ANR Project ECLIPSES

Elliptic Curve Leakage-Immune Processing for Secure Embedded Systems

D1.1

Review of cryptographic protocols based on elliptic curves

Contributor(s)

Anja Becker, Sorina Ionica, Jérôme Plût – UVSQ
Karine Villegas – Gemalto

Due date of deliverable: T0+x
Actual submission date: February 20, 2012
ECLIPSES partner in Charge: Gemalto, UVSQ

Confidentiality level: Restricted to ECLIPSES **Release:** 1.0

History

Version	Date	Author	Modification
1.00	27/08/10	Karine Villegas	Initial revision: Maths, ECDSA, crypto-processors
	27/08/10	Anja Becker	Initial ECDLP, ECDHP, ECDH key agreement, ECIES
	27/08/10	Sorina Ionica	Maths
	2010-09-03	Jérôme Plût	Maths
	13/09/10	Karine Villegas	Montgomery Ladder, ECDSA
	2010-09-20	Jérôme Plût	Protocols
	2010-09-30	Jérôme Plût	Final

ECLIPSES Partners

Start date of project: 2010, January 21 Duration: 3 years

Contents

1 Background material on elliptic curves over finite fields 3
1.1 Finite fields of prime characteristic . . . 3
1.2 Elliptic curves over prime fields . . . 6
1.3 Binary fields and elliptic curves . . . 10
1.4 Point multiplication techniques . . . 12
1.5 Elliptic curve discrete logarithm problem . . . 15
1.6 Elliptic curve Diffie-Hellman problem . . . 19
1.7 Domain parameters . . . 19
1.8 Group order computation . . . 21
1.9 Standardized curves . . . 22

2 Cryptographic protocols based on elliptic curves 23
2.1 Digital signature . . . 24
2.1.1 Schnorr's protocol . . . 24
2.1.2 ECDSA: Elliptic curve digital signature algorithm . . 24
2.1.3 ECGDSA: "German" ECDSA . . . 25
2.1.4 ECKCDSA: "Korean certificate" DSA . . . 25
2.2 Key agreement protocols . . . 26
2.2.1 ECDH: Elliptic curve Diffie-Hellman . . . 26
2.2.2 ECSTS . . . 27
2.2.3 ECMQV . . . 28
2.2.4 ECHMQV: Hashed MQV . . . 28
2.2.5 ECKMQV: Korean MQV . . . 29
2.2.6 FHMQV: Fully Hashed MQV . . . 29
2.3 Key /data encapsulation . . . 29
2.3.1 Elliptic curve integrated encryption system (ECIES-KEM) . . . 29
2.3.2 PSEC-KEM . . . 30

3 Crypto-processor functionalities state-of-the-art 30
3.1 Operations needed . . . 31
3.2 Co-processors and operations needed . . . 32
3.3 Characteristics and functionalities . . . 32
3.4 Operand constraints . . . 32
3.5 Modular multiplication . . . 33
3.6 Other functionalities . . . 33

List of algorithms

1 Background material on elliptic curves over finite fields 3
1.1.4 Barrett reduction algorithm 5
1.1.8 Montgomery step 6
1.4.1 Double-and-add algorithm 12
1.4.3 Sliding window algorithm 13
1.4.5 Yao's algorithm 13
1.4.6 Montgomery ladder 14
1.7.2 Validation of the domain parameters (E, n, G) 20

2 Cryptographic protocols based on elliptic curves 23
2.0.2 Key pair generation 24
2.1.1 Schnorr signature generation 24
2.1.2 Schnorr signature verification 24
2.1.3 ECDSA signature generation 25
2.1.4 ECDSA signature verification 25
2.1.5 ECGDSA signature generation 26
2.1.6 ECGDSA signature verification 26
2.1.7 ECKCDSA signature generation 26
2.1.8 ECKCDSA signature verification 26
2.2.1 ECDH key agreement 27
2.2.2 ECSTS key agreement 27
2.2.3 ECMQV key agreement 28
2.2.4 ECKMQV key agreement 29
2.3.1 ECIES-KEM encapsulation 30
2.3.2 ECIES-KEM decapsulation 30

3 Crypto-processor functionalities state-of-the-art 30
2.3.3 PSEC-KEM encapsulation 31
2.3.4 PSEC-KEM decapsulation 31

List of tables

1 Background material on elliptic curves over finite fields 3
1.2.11 Elliptic curve operation costs in various coordinates 10
1.7.1 Elliptic curve domain parameters 20

2 Cryptographic protocols based on elliptic curves 23
2.0.1 Levels of security 23

3 Crypto-processor functionalities state-of-the-art 30

List of figures

1 Background material on elliptic curves over finite fields 3
1.1.2 Algorithms used by gmp 4
1.2.3 Elliptic curve addition law, real numbers 8
1.2.4 Elliptic curve addition law, $\mathbb{F}_{37}$ 8
1.5.4 Record sizes (bits) for DLP in finite fields 17

2 Cryptographic protocols based on elliptic curves 23

3 Crypto-processor functionalities state-of-the-art 30
3.1.1 Required services and functions 31

Introduction

The purpose of this document is to review cryptographic protocols based on elliptic curves. The first chapter addresses the mathematical background, finite fields and elliptic curves, and describes some methods and algorithms for manipulating them. Furthermore, we present the computational problems on which security is based, as well as known attacks and security recommendations for the choice of parameters.
The second chapter presents protocols for digital signature, key exchange, and encryption, using the arithmetic of elliptic curves, and the theoretical security of those protocols is evaluated against the mathematical problems.

1 Background material on elliptic curves over finite fields

1.1 Finite fields of prime characteristic

Finite field arithmetic Let p be a prime number. We will write $k = \mathbb{F}_p$ for the finite field of integers modulo p. Most operations involving elliptic curves will be performed in this finite field, so we need the implementation of the arithmetic of $\mathbb{F}_p$ to be as fast as possible.
Let B be the size of the machine word; for example, $B = 2^{32}$ on a 32-bit CPU. Since p will be greater than B, we will generally represent elements of $\mathbb{F}_p$ as polynomials in B:

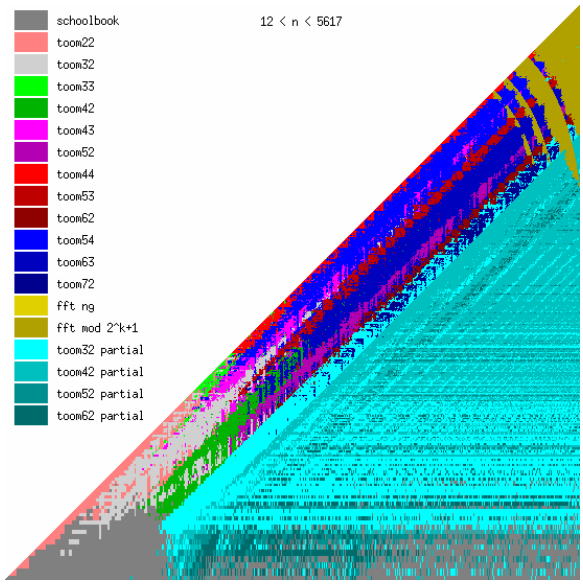
$$x = \sum_{i=0}^{n-1} B^i x_i, \tag{1.1.1}$$

where $n = \lfloor \log_B p \rfloor$. Such numbers are always non-negative, and all modular arithmetic can therefore be assumed to be unsigned.
The cost of most computations will be expressed in terms of arithmetic operations in $\mathbb{F}_p$. These operations are, in increasing order of cost:
– additions, negations and subtractions are comparatively cheap and will usually be neglected;
– $\mathbf{C}_a$ is the cost of a short multiplication (multiplication by an integer $a \leqslant B - 1$);
– **S** is the cost of a field squaring;
– **M** is the cost of a field multiplication;
– **I** is the cost of a field inversion.

Integer multiplication Most algorithms will avoid using field inversions through a suitable choice of coordinates; the determining factor will therefore be the speed of modular multiplications.
The most direct way to perform a modular multiplication $c = ab \pmod p$ is by first computing the integer product ab, then reducing modulo p. Both operations may be done using various algorithms.

Integer multiplication has been implemented and optimized for example in the gmp library: see figure 1.1.2. For balanced operands of sizes $\leqslant 512$ bits, schoolbook and Karatsuba (= toom22) are likely to be the two most useful methods.

Figure 1.1.2 Algorithms used by gmp for integer multiplication, depending on operand size (log scale, $32n$ bits, $12 < n < 5617$).



(source: `gmplib.org/devel`, CPU: Pentium 4)

Particular moduli Reduction modulo p can be made cheaper if the modulus p is a *pseudo-Mersenne prime* [Sol99], i.e. has a very sparse binary representation. Examples are given in the NIST recommendations [FIP00] or by the SEC group [SEC00], such as

$$p_{256} = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1, \qquad p_{521} = 2^{521} - 1. \quad (1.1.3)$$

It is possible to derive ad-hoc formulas for reduction modulo such a p that are much faster than full Euclidean division; see for instance [BHLM01], [NSA08].

Barrett and Quisquater reduction algorithms Barrett's modular reduction algorithm uses the fact that the number to be reduced, $x = ab$, is known to be not greater than p^2. Instead of performing a full Euclidean division, it uses an estimate $\widehat{q}$ of the quotient $q = x/p$ to compute the remainder. Barrett's algorithm uses the formula

$$q = \left\lfloor \frac{x}{p} \right\rfloor = \left\lfloor \frac{(x/B^{n-1})(B^{2n}/p)}{B^{n+1}} \right\rfloor \approx \left\lfloor \frac{\lfloor x/B^{n-1} \rfloor R}{B^{n+1}} \right\rfloor,$$

Algorithm 1.1.4 Barrett reduction algorithm

This algorithm computes $r = x \pmod{p}$.

1. **precompute** $R = \left\lfloor \frac{B^{2n}}{p} \right\rfloor$
2. $\widehat{q} = \left\lfloor \frac{\lfloor x/B^{n-1} \rfloor R}{B^{n+1}} \right\rfloor$
3. $r = \left(x \mod B^{n+1}\right) - \left(p\widehat{q} \mod B^{n+1}\right)$
4. **if** $r < 0$ **then** $r = r + B^{n+1}$ **end if**
5. **while** $r > p$ **do** $r = r - p$ **end while**
6. **return** r

where $R = \lfloor B^{2n}/p \rfloor$.
Since the estimate $\widehat{q} \in \{q-2, q-1, q\}$, the **while** loop in step 5 is executed at most twice, and with more than 90% probability is not executed at all; see [BGV94], [CFA06, §10.4.1] for a more detailed explanation.

Quisquater's algorithm [Qui90], [CFA06, §10.33] replaces division by p by division by a multiple p' of p. We define p' as being the smallest multiple of p such that $p' = cp > B^{n+1}$; we therefore can write, in base B:

$$p' = 1 \cdot B^{n+1} + 0 \cdot B^n + \ldots \tag{1.1.5}$$

Therefore, schoolbook division by p' becomes easy: if we write $x = x_{m-1}B^{m-1} + \cdots + x_0$, the leading B-ary digit of $\lfloor x/p' \rfloor$ is always x_{m-1}. We thus avoid all short divisions during the computation of the Euclidean division $x = p'q' + r'$. There only remains to compute $r = r' \pmod{p}$, where $r' < p'$.

Montgomery multiplication This algorithm [Mon85] allows modular multiplications to be performed efficiently for large moduli p, at the cost of some precomputations.
We represent a number $a \pmod{p}$ by the value $\overline{a} = R \cdot a \pmod{p}$ for some constant $R \in \mathbb{F}_p^\star$; R is chosen coprime to p, so that $a \longmapsto \overline{a}$ is bijective. We immediately note that this operation is linear, which means that $\overline{a+b} = \overline{a} + \overline{b}$, and therefore Montgomery representations can be added and subtracted in the straightforward way. As far as multiplications go, we can compute

$$\overline{ab} = \overline{a} \cdot \overline{b}/R \pmod{p} =: \mathsf{step}(\overline{a}, \overline{b}) \tag{1.1.6}$$

where the step operation is defined by $\mathsf{step}(x, y) = x \cdot y/R \pmod{p}$.
Now set in particular $R = B^n$. Then if we write $a = a_{n-1}B^{n-1} + \cdots + a_0$, we may compute ab/B^n as $(a_{n-1}b/B) + \cdots + (a_0 b/B^n)$ as a polynomial in $1/B$, using Horner's rule:

$$\frac{ab}{B^n} = \left(\ldots\left(\left(a_0 b \cdot \frac{1}{B} + a_1 b\right) \cdot \frac{1}{B} + a_2 b\right)\ldots\right) \cdot \frac{1}{B} \pmod{p}. \tag{1.1.7}$$

This requires performing a division by B at each step of the evaluation. Since B is a power of two, exact division by B in $\mathbb{Z}$ is cheap to perform via a right-shift. What we need is the divison $x/B \pmod p$; this can be done by replacing x by $x' = x + mp$ so that x' is a multiple of B and then right-shifting.
The Montgomery step algorithm can now be fully developed [CFA06, §11.3]:

Algorithm 1.1.8 Montgomery step

This algorithm computes $c = ab/B \pmod p$.

1. **precompute** p' such that $pp' + BB' = 1$.
2. $c \leftarrow 0$
3. **for** $i = 0$ **to** $n - 1$ **do**
4. $\quad m \leftarrow ((c_0 + b_0 a_i)p' \pmod B)$
5. $\quad c \leftarrow (c + a_i b + mp) \cdot \frac{1}{B}$ *(right-shift)*
6. **end for**
7. **if** $c \geqslant p$ **then** $c \leftarrow c - p$ **end if**
8. **return** c

Finally, it remains to see that the Montgomery reduction $a \longmapsto \bar{a}$ can be written as

$$\bar{a} = R \cdot a = \frac{R^2 \cdot a}{R} = \mathsf{step}(R^2, a) \pmod p, \tag{1.1.9}$$

where $R^2 \pmod p$ will of course be precomputed; and the *Montgomery lift* $\bar{a} \longmapsto a$ can be written as

$$a = \bar{a} \cdot R^{-1} = \frac{1 \cdot \bar{a}}{R} = \mathsf{step}(1, \bar{a}) \pmod p. \tag{1.1.10}$$

A Montgomery modular inversion algorithm is given in [CFA06, §11.12].

Performing a single modular multiplication bears the cost of these conversion operations as an overhead to the Montgomery step. Therefore, Montgomery representation is mostly useful when performing long chains of multiplications and squaring, like in modular exponentiation. Moreover, the Montgomery step algorithm has asymptotical complexity $O(n^2)$ in short multiplications, which is the same order as schoolbook multiplication, so it is not suitable for large values of n.

1.2 Elliptic curves over prime fields

An elliptic curve E over the field $k = \mathbb{F}_p$ is a (projective) curve that may be defined by a *(short) Weierstraßequation*:

$$y^2 = x^3 + ax + b, \tag{1.2.1}$$

where the parameters $a, b \in \mathbb{F}_p$ are such that the discriminant $\Delta = 4a^3 + 27b^2$ is not equal to zero in $\mathbb{F}_p$.

The *points of E* over k are the points $(x, y) \in \mathbb{F}_p^2$ satisfying the curve equation, to which we add a *point at infinity*, written $\mathcal{O}$, and which may be thought as the point at infinity in the vertical direction. We write $E(k)$ for the set of those points.
The cardinality of $E(k)$ satisfies the *Hasse-Weil bound*:

$$p - 2\sqrt{p} + 1 \leqslant \#E(q) \leqslant p + 2\sqrt{p} + 1. \tag{1.2.2}$$

In practice, this cardinality is of the same bit size as p. As a consequence of this, a point of an elliptic curves occupies a long-term storage size equal to $\lceil \log_2 p \rceil + 1$: namely, it is enough to store the x-coordinate and the sign of the y-coordinate; the y-coordinate may then be extracted as $y = \pm\sqrt{y^3 + ax + b}$. Point compression is even easier when $p \equiv -1 \pmod 4$, since in this case taking a square root in $\mathbb{F}_p$ is comparatively easy. However, for computational purposes, both coordinates should be used, and redundancy may even be introduced as described below.

Points of a curve may be represented in various coordinate systems, the choice of which will change the cost of the various curve operations we will later want to perform:
- **affine coordinates:** (x, y) represent the point (x, y) on the curve;
- **projective coordinates:** $(X : Y : Z)$ with $Z \neq 0$ represent the point $(x/z, y/z)$ on the curve, and $(0 : 1 : 0)$ represent $\mathcal{O}$;
- **Jacobian coordinates:** $(X : Y : Z)$ with $Z \neq 0$ represent the point $(X/Z^2, Y/Z^3)$, and $(0 : 1 : 0)$ represent $\mathcal{O}$.

Group law The points of E are endowed with a commutative group law by the *chord-and-tangent rule*, which is defined as follows:
- we define $\mathcal{O}$ as the zero element of the curve;
- for any three distinct points A, B, C, we have $A + B + C = 0$ iff A, B and C are aligned (chord rule);
- for any two distinct points A and C, we have $2A + C = 0$ iff the line AC is tangent to E at A (tangent rule).

The tangent rule may be seen as the limit case of the chord rule when $B = A$. The opposite of a point (x, y) is the point $(x, -y)$. Formulas for adding and duplicating points may be derived from these rules. In the following paragraphs, an *addition* means computing the point $P_3 = P_1 + P_2$ and a *doubling* means computing $P_3 = 2P_1$.

Affine coordinates These formulas all need to compute the slope λ of the line through P_1 and P_2.

Addition

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \quad x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1. \tag{1.2.5}$$

Figure 1.2.3 Elliptic curve addition law over the real numbers for the elliptic curve $y^2 = x(x+9)(x+25)$; $P_1 + P_2 = P_3$.

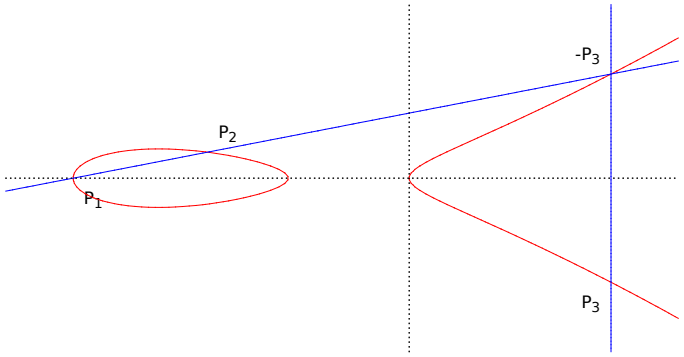
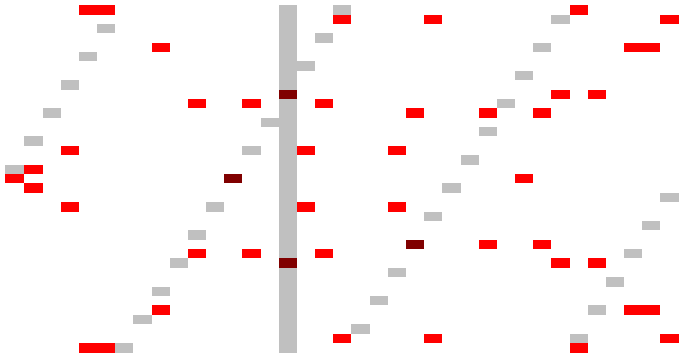


Figure 1.2.4 Same addition over $\mathbb{F}_{37}$



Doubling

$$\lambda = \frac{3x_1^2 + a}{2y_1}, \quad x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1. \tag{1.2.6}$$

All these formulas need a (costly) field inversion, which can be avoided by using projective coordinates. It should also be noted that the addition formula yields a division by zero in the following cases:
– when $P_1 = P_2$: a doubling should be used instead of an addition;
– when $P_1 = -P_2$: the sum is the zero point $\mathcal{O}$.

Projective coordinates

Addition

$$A = Y_2Z_1 - Y_1Z_2, \quad B = X_2Z_1 - X_1Z_2, \quad C = A^2Z_1Z_2 - B^3 - 2B^2Z_1Z_2,$$
$$X_3 = BC, \quad Y_3 = A(B^2X_1Z_2 - C) - B^3Y_1Z_2, \quad Z_3 = B^3Z_1Z_2. \tag{1.2.7}$$

It should be noted that, if $Z_2 = 1$, the cost drops by one multiplication. This is called a *mixed multiplication* and will enable to save some time when performing a point multiplication.
Moreover, these formulas yield $(0 : 0 : 0)$ when used for doublings. The following doubling formulas should be used instead.

Doubling

$$A = aZ_1^2 + 3X_1^2, \quad B = Y_1Z_1, \quad C = X_1Y_1B, \quad D = A^2 - 8C,$$
$$X_3 = 2BD, \quad Y_3 = A(4C - D) - 8B^2Y_1^2, \quad Z_3 = 8B^3. \tag{1.2.8}$$

Jacobian coordinates

Addition

$$A = X_1Z_2^2, \quad B = X_2Z_1^2, \quad C = X_2Z_1^3, \quad D = Y_2Z_1^3, \quad E = B - A, \quad F = D - C,$$
$$X_3 = -E^3 - 2AE^2 + F^2, \quad Y_3 = -CE^3 + F(AE^2 - X_3), \quad Z_3 = Z_1Z_2E. \tag{1.2.9}$$

When a mixed addition is performed, five multiplications are saved and the final cost is $7\mathbf{M} + 4\mathbf{S}$.
These formulas yield $(0 : 0 : 0)$ when used to compute a doubling. The following doubling formulas should be used instead.

Doubling

$$A = 4X_1Y_1^2, \quad B = 3X_1^2 + aZ_1^4,$$
$$X_3 = -2A + B^2, \quad Y_3 = -8Y_1^4 + B(A - Y_3), \quad Z_3 = 2Y_1Z_1. \tag{1.2.10}$$

If $a = -3$ then we can compute B as $B = 3(X_1 - Z_1^2)(X_1 + Z_1^2)$, which is more efficient. Moreover, almost all elliptic curves are isogenous to such a curve [BJ03], and thus curves with $a = -3$ are sufficiently general and not cryptographically weak. For this reason, NIST-recommended elliptic curves all have $a = -3$.

Edwards curves An Edwards curve is, in its most general form, a quartic curve with equation $x^2 + ay^2 = (xy)^2 + b$ ("inverted twisted Edwards coordinates" according to [BBJ+08], [BL10]). To save time in computations, the product xy is kept as a coordinate z, and therefore Edwards coordinates $(X : Y : Z : T)$ with $XY = ZT$ represent the point $(X/T, Y/T)$.
An elliptic curve may be transformed to an Edwards curve if, and only if, the group $E(k)$ contains $\mathbb{Z}/4\mathbb{Z}$ as a subgroup. This is a condition that is discouraged by cryptographic problems (see 1.5). Among the standard curves, none of the NIST curves on prime fields, and only two of the SEC2 curves (`secp112r2` and `secp128r2`) satisfy this condition.

Remarks on side-channel The affine, projective and Jacobian coordinate systems share the property that it is impossible to perform point doublings using the same formulas as point addition. For example, when performing point multiplication $m \cdot P$ using the double-and-add algorithm, simple power analysis leads to direct observation of the bits of the number m, which may be a secret key.
Other coordinate systems, such as Edwards curves or Jacobi curves [LS01] have a *unified* addition law, which means that the same formulae may be used for adding and doubling. This adds an extra layer of protection against side-channel attacks.

Tripling-oriented Doche-Icart-Kohel presented a curve shape that makes comparatively fast triplings. Its main use is in double-base point multiplication.

Coordinates	Condition	Addition	Doubling
Affine		$\mathbf{I} + 2\mathbf{M} + \mathbf{S}$	$\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$
Projective		$12\mathbf{M} + 2\mathbf{S}$	$7\mathbf{M} + 5\mathbf{S}$
Jacobian		$12\mathbf{M} + 4\mathbf{S}$	$4\mathbf{M} + 6\mathbf{S}$
Jacobian	$a = -3$	$12\mathbf{M} + 4\mathbf{S}$	$4\mathbf{M} + 4\mathbf{S}$
Edwards	$\mathbb{Z}/4\mathbb{Z}$	$9\mathbf{M} + 1\mathbf{S} + \mathbf{C}_a + \mathbf{C}_b$	$3\mathbf{M} + 4\mathbf{S} + \mathbf{C}_a + \mathbf{C}_b$

Table 1.2.11: Elliptic curve operation costs in various coordinates

Summary A more comprehensive table of formulas for point addition etc. can be found online at [BL10].

1.3 Binary fields and elliptic curves

Binary fields A *binary field* is a finite field whose cardinality is a power of two. All fields with the same cardinality are isomorphic, and the field $k =$

$\mathbb{F}_{2^e}$ may be represented as the quotient field of the polynomial ring $\mathbb{F}_2[X]$ by an irreducible polynomial $\pi(X)$ of degree e. Thus, elements of k may be represented as

$$a = \sum_{i=0}^{e-1} a_i X^i \pmod{\pi(X)}. \tag{1.3.1}$$

Field addition is then simply a bitwise addition (XOR), and field multiplication is a polynomial multiplication followed by a polynomial Euclidean division. The Frobenius endomorphism φ of k is defined by $\varphi(a) = a^2$; in particular, we have $(a+b)^2 = a^2 + b^2$. Therefore, squaring is significantly simpler to implement than multiplication.
It should be noted that, because of Weil descent attacks (see 1.5), the extension degree e should be prime.
As the choice of the polynomial π (as long as it is irreducible) does not impact on the field $\mathbb{F}_{2^e}$, it is preferable to choose a sparse polynomial as this makes Euclidean division faster. For every degree $e \leqslant 1000$, table [IEE00, §A.8] provides an irreducible polynomial of degree three or five over $\mathbb{F}_2$.

Normal bases A *normal basis* for the field k is a basis of k as a vector space over $\mathbb{F}_2$ of the form $\{\theta, \varphi(\theta), \ldots, \varphi^{e-1}(\theta)\}$. Since $\varphi(x) = x^2$, such a basis is ideally suited for squaring, which can then be implemented as a circular shift. However, multiplication is more cumbersome and depends on the matrix M of the multiplication by θ; multiplication by conjugates of θ can then be performed by shifting the lines and columns of M accordingly.
A *optimal normal basis* is a normal basis where the matrix M has the minimal number $2e-1$ of non-zero entries [GL92]. Such a basis does not exist for each possible size e; there are only 18 possible prime values $e \in [\![50, 500]\!]$ [CFA06, §11.2.1.c].
The NIST standard [FIP00] defines five binary fields $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$, $\mathbb{F}_{2^{409}}$ and $\mathbb{F}_{2^{571}}$ by giving for each of them a generating polynomial and a normal basis.

Elliptic curves An ordinary elliptic curve over k may be represented by the equation

$$E : y^2 + xy = x^3 + ax^2 + b. \tag{1.3.2}$$

The opposite of the point (x, y) is the point $(x, x+y)$.

Affine coordinates These formulas all need to compute the slope λ of the line through P_1 and P_2.

Addition

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1}, \quad x_3 = \lambda^2 + x_1 + x_2 + a, \quad y_3 = \lambda(x_1 + x_3) + x_3 + y_1. \tag{1.3.3}$$

Doubling

$$\lambda = x_1 + \frac{x_1}{y_1}, \quad x_3 = \lambda^2 + \lambda + a, \quad y_3 = \lambda(x_1 + x_3) + x_3 + y_1. \tag{1.3.4}$$

Projective coordinates See [CFA06].

1.4 Point multiplication techniques

Most cryptographic protocols based on elliptic curves make use of *scalar multiplication*, which is the multiplication of a point of the curve by an integer by means of successive additions and doublings.
In the following algorithms, we will use **A** and **D** for the respective costs of a point addition and a point doubling on E. Let P be a point of E and m be an integer of n bits.

Fast exponentiation algorithm This is the basic "double and add" algorithm (i.e., the additively written square-and-multiply algorithm).

Algorithm 1.4.1 Double-and-add algorithm

1. $R \leftarrow \mathcal{O}$;
2. **while** $m > 0$ **do**
3. **if** m is odd **then**
4. $R \leftarrow R + P$
5. **end if**
6. $P \leftarrow 2P$
7. $m \leftarrow m/2$
8. **end while**

The average cost for this method is of $\frac{1}{2}n\mathbf{A} + n\mathbf{D}$. It should be noted than when working in projective or Jacobian coordinates and an affine point P, we may replace all point additions in step 4 by mixed additions; in some cases, it may be actually faster to first convert a projective point P to affine coordinates (at the cost of one inversion and two multiplications) and then to use these to compute mP.

NAF and Windowed representations Some improvement can be achieved by using a *sliding window representation* for the integer m. Let $w \geqslant 1$ be an integer; then each integer m has a unique sliding window representation of size w

$$m = \sum_{i=0}^{n} m_i 2^i, \tag{1.4.2}$$

such that $|m_i| < 2^{w-1}$, each m_i is zero or odd, and among each window of w successive m_i, at most one is not zero.

Let m be given in this form. Then a fast scalar multiplication mP may be computed at the cost of precomputing odd multiples of P from $3P$ to $(2^{w-1}-1)P$:

Algorithm 1.4.3 Sliding window algorithm

1. $R \leftarrow \mathcal{O}$
2. **for** $i = 0 \ldots n$ **do**
3. **if** $t_i \neq 0$ **then**
4. $R \leftarrow R + m_i P$
5. **end if**
6. $R \leftarrow 2R$
7. **end for**

This uses on average $\frac{1}{w+1}(n+1)\mathbf{A} + (n+1)\mathbf{D}$, at the cost of some precomputation which depends only on P. This technique is therefore particularly useful when the point P is fixed.

Yao's algorithm Yao's algorithm is a transposed version of the sliding window algorithm. Starting from the representation 1.4.2, we group all those indices i which have the same digit $m_i = r$ and write $m^{(r)}$ for their sum, so that

$$m = \sum_{i=0}^{n} m_i 2^i = \sum_{r=0}^{2^w-1} r m^{(r)}. \tag{1.4.4}$$

We can then compute all $m^{(r)} \cdot P$ by successive doublings, and then $m \cdot P$ by the formula $m \cdot P = \sum r m^{(r)} \cdot P$.

Algorithm 1.4.5 Yao's algorithm

1. $m^{(r)} \leftarrow 0, P_r \leftarrow \mathcal{O}$ for $r = 0, \ldots, 2^w - 1$; $Q \leftarrow 0$
2. **for** $i = 0, \ldots, n-1$ **do**
3. $P_{m_i} \leftarrow P_{m_i} + Q$
4. $Q \leftarrow 2Q$
5. **end for**
6. $R \leftarrow \mathcal{O}$, $Q \leftarrow \mathcal{O}$
7. **for** $j = 2^w - 1, \ldots, 1$ **do**
8. $Q \leftarrow Q + P_{m_i}$
9. $R \leftarrow R + Q$
10. **end for**
11. **return** R

Although the complexity is the same as that of the sliding window method, this algorithm requires more storage space.

Montgomery ladder The usual trick for removing the conditional branching in the double-and-add algorithm that could be unsecure regarding

physical attacks consists in performing a dummy point addition when multiplier bit m_j is zero. As a result, each iteration appears as a point doubling followed by a point addition A popular point multiplication algorithm was developed by Montgomery as a means for speeding up the ECM factoring method on a special form of elliptic curves.

Algorithm 1.4.6 Montgomery ladder

1. $R_0 \leftarrow P$;
2. $R_1 \leftarrow 2P$;
3. **for** $i = n - 2 \ldots 0$ **do**
4. $\quad b \leftarrow 1 - m_i$
5. $\quad R_b \leftarrow R_0 + R_1$
6. $\quad R_{m_i} \leftarrow 2R_{m_i}$
7. **end for**
8. **return** R_0

This algorithm behaves regularly but seems, at first glance, as costly as the double-and-add-always algorithm. However, it does not need to handle the y-coordinates: the sum of two points whose difference is a known point can be computed without the y-coordinates [Mon87]. Note, that the difference $R_1 - R_0$ remains invariant throughout the algorithm (and hence is equal to P). Further, the y-coordinate of a point R_0 can be recovered from a point P, the x-coordinate of R_0, and the x-coordinate $R_0 + P$. The y-coordinate of $R = [m]P$ can thus also be recovered when only x-coordinates are used in the Montgomery point multiplication algorithm.

Using addition chains Some methods for computing multiples in the group of points of E arise from the attempt to minimise the number of group operations (additions and doublings) necessary to compute $m \cdot P$. An *addition chain* for an integer m is a sequence of integers $(a_1 = 1, \ldots, a_r = m)$ such that each a_i is the sum of two previously occurring $a_{i'}$ and $a_{i''}$; minimizing such a chain naturally leads to faster computations for multiples.
For example, double-and-add and NAF methods may be thought of as particular addition chains.
While this is true for all groups, some remarks apply to the particular case of elliptic curves.

- Negating a point is free, and therefore a subtraction costs the same as an addition; consequently, we may allow subtractions in our chains.
- In most useful coordinates systems, doubling a point is significantly cheaper than adding two separate points; therefore the appropriate measure of complexity of a chain must weigh doublings accordingly.
- While we may write down formulas for point tripling for all coordinate systems, some shapes are especially suited to tripling and have a comparatively fast tripling operation. Therefore, we may also allow tripling in operation chains, provided that we still use a good weighting.

Also note that, particularly in the case of a fixed base point, some precomputation may be useful to speed these methods.
The interest of this is that the length of the shortest addition chain for a given m is sub-linear. Although finding this shortest chain is a hard problem, finding one short chain is generally enough.
An example of such a chain is given by a decomposition of the integer m in double-base $\{a,b\}$:

$$m = \sum_{i=0}^{r} \pm m_i a^{\alpha_i} b^{\beta_i}, \tag{1.4.7}$$

where α_i and β_i are integers. In practical uses, we will always take $\{a,b\} = \{2,3\}$. This representation is redundant: in general, m has several such representations.
A suitable $\{2,3\}$-representation of m may be found by a greedy algorithm [DIM08], starting by the biggest value $2^\alpha 3^\beta \leqslant m$. This yields a sub-linear algorithm for addition in $O(\frac{\log m}{\log\log m})$ curve operations. However, the first part of this algorithm relies on finding the best approximation by default of $\log m$ by $\alpha\log 2 + \beta\log 3$, and thus on floating-point arithmetic including computation of logarithms.
Further study [BBLP07] has found this method to be somewhat faster than sliding-window methods for tripling-oriented curves, but not significantly better for doubling-oriented Edwards or Jacobi quartic curves in cryptographic sizes.

1.5 Elliptic curve discrete logarithm problem

Let Γ be a cyclic group of order N and g be a generator of Γ. For any integer m, computing $a = g^m$ can be done in $O(\log N)$ operations in Γ. The inverse problem, given g and a, finding m such that $a = g^m$, is known as the *discrete logarithm problem* in the group Γ. The integer m is known as the *discrete logarithm* of a and may be written $m = \log_g a$.
Two particular instances of this problem are used in cryptography:
– when $\Gamma = \mathbb{F}_p^\times$ is the multiplicative group of a finite field; Diffie-Hellman, DSA, and ElGamal are based on this scheme.
– when $\Gamma = E(k)$ is the group of points of an elliptic curve E, which is the only case in which we are interested.
The abstract DLP requires a worst-case average $O(\sqrt{N})$ to solve for a generic group Γ (Shoup, 1997), although ad-hoc faster techniques may exist for particular groups Γ. The naïve algorithm of brute-force attack has an exponential complexity of $O(N)$ operations in Γ.
Moreover, if N factors in prime powers $p_1^{e_1} \dots p_r^{e_r}$, then the Pohlig-Hellman attack reduces the DLP in Γ to that in groups of order $p_1, \dots, p_r$; therefore, we will want to work in groups of prime, or almost prime (*i.e.* of the form $h \cdot p$ with h small and p prime), order. The integer h is called the *cofactor* and needs to be extremely small.

For example, elliptic curves representable as Edwards curves have a subgroup isomorphic to $\mathbb{Z}/4\mathbb{Z}$, and thus a cofactor $h \geqslant 4$. Therefore, instead of working in $E(k)$, we will work in a subgroup of index 4 and thus lose two bits of group size, which amounts to one bit of security as shown below.

Baby-step, giant step This is a generic time vs. space tradeoff technique for attacking the DLP. Let $r < N$, for example $r = \left\lceil \sqrt{N} \right\rceil$. Then any number $m < N$ may be written as $m = m_1 r + m_0$ with $m_0 < r$ and $m_1 < N/r$. If, for such an m, we have $a = g^m$, then

$$a(g^{-r})^{m_1} = g^{m_0}. \tag{1.5.1}$$

We may therefore store values of g^i for $i = 0, \ldots, r-1$ in a hash table and search the list of $a(g^{-r})^j$ for $j = 0, \ldots, N/r$ for such a value.
When choosing $r = \sqrt{N}$, the complexity will be $O(\sqrt{N})$ with storage space requirement $O(\sqrt{N})$.

Pollard's rho method This technique also works with any group Γ. If we can find four numbers u, v, u', v' such that $g^u a^v = g^{u'} a^{v'}$, then we can compute

$$\log_g a = \frac{u - u'}{v - v'} \pmod{N}. \tag{1.5.2}$$

We equip the group Γ with a pseudorandom function $f : \Gamma \longrightarrow \Gamma$ such that the iterated sequence $x_i = f^i(x_0)$ is easy to write as $x_i = g^{u_i} a^{v_i}$. This sequence will eventually loop and, since f is pseudorandom, we expect the final cycle of the sequence $(x_i)_{i \leqslant 0}$ to be of length $O(\sqrt{N})$. When we find a collision in the sequence, we may deduce m using the formula 1.5.2.
To find the cycle, it is not necessary to store all computed values x_i. We may instead compute simultaneously both sequences (x_i) and (x_{2i}) and stop when we find i such that $x_i = x_{2i}$.
This method requires polynomial storage space and is easy to implement in parallel by starting several sequences $x_i^{(j)}$ with various starting points $x_0^{(j)}$; over r processors, an average speedup of $\sqrt{s}$ is expected.

Index calculus Index calculus [EG02] may be used to attack in subexponential time the DLP in groups Γ endowed with the following data:
– a covering monoid M such that Γ is a quotient group of M;
– a set of *primes*, $\mathfrak{P}$, in M;
– a size function $M \longrightarrow [0, +\infty[$.
For any real number t, we define an element x in Γ as being *t-smooth* if it has a representative in M that may be written as the product of primes of size $\leqslant t$. We assume that:
– t-smoothness can be tested for in polynomial time;
– the prime decomposition of a t-smooth element can be computed in polynomial time.

We then choose a $t > 0$ and construct the set $\mathfrak{P}_t = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_n\}$ of all primes of size $\leqslant t$. To compute $\log_g a$ in Γ, we then select randomly numbers α_j and β_j such that $g^{\alpha_j} a^{\beta_j}$ is t-smooth, and compute its prime decomposition:

$$g^{\alpha_j} a^{\beta_j} = \mathfrak{p}_1^{a_{1j}} \ldots \mathfrak{p}_n^{a_{nj}}.$$

Taking logarithms in base g, we thus obtain a linear relation between the unity, $\log_g a$ and the $\log_g \mathfrak{p}_j$. We then iterate this process until we find enough relations to compute $\log_g a$.
Specific applications for index calculus include the following.

1. *Prime fields*. We take $M = \mathbb{N}$, $\mathfrak{P}$ is the set of prime numbers, and the size function is $\log(x)$.

   Sieving can be improved by the use of the number field sieve algorithm [Gor93], which is currently the fastest known algorithm for attacking the DLP in finite fields.

   Let K be a number field in which p splits, *i.e.* there exists a surjective ring morphism $\theta : \mathcal{O}_K \longrightarrow \mathbb{F}_p$. Then, in view of the index calculus setting, we may choose as our monoid M the group of fractional ideals of K. The size function maps an ideal $\mathfrak{a}$ to its norm in $\mathbb{Z}$.

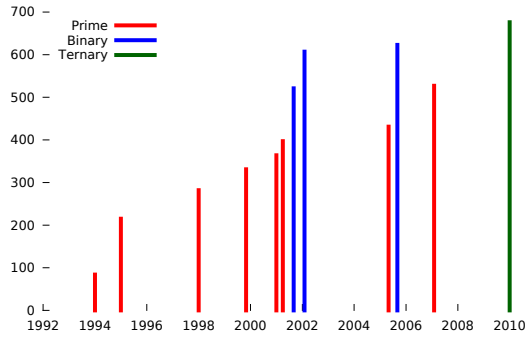
   Let $x \in \mathcal{O}_K$ be an antecedent of a for θ; then we may sieve algebraic numbers of the form $u + vx$ to find linear relations.

   The asymptotic complexity of this method [Cop93] is $L_p(\frac{1}{3}, c)$ for some constant $c \leqslant 1.9018$, where

   $$L_x(\alpha, c) = \exp\left(c \cdot (\log x)^{\alpha} \cdot (\log\log x)^{1-\alpha}\right). \tag{1.5.3}$$

   Various records for DLP solving have been obtained through the NFS and its sister algorithm, the function field sieve, for fields of small characteristic.

Figure 1.5.4 Record sizes (bits) for DLP in finite fields



2. *Hyperelliptic curves*. We take $M =$ the group of zero-degree divisors on an hyperelliptic curve C of order g, $\mathfrak{P}$ the set of all $P - [\mathcal{O}]$ where P is prime, and the size function is given by the degree of the

x-part polynomial. It is thus possible to solve DLP in $\Gamma = \mathrm{Jac}(C)(\mathbb{F}_q)$ in $O(q^{2+\varepsilon})$; since Γ is of order $\sim q^g$, this is an improvement over the square-root methods whenever g is large enough. A variant of this attack [Thé03] is better than the square-root method for $g \geqslant 3$. Thus, the only hyperelliptic curves useful for cryptographic purposes are those of genus $g = 2$.

In particular, this method is not directly available for attacking ECDLP: namely, all elements of the group of divisors are representable by points of E and prime.

Reduction techniques for the discrete logarithm in elliptic curves

Anomalous elliptic curves The curve E is said to be *anomalous* if $\left|E(\mathbb{F}_p)\right| = p$. In this case, (Smart; Akari-Satoh; Semaev) showed in 1997 that there exists a group isomorphism $E(\mathbb{F}_p) \longrightarrow \mathbb{F}_p$ that is computable in polynomial time. Since the DLP in $\mathbb{F}_p$ amounts to a single field division, this means that the DLP in anomalous curves is solvable in polynomial time.

Weil descent and GHS reduction Let E be an elliptic curve over $k = \mathbb{F}_{p^n}$ where $n \leqslant 2$ is a composite number. Then its Weil scalar reduction to a subfield of k is a higher-dimensional abelian variety A; under some conditions [GHS02, Die03] the map $E \longrightarrow A$ does not collapse the useful prime subgroup of $E(k)$ in A, and A is an hyperelliptic curve in which index calculus techniques are faster than Pollard rho method in $E(k)$, and the

Weil pairing and MOV reduction Let E be an elliptic curve over k and G be a point of E generating a group of order ℓ. We may assume that ℓ is a prime number distinct from p.
We write $E[\ell]$ for the subgroup of points of ℓ-torsion of E over the algebraic closure of k; one may define the *Weil pairing*

$$e_\ell : E[\ell] \times E[\ell] \longrightarrow \mu_\ell(\overline{\mathbb{F}_p}), \tag{1.5.5}$$

where $\mu_l(\overline{\mathbb{F}_p})$ is the group of ℓ-roots of unity. Since this pairing is non-degenerate, this allows us to transport the DLP in $E[\ell]$ to the DLP in $k' = k(\mu_\ell)$, where k' is the ℓ-th cyclotomic field extension of k.
More precisely, if $k = \mathbb{F}_q$, we will have $k' = F_{q^e}$, where e is the smallest integer such that ℓ divides $q^e - 1$; or, in other words, $q^e \equiv 1 \pmod{\ell}$.

In order to protect from this attack, we need to ensure that discrete logarithms in k' will be at least as hard as discrete logarithms in E. We therefore compute B such that DLP in $\mathbb{F}_{q^B}$ is as hard as DLP in E, and check that $q^e \not\equiv 1 \pmod{\ell}$ for $e = 1, \ldots, B$. The IEEE standard P1363 provides a table of suitable values of B [IEE00, §A.12.1]. We see that $B = 10$ is enough for 192-bit curves, $B = 20$ for 384-bit curves, and $B = 30$ for 512-bit curves.

Summary An elliptic curve E, with order r, over $\mathbb{F}_p$ is suitable for ECDLP-based cryptography if it meets the following criteria:

1. r is almost prime, *i.e.* there exist ℓ prime and h small ($h \leq 4$) such that $r = h\ell$;
2. E is not anomalous, *i.e* $r \neq p$;
3. E satisfies the MOV condition: $p^e \not\equiv 1 \pmod{\ell}$ for $e = 1, \ldots, B$.

Provided those criteria are satisfied, the best method known for attacking this particular instance of the discrete logarithm problem is Pollard's rho method. The record size of 112 bits for a prime field was obtained through this method in July 2009 [BKM09].
For a more complete historical perspective on ECDLP we refer to [Men08].

1.6 Elliptic curve Diffie-Hellman problem

The *computational Diffie-Hellman problem* (ECCDHP) on an elliptic curve E is: given three points P, aP and bP, compute abP.
This problem is obviously weaker than the ECDLP on the same elliptic curve, since taking logarithms directly leads to a solution of the problem. However, in most cases the two problems are actually equivalent [MW00], and thus the ECCDHP is widely accepted as about as strong as the ECDLP.
There exist some variants of this problem:
– the *decisional Diffie-Hellman problem* (ECDDH) is, given P, aP, bP and Q, deciding whether or not $Q = abP$;
– the *gap Diffie-Hellman problem* is to solve the computational Diffie-Hellman problem, assuming access to an oracle who can solve the decisional Diffie-Hellman problem.

A list of related problems is maintained on `http://www.ecrypt.eu.org/wiki/index.php/Discrete_Logarithms`. The best currently known method for attacking most of them (including the three variants above) is to solve the DLP.

1.7 Domain parameters

The domain parameters for an elliptic curve scheme describe an elliptic curve E defined over a finite field $\mathbb{F}_p$, a base point $G \in E(\mathbb{F}_p)$, and its order n. They may be distributed in a public key certificate and are:

Generating a suitable elliptic curve There are several methods for generating an appropriate elliptic curve, in decreasing order of generality of the resulting curve [KMV00].

1. *Random curves*. We pick randomly $a, b \in \mathbb{F}_p$ such that $\Delta = 4a^3 + 27b^2 \neq 0$; we then compute the order r of the curve $E : y^2 = x^3 + ax + b$ and check that it satisfies all our criteria.

p	Number of elements in $\mathbb{F}_p$
S	Optional random number seed
a, b	Parameters for the curve $E : y^2 = x^3 + ax + b$
G	Base point on the curve E
n	Order of G in the group of points of E
h	Cofactor, *i.e.* $\left\|E(\mathbb{F}_p)\right\| / n$

Table 1.7.1: Elliptic curve domain parameters

This method produces the most general curves, with the added bonus that it is easy to check that the curve was randomly produced (by outputting the seed).

2. *Complex multiplication*. This method is detailed in 1.8. As we compute the group order before doing any computations on the curve itself, it is much faster and easier to implement than any method based on point counting. However, the curves obtained are not fully generic: namely, their endomorphism ring has a small class group, although no known attack is based on this fact.
3. *Koblitz curves*. We start with a known curve E over $\mathbb{F}_p$ with p small, and then use the curve E over some extension $\mathbb{F}_{p^e}$ of $\mathbb{F}_p$. While the group order of $E(\mathbb{F}_{p^e})$ is easy to compute, curves produced with this method are extremely particular.
4. *Use a published curve*. Since in most protocols the curve itself is not a part of the secret key, we can finally use a published curve for which no attack is known. These curves obviously meet all the previous criteria, but might be vulnerable to a future attack. See § 1.9.

The complex multiplication and random curves method require extensive computations, including a huge storage space and floating-point multi-precision computations. As such, they are far too cumbersome to consider implementing them on a small chip.
They can be replaced by a procedure for curve validation, where the curve order (provided to the chip in the domain parameters) is validated.

Algorithm 1.7.2 Validation of the domain parameters (E, n, G)

1. check that p is a prime number of the required size.
2. check that $a, b, x_G, y_G \in [\![0, p-1]\!]$.
3. check that $\Delta = 4a^3 + 27b^2 \neq 0 \pmod{p}$.
4. check that $G \in E$.
5. check that $n \neq p$
6. check that $p^e - 1 \not\equiv 0 \pmod{n}$ for $e = 1, \ldots, B$.
7. check that $G \neq \mathcal{O}$ and $n \cdot G = \mathcal{O}$ in E.
8. check that n is prime.

Primality checking may be performed using probabilistic algorithms such as the Miller-Rabin test. If an integer n is composite, then the probability for a random $a \in [\![1, n-1]\!]$ that the Miller-Rabin test with base a returns "prime" is at most $\frac{1}{4}$ [DLP93]. Therefore, if at least $N/2$ bases a are used, then an attacker sending composite numbers will on average have to send $4^{N/2} = 2^N$ numbers n, thus providing N bits of security.
In case a random seed was provided to prove that the parameters were randomly generated, checking them should obviously be performed in addition to the previous steps. Using either random or standardized parameters is advisable; there exist some attacks reposing on substitution of the base point G by a malicious point or of the curve E itself by a curve E' with the same j-invariant [Vau04].

1.8 Group order computation

Whether an elliptic curve E is suitable for cryptographic purposes depends only on its group order r. We recall that this group order lies between the Hasse-Weil bounds (1.2.2). More precisely, we will write

$$r = \left|E(\mathbb{F}_p)\right| \;=\; p + 1 - t, \qquad |t| \leqslant 2\sqrt{p}. \tag{1.8.1}$$

The number t is the trace of the Frobenius endomorphism of E: more precisely, the Frobenius endomorphism $\varphi : (x, y) \longmapsto (x^p, y^p)$ satisfies $\varphi^2 - t\varphi + p = 0$.

Baby step, giant step This algorithm proceeds by picking random elements x_i from the curve and computing the order r_i of each x_i. Since this order divides r, this is also true of their least common multiple $\mathsf{lcm}(r_1, \ldots, r_n)$. When collecting enough r_i, eventually this number will lie inside the Hasse-Weil bounds and thus be equal to r.
The time complexity of this algorithm is $O(p^{\frac{3}{4}})$.

Schoof-Elkies-Atkin algorithm The basic idea of the family of ℓ-adic algorithms is that knowing $t_\ell = t \pmod{\ell}$ for sufficiently many prime numbers ℓ is enough to compute t. The number t_ℓ is given by the action of the Frobenius endomorphism on the ℓ-torsion points of E. More precisely, t_ℓ must satisfy:

$$\varphi^2(P) - t_\ell \cdot \varphi(P) + (p \pmod{\ell}) \cdot P \;=\; \mathcal{O} \tag{1.8.2}$$

for each point P of ℓ-torsion; in other words, the formula for computing 1.8.2 must be divisible by the polynomial for point multiplication by ℓ. Trying all possible values for t_ℓ gives an algorithm (Schoof, 1985) with time complexity $O(\log^8 p)$.
Further improvements due to Elkies and Atkin [BSSC05] give an algorithm with time complexity $O(\log^6 p)$ and storage requirement $O(\log^2 p)$. Although this algorithm is polynomial, it is quite unwieldy; in particular, it

requires the precomputation of the modular polynomials Φ_ℓ (or some other polynomials generating the same field extensions), the full size of which is of the order of 30 Mbytes for cryptographic curve sizes.

***p*-adic algorithms** These algorithms compute $|E(k)| \pmod{p^n}$ for n large enough that the curve size is uniquely determined. Let $q = p^e$ be the cardinality of the base field, where the prime number p is the characteristic; all p-adic algorithms have in common an exponential dependency in p and a polynomial dependency in e. Therefore, they are best suited for computing group sizes for binary elliptic curves, that is when $p = 2$. See [CFA06] for more details.

Complex multiplication The goal of this algorithm is to generate an elliptic curve whose group order is known in advance of expensive computations, thus limiting the "trial-and-error" part.
An elliptic curve E over $k = \mathbb{F}_p$ has a ring of endomorphisms isomorphic to an order $\mathcal{O}$ in a quadratic extension K of $\mathbb{Q}$; knowing $\mathcal{O}$ is enough to compute the order of E. Namely, the Frobenius endomorphism φ of E over $\mathbb{F}_p$ satisfies $\varphi^2 - t\varphi + p = 0$. Therefore, if φ' is the Galois conjugate of φ in $\mathcal{O}$, then we have $\varphi\varphi' = p$ and $\varphi + \varphi' = t/2$; which means that t is a root of the Diophantine equation

$$4p \;=\; t^2 + Ds^2, \tag{1.8.3}$$

where D is the reduced discriminant of K.
We start by choosing a discriminant D such that 1.8.3 has a suitable t as a solution. The j-invariant of E is then a generator of the Hilbert class field L of K; more precisely, it is a root of the class polynomial h_D of D, which is a polynomial with integer coefficients. Floating-point computations allow us to compute exactly H_D, and factorizing it over $\mathbb{F}_p$ gives a value $j(E) \in \mathbb{F}_p$; once $j(E)$ is known, admissible coefficients for E follow by a simple formula.
The limit of this algorithm lies in the factorization of $H_D \pmod{p}$. This polynomial is of degree equal to the order of the ideal class group of the field $Q(\sqrt{-D})$; therefore, we are limited to small class numbers h_D ($h_D \leqslant 10^7$). The curves produced by this method lie in a strict subset of all elliptic curves, and it is possible that special attacks on the discrete logarithm exploit this [JMV04], although none is known now. Therefore, when generating a random curve, it is advisable to check that $h_D > 10^7$, which is almost always true.

1.9 Standardized curves

Several sources for standardized curves exist.
The NIST [FIP00] published five prime curves defined over pseudo-Mersenne prime fields (of size 192, 224, 256, 384 and 521 bits) and ten binary curves, including five Koblitz curves and five pseudo-randomly gen-

erated (of size 163, 233, 283, 409 and 571 bits). Of these, the prime curves of size 256, 384 and 521 are mandated by IPsec.
The SEC group [SEC00] published fifteen prime curves of size 112, 128, 160, 192, 224, 256, 384 and 521 bits (including the NIST curves) over pseudo-Mersenne prime fields, as well as seventeen binary curves of size 113, 131, 163, 193, 233, 239, 283, 409 and 571 bits; both types include pseudo-random as well as Koblitz curves.
The Brainpool consortium [The05] published seven curves defined over prime fields of pseudo-random characteristic (of size 160, 192, 224, 256, 320, 384, 512 bits).

2 Cryptographic protocols based on elliptic curves

Throughout this part, a set of domain parameters (p, E, G, n, h) (see 1.7) is public and agreed upon. As the protocols are mostly backed by the ECDLP or ECDH cryptographic problems in the subgroup generated by the base point G, correct choice of the parameters (as in § 1.5) ensures that the best known attacks are the generic square-root attacks described above. Therefore, the expected security of the protocols will be $\ell/2$ bits, where $\ell = \lceil \log_2 n \rceil$ is the bit size of n.
Table 2 shows the different levels of security and the according bit size of the subgroup in which computations are processed. The range for p in ECLIPSES is $\{160, ..., 384\}$.

Security level (bits)	Subgroup size
80	160
112	224
128	256
192	384
256	512

Table 2.0.1: Levels of security and the according bit size of the subgroup generated by the base point

We recall the following data about storage size, bandwidth and computation times:
– a point of E requires a long-term storage size of ℓ, and about 3ℓ short-term storage size (depending on the curve representation);
– most integer computations will be done either modulo p (for point coordinates) or modulo n (for point multiples), which requires a storage size of ℓ.

All following protocols are based either on ECDH or ECDLP, except where noted. In this view, a *key pair* for a user A is a pair (d_A, D_A), where $d_A \in [\![1, n-1]\!]$ is the private key, and $D_A = d_A \cdot G$ is the public key derived from d_A.

Algorithm 2.0.2 Key pair generation

1. Select random $d_A \in [\![1, n-1]\!]$; this is A's private key.
2. Compute $D_A = d_A \cdot G$ and check that D_A is on the curve E.
3. D_A is A's public key.

We see that both d_A and D_A are of size ℓ. This algorithm requires only one point multiplication.

Auxiliary functions Some protocols may also require the following auxiliary functions:

Hash is a cryptographic hash function with output size of at least ℓ bits. According to the value of ℓ, SHA-256, SHA-384 or SHA-512 may be used.

(Enc, Dec) is a symmetric cypher; $\mathsf{Enc}_k(m)$ is the encryption of plaintext m with key k, and $\mathsf{Dec}_k(c)$ is the decryption of cyphertext c with key k.

KDF is a key derivation function that computes a key for a symmetric cypher from some seed (usually a point of the elliptic curve); $\mathsf{KDF}(P, \ell)$ is a key of size ℓ generated from seed P.

MAC is a message authentication code; $\mathsf{MAC}_k(m)$ is the authentication of message m with key k.

Sign is a signature function, as described below; $\mathsf{Sign}_A(m)$ is the signature of message m by user A.

2.1 Digital signature

2.1.1 Schnorr's protocol

Let m be the message to sign.

Algorithm 2.1.1 Schnorr signature generation

1. Select random $k \in [\![1, n-1]\!]$.
2. Compute $e = \mathsf{Hash}(m \parallel k \cdot G)$.
3. Compute $s = (k - d_A \cdot e) \pmod{n}$.
4. The signature is (e, s).

Algorithm 2.1.2 Schnorr signature verification

1. Compute $R = s \cdot G + e \cdot D_A$.
2. The signature is valid if $e = \mathsf{Hash}(m \parallel R)$.

2.1.2 ECDSA: Elliptic curve digital signature algorithm

The elliptic curve digital signature algorithm (ECDSA) is standardized in [IEE00, ANS99, FIP00, ISO00]. As for RSA, the sensitive elements as the

private key might be recovered in a standard straight forward implementation of ECDSA. A secure implementation will therefore differ from algorithms 2.1.3 and 2.1.4.

Signature generation To sign the message m, an entity A with key pair (d_A, D_A) does the following:

Algorithm 2.1.3 ECDSA signature generation

1. Select random $k \in [\![1, n-1]\!]$.
2. Compute $kG = (x_1, y_1)$.
3. Compute $r = x_1 \mod n$. If $r = 0$, then go to step 1.
4. Compute $k^{-1} \mod n$.
5. Compute $e = \ell$ most significant bits of integer $\mathsf{Hash}(m)$.
6. Compute $s = k^{-1}(e + d_A r) \mod n$. If $s = 0$ go to step 1.
7. The signature of the message m is the pair (r, s).

As both integers r and s are defined modulo n, the size of the signature is 2ℓ.
A modification of ECDSA, replacing $\mathsf{Hash}(m)$ by $\mathsf{Hash}(m \parallel r)$, is as secure as the discrete logarithm in E under the random oracle model for the hash function [MLS03].

Signature verification To verify signature (r, s) on m, the entity B obtains the copy of domain parameters $D = (p, E, G, n, h)$ and the associated public key D_A. B then does the following:

Algorithm 2.1.4 ECDSA signature verification

1. Compute $w = s^{-1} \mod n$.
2. Compute $e = \ell$ most significant bits of integer $\mathsf{Hash}(m)$.
3. Compute $u_1 = ew \mod n$ and $u_2 = rw \mod n$.
4. Compute $X = (x_1, y_1) = u_1 G + u_2 D_A$.
5. If $X = \mathcal{O}$ then reject the signature. Otherwise compute $v = x_1 \mod n$.
6. Accept signature if $v = r$.

2.1.3 ECGDSA: "German" ECDSA

This algorithm [ISO02] simplifies ECDSA by avoiding inversion of the ephemeral secret k.

2.1.4 ECKCDSA: "Korean certificate" DSA

This protocol [ISO02] is based on a certificate. In what follows, h_{cert} is the hash value of a certificate including the signer (A)'s identifier, domain parameters, and public key d_A.

Algorithm 2.1.5 ECGDSA signature generation

1. Select random $k \in [\![1, n-1]\!]$.
2. Compute $kG = (x_1, y_1)$.
3. Compute $r = x_1 \mod n$. If $r = 0$, then go to step 1.
4. Compute $e = \mathsf{Hash}(m)$.
5. Compute $s = d_A(kr - e) \mod n$. If $s = 0$, then go to step 1.
6. The signature of m is the pair (r, s).

Algorithm 2.1.6 ECGDSA signature verification

1. Compute $r' = r^{-1} \mod n$.
2. Compute $e = \mathsf{Hash}(m)$.
3. Compute $h_1 = r'e \mod n$ and $h_2 = r's \mod n$.
4. Compute $X = (x_1, y_1) = h_1 G + h_2 D_A$.
5. If $X = \mathcal{O}$ then reject the signature. Otherwise compute $v = x_1 \mod n$.
6. Accept signature if $v = r$.

Algorithm 2.1.7 ECKCDSA signature generation

1. Select random $k \in [\![1, n-1]\!]$.
2. Compute $kG = (x_1, y_1)$.
3. Compute $r = \mathsf{Hash}(x_1) \mod n$. If $r = 0$, then go to step 1.
4. Compute $e = \mathsf{Hash}(h_{\text{cert}} \parallel m)$.
5. Compute the integer $w = r \oplus e \pmod n$.
6. Compute $s = d_A(k - w) \mod n$. If $s = 0$, then go to step 1.
7. The signature of m is the pair (r, s).

Algorithm 2.1.8 ECKCDSA signature verification

1. Compute $e = \mathsf{Hash}(h_{\text{cert}} \parallel m)$.
2. Compute the integer $w = r \oplus e \pmod n$.
3. Compute $X = (x_1, y_1) = wG + sD_A$.
4. If $X = \mathcal{O}$ then reject the signature. Otherwise compute $v = \mathsf{Hash}(x_1) \mod n$.
5. Accept signature if $v = r$.

2.2 Key agreement protocols

2.2.1 ECDH: Elliptic curve Diffie-Hellman

The elliptic curve Diffie-Hellman scheme (ECDH) as specified in [IEE00, ANS01] derives a shared secret value k between two communication partners A and B using the public and private keys of the entities. The protocol is presented in algorithm 2.2.1.
In steps 5 and 6, he two communication partners have computed the same point S on E since $S = r_A R_B = r_A r_B G = r_B R_A$. The protocol requires about ℓ bits to be transferred in each direction.

Algorithm 2.2.1 ECDH key agreement

1. The two parties first agree on a set of domain parameters $(\mathbb{F}_p, E, n, h, G)$.
2. A selects her private key r_A as a random value $r_A \in [\![1, n-1]\!]$ and computes her public key $R_A = r_A G$.
3. B follows the same steps and computes his public key $R_B = r_B G$ via the private key $r_B \in [\![1, n-1]\!]$.
4. The parties exchange their public keys.
5. A computes $S = r_A R_B$.
6. B computes $S = r_B R_A$.
7. A and B compute the hash value of the x-coordinate of S as the shared secret k : $k = \mathrm{Hash}(x(S))$.

The point S is only a *shared secret*: it may have some weak bits and thus should not be used directly as a key. Instead, a key derivation function should be used to even out the entropy of the key.
The public keys should belong to the large subgroup of size n. Intentionally or unintentionally, elements of small order (h or a divisor of h) might be chosen as public keys. This would quickly disclose the key. The *cofactor DH scheme* omits this vulnerability by multiplying with h to obtain K. The two parties compute $K = hr_A R_B$ and $K = hr_B R_A$, respectively. If R_B or R_B are of small order then $K = \mathcal{O}$. A test for $hR_B \neq \mathcal{O}$ and $hR_A \neq \mathcal{O}$ is necessary.

It should also be noted that the basic ECDH protocol is vulnerable to a simple "man-in-the-middle" attack. To protect against this, the public keys R_A and R_B may be signed by Alice and Bob.
The ECDH protocol, plus some level of key certification, is the basis for all following key agreement schemes.

2.2.2 ECSTS

Here, KDF is a key derivation function, and Sign is a signature protocol. A symmetric encryption function E is also used.

Algorithm 2.2.2 ECSTS key agreement

1. A chooses a random $r_A \in [\![1, n-1]\!]$, and sends $R_A = r_A \cdot G$. B does likewise.
2. Both partners compute the shared secret $S = r_A R_B = r_B R_A$ and the shared key $k = \mathsf{KDF}(S)$.
3. A sends $\mathsf{Enc}_k(\mathsf{Sign}_A(R_A \parallel R_B))$.
4. B sends $\mathsf{Enc}_k(\mathsf{Sign}_B(R_B \parallel R_A))$.
5. Both parties decrypt using the shared key k and check signatures.

This protocol needs two rounds of transfer in each direction: ℓ bits for the shared secret and then 2ℓ bits for the signature.
It should be noted that the shared key k generated should be separated in two parts $k_1 \parallel k_2$, with the key k_1 used to encrypt the signautres in steps 3

and 4, and the key k_2 being the key used after the protocol. Therefore, the shared secret S should have an entropy of 2ℓ bits.

2.2.3 ECMQV

ECMQV [LMQ$^+$03] adds a layer of protection atop the ECDH protocol without the extra cost incurred by straightforward signed ECDH variants. Each entity A is now supposed to own the long-term public-private key pair $(d_A, D_A = d_A \cdot G)$. Let $f : E(k) \longrightarrow [\![1, \sqrt{n}]\!]$ be an "almost surjective" function (for instance, [LMQ$^+$03] uses $f(P) = x(P) \pmod{2^{\ell/2}} + 2^{\ell/2}$). Then A knows the secret value $s_A = r_A + f(R_A)d_A \pmod{n}$, which is associated to the public point $S_A = s_A \cdot G = R_A + f(R_A) \cdot D_A$. The two partners can then use the shared secret $s_A S_B = s_B S_A = s_A s_B G$.

Algorithm 2.2.3 ECMQV key agreement

1. A chooses a random $r_A \in [\![0, n-1]\!]$, computes $R_A = r_A \cdot G$ and sends it to B.
2. B chooses a random $r_B \in [\![0, n-1]\!]$, computes $R_B = r_B \cdot G$ and sends it to A.
3. A computes the secret value $s_A = (r_A + f(R_A)d_A) \pmod{n}$ and the public value $S_B = R_B + f(R_B) \cdot D_B$.
4. B computes the secret value $s_B = (r_B + f(R_B)d_B) \pmod{n}$ and the public value $S_A = R_A + f(R_A) \cdot D_A$.
5. The shared secret is $K = s_A \cdot S_B = s_B \cdot S_A$.

The bandwith need of this protocol is the same as that of the ECDH protocol. The size of the destination set of the function f is $\sqrt{n}$, which is in accord with the current hardness of the discrete logarithm problem. Moreover, having $f(P)$ of size about half that of n allows us to perform the exponentiation in the computation of S_A, S_B as only a half-exponentiation. Thus, the only extra cost for this protocol (comparated to ECDH) is one half-exponentiation for each partner.
Some impersonation, unknown key share, and man-in-the-middle attacks may be performed against MQV. For this reason, MQV has been dropped from the NSA Suite B protocol list.

2.2.4 ECHMQV: Hashed MQV

This protocol, described in [Kra05] is the same as the basic MQV protocol, except that the function f used to compute the secrets is now a hash function. This solves some of the weaknesses of MQV.
In the random oracle model, HMQV achieves weak forward secrecy (that is, the current secret is not threatened by possible future public key disclosure) and is secure against impersonation and temporary secret disclosure.

2.2.5 ECKMQV: Korean MQV

This protocol is described in [JKL06]. It achieves the same security features as HMQV, with slightly less assumptions about the hash function involved; however, the bandwidth is twice that of ECDH.

Algorithm 2.2.4 ECKMQV key agreement

1. A chooses a random $r_A \in [\![1, n-1]\!]$ and sends $R_A = r_A \cdot G$; B does likewise.
2. A computes $k_A = \mathsf{Hash}(d_A \cdot R_B)$ and sends $t_A = \mathsf{MAC}_{k_A}(A \parallel B \parallel R_A)$.
3. A computes $k_B = \mathsf{Hash}(r_A \cdot D_B)$ ($k_B \neq k_A$) and checks that $t_B = \mathsf{MAC}_{k_B}(B \parallel A \parallel R_B)$.
4. The shared secret is $\mathsf{Hash}(r_A \cdot R_B) \oplus \mathsf{Hash}(d_A \cdot D_B)$.

2.2.6 FHMQV: Fully Hashed MQV

This protocol [SEVB09], based on HMQV, protects against consequences of leakage of the secret exponent by replacing $f(R_A)$ and $f(R_B)$ by $\mathsf{Hash}(R_A \parallel R_B \parallel A \parallel B)$.

2.3 Key /data encapsulation

While asymmetric cryptography has theoretic advantages, its implementation is quite slow and it is therefore ill-fitted to long message encryption. Therefore it is advisable to split its use in two parts:

– a *key encapsulation mechanism* (KEM), in which the public key D_B of the receiver is used to compute a key K for a symmetric encryption algorithm; the key K is then sent to B in an encapsulated manner that only he can decypher;
– a *data encapsulation mechanism* (DEM) in which the symmetric key K and a symmetric encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ are used to convey the data itself.

The key K should be of sufficient size to ensure resistance of the symmetric encryption scheme.

As the only part of the KEM/DEM scheme concerned about elliptic curves is the KEM, this is the only part that will be discussed here.

2.3.1 Elliptic curve integrated encryption system (ECIES-KEM)

The elliptic curve integrated encryption system (ECIES) is described in [AYZ98, ANS01, IEE00, Sho01, BSSC05].

After execution of encapsulation, A may then immediately send the cyphertext $c = \mathsf{Enc}_K(m)$ to B.

To prevent against adaptive chosen-cyphertext attacks, the secret key K should actually be split in two parts $k_1 \parallel k_2 = \mathsf{KDF}(S \parallel R_A, \ell)$, with k_1 being

Algorithm 2.3.1 ECIES-KEM encapsulation

1. The sender A chooses a random $r_A \in [\![1, p-1]\!]$.
2. A computes $R_A = r_A \cdot G$.
3. A computes $S = r_A \cdot D_B$.
4. A computes $K = \mathsf{KDF}(S \parallel R_A, \ell)$.
5. The encapsulation is R_A; the secret key is K.

the symmetric key used for sending the message, and k_2 used for authenticating it by sending $\mathsf{MAC}_{k_2}(c)$ with the cyphertext.

Algorithm 2.3.2 ECIES-KEM decapsulation

1. B receives the encapsulation R_A
2. B computes $S_B = d_B \cdot R_A$.
3. The secret key is $K = \mathsf{KDF}(S_B \parallel R_A)$.

We have $S_B = d_B \cdot R_A = d_B r_A \cdot G = S$, and therefore A and B both compute the same key K. Therefore, B is able to recover the plaintext $m = \mathsf{Dec}_K(c)$. This scheme is resistant to chosen cyphertext attack under the gap Diffie-Hellman problem on E [pse08].

Variation Instead of using the point T, we can also restrict ourselves to the x-coordinate of the point as done in various standards. This causes the problem that (U, c, r) and $(-U, c, r)$ are valid ciphertexts for the same message as the x- coordinate of U and $-U$ is the same. The problem is called *benign malleability* and the scheme is then formally not secure against adaptive chosen ciphertexts in the complete sense.
To avoid a small subgroup one can choose to apply KDF to hT. If T is point of small order h, the product will evaluate to the point at infinity.

2.3.2 PSEC-KEM

This scheme is described in [pse08]. It is designed to reject improperly formed cyphertexts. It is based on the plain ECIES-KEM and may actually use it as a subroutine in its implementation. However, the decapsulation algorithm is slower than ECIES by one curve multiplication, and the bandwidth required for KEM is twice that of ECIES.
Here $\mathsf{KDF}_i(P, \ell)$ stands for $\mathsf{KDF}(i_{32} \parallel P, \ell)$, where i_{32} is the 32-bit integer i.

This scheme is resistant to chosen cyphertext attack under the computational Diffie-Hellman problem [pse08].

3 Crypto-processor functionalities state-of-the-art

We list the operations required to implement the previously presented cryptographic algorithms. We furthermore give an overview of the existing

Algorithm 2.3.3 PSEC-KEM encapsulation

1. Select a random seed ρ.
2. Set $(r_A \parallel K) = \mathsf{KDF}_0(\rho)$, where r_A is an integer (of size ℓ) reduced modulo n.
3. Compute $R_A = r_A \cdot G$.
4. Compute $S = r_A \cdot D_B$.
5. Compute $T = \rho \oplus \mathsf{KDF}_1(S \parallel R_A)$.
6. The key is K; the encapsulation is (R_A, T).

Algorithm 2.3.4 PSEC-KEM decapsulation

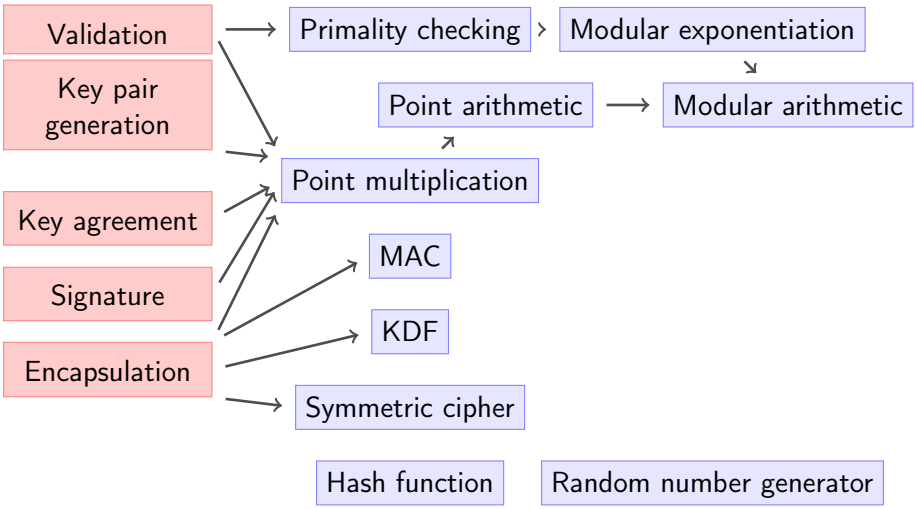
1. Receive the encapsulation as (R_A, T).
2. Compute $S = d_B \cdot R_A$.
3. Compute $\rho = T \oplus \mathsf{KDF}_1(S \parallel R_A)$.
4. Compute $(r_A \parallel K) = \mathsf{KDF}_0(\rho)$.
5. Check $r_A \cdot G = R_A$.
6. The encapsulated key is K.

crypto-coprocessors used to perform public key algorithms such as RSA and those based on elliptic curves.

3.1 Operations needed

As most of the previous protocols use the same basic functions, a summary of recommended cryptographic services and related base functions is presented in figure 3.1.1.

Figure 3.1.1 Required services and functions



Point arithmetic operations include point checking (testing if a point belongs to the curve), addition, duplication, conversion between various coordinate system, compression/decompression, multiplication, and com-

parison. In particular, random point generation is not used in any of the protocols above.
Modular arithmetic operations include modular reduction, addition, subtraction, complement, multiplication, short multiplication, squaring, inversion, exponentiation, and square root.

3.2 Co-processors and operations needed

The minima resources in terms of software or/and hardware to be able to reach acceptable performances for elliptic curve algorithms are listed below:

1. A random number generator
2. Modular multiplication, modular addition and modular subtraction
3. Hash function (Rotation and logical operation on 32 bits)
4. Modular exponentiation

According to the crypto-coprocessor in the component, an integer division and a reduction could be also necessary. For some chips, the crypto-coprocessor embeds all these operations but for some they will be coded in part by the CPU.

3.3 Characteristics and functionalities

In a component, crypto-coprocessors are peripherals; they could have their own clock. They could be programmed in a specific language.

1. The modular exponentiation $M^d \mod n$ thanks to the operation $XY \mod n$
2. The scalar multiplication $Q = dP$ thanks to modular addition, subtraction and multiplication over $\mathbb{F}_p$ or $\mathbb{F}_{2^n}$

Moreover, most cryptoprocessors offer arithmetic and boolean operations with large numbers that enable accelerating long data transfers when compared to classical 8-bit or 16-bit CPU process.
However, none of the crypto-coprocessors used in smart cards embeds full hardware modular exponentiation or scalar multiplication. Regarding physical behaviour, the power consumption and electromagnetic/radio frequency emanation is very important compared to that of the CPU.

3.4 Operand constraints

According to the hardware design of the crypto-coprocessor, the operands are handled in various ways. A first kind of architecture is the RAM or EEPROM pointers access. In this case, to manage the operands the crypto-coprocessor user has to handle specific memory addresses and data size via specific function registers. The RAM is shared by the crypto-coprocessor

and the CPU and sometimes a faster RAM access is implemented at hardware level. The operands sizes are limited to the RAM size.
A second kind of design is characterized by internal registers only dedicated to the crypto-coprocessor operands. If the operands exceed the maximum value handled by these registers, then the available hardware operation cannot be used anymore. A mix of the two previous architectures is conceivable and is defined by operands mapped at RAM fixed addresses. The size of the operands is limited by the interval free between each fixed address.

3.5 Modular multiplication

Several kind of modular multiplications are available in the various crypto-coprocessors. Indeed, Quisquater, Montgomery, Sedlack or ZDN modular multiplications are available. For details the reader could have a look at following references [Dhe98, NM96] and [Sed88] (XXX very old references, what is the state-of-the-art here?, D98 is only a master thesis, is there a conference publication?)

3.6 Other functionalities

For security and performance reasons, CPU and crypto-coprocessors could work in parallel. Operations on large numbers other than modular multiplication might be available. Modular inversion, addition, subtraction, logical operations are often present. As elliptic curves become popular, the crypto-coprocessor might embed specific operations over $\mathbb{F}_{2^p}$ and modular addition and subtraction over the integers. However, point addition, point doubling, scalar multiplication or modular exponentiations are not present in the current crypto-coprocessors.

References

[ANS99] ANSI X9.62. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, American National Standards Institute edition, 1999.

[ANS01] ANSI X9.63. *Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American National Standards Institute edition, January 2001. `http://grouper.ieee.org/groups/1363/private/x9-63-01-08-99.pdf`.

[AYZ98] Michel Abdalla, Mihir Bellare Y, and Phillip Rogaway Z. Submission to IEEE P1363a. DHAES: An encryption scheme based on the diffie-hellman problem, 1998.

[BBJ+08] Daniel Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted edwards curves. In Serge Vaudenay, editor, *Progress in Cryptology – AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-68164-9_26.

[BBLP07] D.J. Bernstein, P. Birkner, T. Lange, and C. Peters. Optimizing double-base elliptic-curve single-scalar multiplication. In *Proceedings of the cryptology 8th international conference on Progress in cryptology*, pages 167–182. Springer-Verlag, 2007.

[BGV94] A. Bosselaers, R. Govaerts, and J. Vandewalle. Comparison of three modular reduction functions. In *Advances in Cryptology–CRYPTO'93*, pages 175–186. Springer, 1994.

[BHLM01] M. Brown, D. Hankerson, J. López, and A. Menezes. Software implementation of the NIST elliptic curves over prime fields. *Topics in Cryptology–CT-RSA 2001*, pages 250–265, 2001.

[BJ03] E. Brier and M. Joye. Fast point multiplication on elliptic curves through isogenies. *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 603–603, 2003.

[BKM09] J.W. Bos, M.E. Kaihara, and P.L. Montgomery. Pollard rho on the PlayStation 3. *SHARCS'09 Special-purpose Hardware for Attacking Cryptographic Systems*, page 35, 2009.

[BL10] D. J. Bernstein and T. Lange. Explicit formula database. `http://www.hyperelliptic.org`, 2010.

[BSSC05] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.

[CFA06] Henri Cohen, Gerhard Frey, and Roberto. Avanzi. *Handbook of elliptic and hyperelliptic curve cryptography / [editors], Henri Cohen,*

Gerhard Frey ; [authors], Roberto Avanzi ... [et al.]. Chapman & Hall/CRC, Boca Raton :, 2006.

[Cop93] D. Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, 6(3):169–180, 1993.

[Dhe98] J.F. Dhem. *Design of an efficient public key cryptographic public key for RISC based architecture*. PhD thesis, Faculté des sciences appliquées laboratoire microélectronique, May 1998.

[Die03] C. Diem. The GHS attack in odd characteristic. *JOURNAL-RAMANUJAN MATHEMATICAL SOCIETY*, 18(1):1–32, 2003.

[DIM08] V. Dimitrov, L. Imbert, and P.K. Mishra. The double-base number system and its application to elliptic curve cryptography. *Mathematics of Computation*, 77(262):1075–1104, 2008.

[DLP93] I. Damgård, P. Landrock, and C. Pomerance. Average case error estimates for the strong probable prime test. *Math. Comp.*, pages 177–194, 1993.

[EG02] A. Enge and P. Gaudry. A general framework for subexponential discrete logarithm algorithms. *Acta Arith*, 102(1):83–103, 2002.

[FIP00] *FIPS PUB 186-2: Digital Signature Standard (DSS)*, National Institute of Standards and Technology edition, January 2000.

[GHS02] P. Gaudry, F. Hess, and N.P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15(1):19–46, 2002.

[GL92] S. Gao and H.W. Lenstra. Optimal normal bases. *Designs, Codes and Cryptography*, 2(4):315–323, 1992.

[Gor93] D.M. Gordon. Discrete Logarithms in GF(P) Using the Number Field Sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138, 1993.

[IEE00] IEEE 1363. *Standard Specification for Public Key Cryptography*, the institute of electrical and electronics engineers, inc. (ieee) p1363 edition, 2000.

[ISO00] *International Standard 15946-2: Information Technology — Security Techniques — Cryptographic techniques based on elliptic curves — Part 2: International Standard 15946-2: Information Technology — Security Techniques — Cryptographic techniques based on elliptic curves — Part 2: Digital Signatures*, international standards organization edition, 2000.

[ISO02] Iso-iec 15946-2. information technology — security techniques — cryptographic signatures based on elliptic curves — part 2: digital signatures. Technical report, 2002.

[JKL06] I. Jeong, J. Kwon, and D. Lee. A Diffie-Hellman key exchange protocol without random oracles. *Cryptology and Network Security*, pages 37–54, 2006.

[JMV04] D. Jao, S.D. Miller, and R. Venkatesan. Ramanujan graphs and the random reducibility of discrete log on isogenous elliptic curves. 2004.

[KMV00] N. Koblitz, A. Menezes, and S. Vanstone. The state of elliptic curve cryptography. *Designs, Codes and Cryptography*, 19(2):173–193, 2000.

[Kra05] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology–CRYPTO 2005*, pages 546–566. Springer, 2005.

[LMQ+03] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[LS01] P. Liardet and N. Smart. Preventing SPA/DPA in ECC systems using the Jacobi form. In *Cryptographic Hardware and Embedded Systems–CHES 2001*, pages 391–401. Springer, 2001.

[Men08] A. Menezes. The elliptic curve discrete logarithm problem: State of the art. *Advances in Information and Computer Security*, pages 218–218, 2008.

[MLS03] J. Malone-Lee and N. Smart. Modifications of ECDSA. In *Selected Areas in Cryptography*, pages 1–12. Springer, 2003.

[Mon85] P.L. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[Mon87] P.L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[MW00] U.M. Maurer and S. Wolf. The Diffie–Hellman Protocol. *Designs, Codes and Cryptography*, 19(2):147–171, 2000.

[NM96] D. Naccache and D. M'Raihi. Arithmetic co-processors for public-key cryptography: The state of the art. *IEEE Micro*, 16(3), 1996.

[NSA08] NSA. Mathematical routines for the nist prime elliptic curves. `http://www.nsa.gov/ia/_files/nist-routines.pdf`, 2008.

[pse08] PSEC-KEM specification (version 2.2). 2008.

[Qui90] J.J. Quisquater. Fast modular exponentiation without division. *Rump session of EUROCRYPT*, 90, 1990.

[SEC00] Sec 2: recommended elliptic curve parameters. Technical report, The SEC group, 2000.

[Sed88] Holger Sedlak. The RSA cryptography processor. In *EUROCRYPT'87: Proceedings of the 6th annual international conference on Theory and application of cryptographic techniques*, pages 95–105, Berlin, Heidelberg, 1988. Springer-Verlag.

[SEVB09] A. Sarr, P. Elbaz-Vincent, and J.C. Bajard. A Secure and Efficient Authenticated Diffie-Hellman Protocol. 2009.

[Sho01] Victor Shoup. A proposal for an iso standard for public key encryption (version 2.0), 2001.

[Sol99] Jerome A. Solinas. Generalized mersenne numbers. Technical Report CORR 99–39, University of Waterloo, 1999.

[Thé03] N. Thériault. Index calculus attack for hyperelliptic curves of small genus. *Advances in Cryptology-ASIACRYPT 2003*, pages 75–92, 2003.

[The05] The Brainpool consortium. ECC Brainpool Standard Curves and Curve Generation v1.0. `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`, 2005.

[Vau04] S. Vaudenay. Digital signature schemes with domain parameters: Yet another parameter issue in ECDSA. In *In Proceedings of the 9th Australasian Conference on Information Security and Privacy*. Citeseer, 2004.