (\alpha^8, \alpha^{13}) & 15G = (\alpha^{12}, \alpha^9) \\
16G = (\alpha, \alpha) & 17G = (\alpha^{13}, \alpha^{14}) & 18G = (\alpha^4, \alpha^7) & 19G = (\alpha^3, \alpha^{11}) & 20G = \mathcal{O}.
\end{array}$$

A.6 Examples of ECES, ECSS, ECDSA, ECKEP

Example A.8 This example illustrates the elliptic curve encryption scheme (ECES). For simplicity, the example follows the high-level description of ECES given in Section 4.1.2, but not all of the detailed description given in Section 4.3.

System Setup

The underlying finite will be $\mathbb{F}_{2^4}$ and the elliptic curve the one described in Example A.7. The point $P = (x_P, y_P) = (\alpha^5, \alpha^{11})$ is selected. Since $5P = \mathcal{O}$, the point P has order $n = 5$.

The system parameters (the public information) are the field $\mathbb{F}_{2^4}$, the curve E, the point P and the order $n = 5$.

Key Generation

Entity A performs the following operations.

1. A selects a random integer $d = 3$ in the interval $[1, 4]$.
2. A computes the point $Q = dP = 3P = (\alpha, 0) = ((1100), (0000))$.
3. A makes public the point Q.
4. A's private key is the integer $d = 3$.

Encryption Process

(Entity B sends data $M = 0111$ to entity A)

Entity B performs the following steps:

1. B looks up A's public key: $Q = (x_Q, y_Q) = (\alpha, 0)$.
2. B represents the data M as field element $m = \alpha^7$.
3. B selects a random integer $k = 2$ in the interval $[1, 4]$.
4. B computes the point $2P = (x_1, y_1) = (\alpha, \alpha) = ((1100), (1100))$.
5. B computes the point $2Q = (x_2, y_2) = (\alpha^5, \alpha^{11}) = ((1010), (1110))$.
6. B computes $c = m \cdot x_2 = \alpha^7 \cdot \alpha^5 = \alpha^{12} = (0001)$.
7. B transmits the encrypted data

$$\begin{aligned}(x_1, y_1, c) &= (\alpha, \alpha, \alpha^{12}) \\ &= ((1100), (1100), (0001))\end{aligned}$$

   to A.

Decryption Process

Entity A decrypts the ciphertext $C = (\alpha, \alpha, \alpha^{12})$ received from B.

Entity A performs the following steps.

1. A computes the point
$$d(x_1, y_1) = 3(\alpha, \alpha) = (\alpha^5, \alpha^{11}) = ((1010), (1110)) = (x_2, y_2).$$
2. A recovers the data $m = c \cdot x_2^{-1} = \alpha^{12} \cdot \alpha^{-5} = \alpha^7 = (0111)$.

Example A.9 This example illustrates the elliptic curve signature scheme (ECSS). For simplicity, the example follows the high-level description of ECSS given in Section 4.1.3, but not all of the detailed description given in Section 4.4. System setup and key generation for entity A are as in Example A.8.

Signature Generation for ECSS

Entity A signs message $M = 10110100101110$.

Suppose that the hash value of M is $H(M) = 15$. A performs the following steps.

1. Select a random integer $k = 4$ in the interval $[1, 4]$.
2. Compute $kP = 4P = (\alpha^5, \alpha^3) = (x_1, y_1)$.
3. Represent $x_1 = \alpha^5 = (1010)$ as the integer $8 + 2 = 10$.
4. Compute
$$\begin{aligned} r &= (x_1 + e) \bmod q \\ &= (10 + 15) \bmod 16 \\ &= 9. \end{aligned}$$
5. Use the private key $d = 3$ to compute
$$\begin{aligned} s &= (k - dr) \bmod n \\ &= (4 - 3 \cdot 9) \bmod 5 \\ &= 2. \end{aligned}$$
6. The signature on message M is $(r, s) = (9, 2)$.

Signature Verification for ECSS

Entity B verifies signature $(r, s) = (9, 2)$ on M as follows.

1. B looks up A's public key $Q = (\alpha, 0)$.
2. B computes the point
$$\begin{aligned} sP + rQ &= (\alpha, \alpha) + (\alpha, \alpha) \\ &= (\alpha^5, \alpha^3) \\ &= (x_1, y_1). \end{aligned}$$

3. B computes $H(M) = e = 15$.
4. B represents $x_1 = \alpha^5 = (1010)$ as the integer $8 + 2 = 10$.
5. B computes

$$\begin{aligned} r' &= (x_1 + e) \bmod q \\ &= (10 + 15) \bmod 16 \\ &= 9. \end{aligned}$$

6. B accepts A's signature on M since $r' = r = 9$.

Signature Generation for ECDSA

Entity A signs message $M = 11100011010111100$.

Suppose that the hash value of M is $H(M) = e = 7$. A performs the following steps.

1. Select a random integer $k = 2$ in the interval $[1, 4]$.
2. Compute $(x_1, y_1) = 2P = (\alpha, \alpha)$.
3. Represent $x_1 = \alpha = (1100)$ as the integer $8 + 4 = 12$.
4. Set $r = x_1 \bmod n = 12 \bmod 5 = 2$.
5. Compute $s = k^{-1}(e + rd) \bmod n = 3 \cdot (7 + 2 \cdot 3) \bmod 5 = 4$.
6. The signature on message M is $(r, s) = (2, 4)$.

Signature Verification for ECDSA

Entity B verifies signature $(r, s) = (2, 4)$ on M as follows.

1. B looks up A's public key $Q = (\alpha, 0)$.
2. B computes $H(M) = e = 7$.
3. B computes $s^{-1} \bmod n = 4^{-1} \bmod 5 = 4$.
4. B computes $u = s^{-1}e \bmod n = 3$ and $v = s^{-1}r \bmod n = 3$.
5. B computes the point $(x_1, y_1) = uP + vQ = 3P + 3Q = (\alpha, 0) + (\alpha^5, \alpha^3) = (\alpha, \alpha)$.
6. B represents $x_1 = \alpha = (1100)$ as the integer $8 + 4 = 12$.
7. B computes $r' = (x_1 \bmod n) = 2$.
8. B accepts the signature since $r' = r = 2$.

Appendix B

Algorithms (Informative)

B.1 Computing the Multiple of a Point

If k is a positive integer and P is an elliptic curve point, then kP is the point obtained by adding together k copies of P. This computation (called *exponentiation*) can be performed efficiently by the "repeated double-and-add" method outlined below.

Input: A positive integer k, and an elliptic curve point P.
Output: The elliptic curve point kP.

1. Let $k = k_r k_{r-1} \ldots k_1 k_0$ be the binary representation of k, where the most significant bit k_r of k is 1.
2. Set $Q \longleftarrow P$.
3. For i from $r-1$ downto 0 do

   3.1 Set $Q \longleftarrow 2Q$.
   3.2 If $k_i = 1$ then set $Q \longleftarrow Q + P$.

4. Output Q.

There are several variations of this method which can be used to speed up the computations. One such method which requires some precomputations is described in [5]. See also Knuth [11, pages 441-466].

B.2 Selecting an Appropriate Curve

An elliptic curve E over $\mathbb{F}_q$ is *suitable* for cryptographic purposes if the following conditions are satisfied:

1. The order $u = \#E(\mathbb{F}_q)$ of the curve should be divisible by a large prime number r. (See Appendix D for an explanation of "large".).

2. The large prime divisor r should not divide $q^v - 1$ for $v = 1, 2, 3, \ldots, 10$. This condition ensures that the reduction algorithm of Menezes, Okamoto and Vanstone [19] is not feasible.

There are three approaches to selecting such an elliptic curve over $\mathbb{F}_q$.

B.2.1 Method 1 – Selecting the curve at random

1. Randomly select parameters $a, b \in \mathbb{F}_q$ to define the elliptic curve equation.
   In the case that q is a prime, verify that $4a^3 + 27b^2 \neq 0$. The curve equation is $E : y^2 = x^3 + ax + b$.
   In the case that $q = 2^m$, verify that $b \neq 0$. The curve equation is $E : y^2 + xy = x^3 + ax^2 + b$.
2. Compute the order of the curve $u = \#E(\mathbb{F}_q)$. (See notes below.)
3. Factor u; if this order is not divisible by a large prime number r, then go to step 1.
4. Verify that the large prime divisor r of u does not divide $q^v - 1$, for $v = 1, 2, 3, \ldots, 10$. If this test fails, then go to step 1.
5. Output the curve E.

Notes

The order $\#E(\mathbb{F}_q)$ can be computed by using Schoof's algorithm [27]. Although the basic algorithm is quite inefficient, several dramatic improvements and extensions of this method have been discovered during the last three years. Currently it is feasible to compute orders of elliptic curves over $\mathbb{F}_p$ where p is as large as 10^{499}, and orders of elliptic curves over $\mathbb{F}_{2^m}$ where m is as large as 601 [15]. Of more relevance is that the orders of elliptic curves over the field $\mathbb{F}_{2^{155}}$ can now be computed in less than 4 minutes on a workstation [15].

B.2.2 Method 2 – Selecting the order of the curve first

Elliptic curves over $\mathbb{F}_p$

1. Select an order u such that

   (a) $p + 1 - 2\sqrt{p} \leq u \leq p + 1 + 2\sqrt{p}$.
   (b) u is divisible by a large prime r.
   (c) r does not divide $p^v - 1$ for $v = 1, 2, 3, \ldots, 10$.

2. Use the Atkin-Morain algorithm [21] to find parameters $a, b \in \mathbb{F}_p$ such that the elliptic curve E defined by them has order $\#E(\mathbb{F}_p) = u$.
3. Output the curve E.

Elliptic curves over $\mathbb{F}_{2^m}$

1. Select a small integer $c_{\max}$.
2. Use the Lay-Zimmer algorithm [14] to find $a, b \in \mathbb{F}_{2^m}$ such that the elliptic curve E defined by them has order $u = cr$, where r is a prime and $c \le c_{\max}$.
3. Verify that r does not divide $(2^m)^v - 1$ for $v = 1, 2, 3, \ldots, 10$. If this fails, then go to step 1.
4. Output the curve E.

The Lay-Zimmer algorithm [14] requires some precomputations for each particular field $\mathbb{F}_{2^m}$. Once this is done, then the algorithm takes a few minutes on a workstation, even when q is as large as 2^{300}.

B.2.3 Method 3 – Using the Weil Theorem

This technique can be used for picking curves over $\mathbb{F}_{2^m}$, when m is divisible by a small number, say l.

1. Select a random curve $E : y^2 + xy = x^3 + ax^2 + b$, $b \neq 0$, where $a, b \in \mathbb{F}_{2^l}$. Note that since $\mathbb{F}_{2^l}$ is contained in $\mathbb{F}_{2^m}$, it is also true that $a, b \in \mathbb{F}_{2^m}$, and so E is also a curve over $\mathbb{F}_{2^m}$.
2. Compute $w = \#E(\mathbb{F}_{2^l})$. This can easily be done exhaustively since l is small.
3. Let $t = q^l + 1 - w$, and let $c = m/l$. Then compute the c'th term of the sequence $\{V_n\}$ defined by
$$V_0 = 2, \quad V_1 = t, \quad V_n = tV_{n-1} - 2^l V_{n-2},\ n \ge 2.$$
Then set
$$u \ = \ \#E(\mathbb{F}_{2^m}) = 2^m + 1 - V_c.$$
4. Factor u; if this order is not divisible by a large prime number r, then go to step 1.
5. Verify that the large prime divisor r of u does not divide $q^v - 1$, for $v = 1, 2, 3, \ldots, 10$. If this test fails, then go to step 1.
6. Output the curve E.

B.3 Computing the Order of a Point

The following algorithm is an efficient method for computing the order of an elliptic curve point, given the prime factorization of the elliptic curve order.

Input: An elliptic curve E defined over $\mathbb{F}_q$, where the prime factorization of $\#E(\mathbb{F}_q)$ is
$$\#E(\mathbb{F}_q) = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}, \quad e_i \ge 1,$$
and a point P on E.
Output: The order n of P.

1. Set $n \longleftarrow \#E(\mathbb{F}_q)$.
2. For i from 1 to k do

   2.1 Set $n \longleftarrow n/p_i^{e_i}$.
   2.2 Compute $P_1 \longleftarrow nP$.
   2.3 While $P_1 \neq \mathcal{O}$, compute $P_1 \longleftarrow p_i P_1$ and set $n \longleftarrow np_i$.

3. Output n.

B.4 Computing Square Roots Modulo p

Let p be an odd prime, and let α be an integer in the interval $[1, p-1]$ such that α has a square root modulo p, i.e., there exists an integer β such that $\beta^2 \equiv \alpha \pmod{p}$.

(i) If $p \equiv 3 \pmod 4$, that is $p = 4u + 3$ for some positive integer u, then a square root of α modulo p can be obtained by computing $\beta = \alpha^{u+1} \bmod p$.

(ii) An algorithm for computing square roots of α in the case $p \equiv 1\ pmod4$ can be found in pages 48-49 of Koblitz's book [13].

Appendix C

Validation Suite (Test Vectors)

Appendix D

Known State of Attacks (Informative)

The security of the elliptic curve systems described in this standard hinges on the apparent difficulty of the *discrete logarithm problem* in the elliptic curve. Namely, given an elliptic curve E defined over a finite field $\mathbb{F}_q$, and given points P and $Q(= kP)$, find the integer k.

Unlike the case of the discrete logarithm problem in finite fields, there is no subexponential time algorithm known for the elliptic curve discrete logarithm problem (in the case that the curve is non-supersingular). The best algorithm known to date is the Pollard-ρ method [26] which takes about $\sqrt{\pi r/2}$ steps, where r is largest prime divisor of the order n of P.

Recently, van Oorschot and Wiener [25] discovered a technique for parallelizing the Pollard-ρ method so that if m processors are used, then the expected number of steps by each processor before a discrete logarithm is obtained is $(\sqrt{\pi r/2})/m$. Hence, to avoid this attack, it is necessary to select a curve and a point P of order divisible by a prime r so that $(\sqrt{\pi r/2})/m$ is sufficiently large for all feasible values of m. Preferably, the order n should itself be prime.

As a concrete example, van Oorschot and Wiener estimated that if r is about 10^{36}, then a machine with $m = 325,000$ processors can be built for about \$10 million which could compute a single logarithm in about 35 days.

The special classes of supersingular curves have been avoided in this standard since there is a method for efficiently reducing the discrete logarithm problem in these curves to the discrete logarithm problem in a finite field. However, it should be pointed out that there are some particular supersingular curves whose use may have some advantages over non-supersingular curves.

References (Informative)

Elliptic curve cryptosystems were first proposed in 1985 independently by Neil Koblitz [12] and Victor Miller [20]. Since then a lot of research has been done towards improving the efficiency of these systems and evaluating their security. For a summary of this work, consult [18]. A description of a hardware implementation of an elliptic curve cryptosystem can be found in [2].

Three good references on the theory of finite fields are the books by McEliece [17], Lidl and Niederreiter [16] and Jungnickel [10]. The article [1] discusses how to efficiently perform arithmetic operations in finite fields of characteristic 2. A hardware implementation of arithmetic in such fields which exploits the properties of so-called optimal normal bases is described in [3].

The ElGamal public-key encryption and signature schemes were introduced in [6]. The NIST Digital Signature Algorithm (DSA) is described in [23]. The Secure Hash Algorithm (SHA-1) is described in [4] and [22]. Abstract Syntax Notation One (ASN.1), Basic Encoding Rules (BER) and Distinguished Encoding Rules (DER) are described in [7], [8] and [9] respectively.

[1] G. Agnew, T. Beth, R. Mullin and S. Vanstone, "Arithmetic operations in $GF(2^m)$", *Journal of Cryptology*, **6** (1993), 3-13.

[2] G. Agnew, R. Mullin and S. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$", *IEEE Journal on Selected Areas in Communications*, **11** (1993), 804-813.

[3] G. Agnew, R. Mullin, I. Onyszchuk and S. Vanstone, "An implementation for a fast public-key cryptosystem", *Journal of Cryptology*, **3** (1991), 63-79.

[4] American National Standards Institute, "Public key cryptography using irreversible algorithms for the financial services industry: part 2: the secure hash algorithm (SHA)", ANSI X9.30-1993.

[5] E. Brickell, D. Gordon, K. McCurley and D. Wilson, "Fast exponentiation with precomputation", *Advances in Cryptology – EUROCRYPT '92*, Lecture Notes in Computer Science, **658** (1993), Springer-Verlag, 200-207.

[6] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, **31** (1985), 469-472.

[7] ISO/IEC 8824, "Information Technology – Open Systems Interconnection – Abstract syntax notation one (ASN.1)".

[8] ISO/IEC 8825-1, "Information Technology – Open Systems Interconnection – Specification of ASN.1 encoding rules – Part 1: Basic Encoding Rules (BER)".

[9] ISO/IEC 8825-3, "Information Technology – Open Systems Interconnection – Specification of ASN.1 encoding rules – Part 3: Distinguished canonical encoding rules".

[10] D. Jungnickel, *Finite Fields: Structure and Arithmetics*, B.I.-Wissenschaftsverlag, Mannheim, 1993.

[11] D. Knuth *The Art of Computer Programming*, volume 2, 2nd edition, 1981.

[12] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48** (1987), 203-209.

[13] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 2nd edition, 1994.

[14] G. Lay and H. Zimmer, "Constructing elliptic curves with given group order over large finite fields", *Algorithmic Number Theory: First International Symposium*, Lecture Notes in Computer Science, **877** (1994), Springer-Verlag, 250-263.

[15] R. Lercier and F. Morain, "Counting the number of points on elliptic curves over finite fields: strategies and performances", *Advances in Cryptology – EUROCRYPT '95*, Lecture Notes in Computer Science, **921** (1995), Springer-Verlag, 79-94.

[16] R. Lidl and H. Niederreiter, *Finite Fields*, Cambridge University Press, 1987.

[17] R.J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

[18] A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[19] A. Menezes and T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *IEEE Transactions on Information Theory*, **39** (1993), 1639-1646.

[20] V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology – CRYPTO '85*, Lecture Notes in Computer Science, **218** (1986), Springer-Verlag, 417-426.

[21] F. Morain, "Building cyclic elliptic curves modulo large primes", *Advances in Cryptology – EUROCRYPT '91*, Lecture Notes in Computer Science, **547** (1991), Springer-Verlag, 328-336.

[22] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS Publication 180-1, 1994 (revision of FIPS 180).

[23] National Institute of Standards and Technology, "Digital Signature Standard", FIPS Publication 186, 1993.

[24] K. Nyberg and R. Rueppel, "A new signature scheme based on the DSA giving message recovery", 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, November 3-5, 1993.

[25] P. van Oorschot and M. Wiener, "Parallel collision search with application to hash functions and discrete logarithms", in Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, November 2-4, 1994, 210-218.

[26] J. Pollard, "Monte Carlo methods for index computation mod p", *Mathematics of Computation*, **32** (1978), 918-924.

[27] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p", *Mathematics of Computation*, **44** (1985), 483-494.