# Security considerations for elliptic curve domain parameters selection

**Thesis** · June 2018

DOI: 10.13140/RG.2.2.35969.89449

Masaryk University
Faculty of Informatics


# Security considerations for elliptic curve domain parameters selection

Bachelor's Thesis

**Ján Jančár**

Brno, Spring 2018

Masaryk University
Faculty of Informatics


# Security considerations for elliptic curve domain parameters selection

Bachelor's Thesis

**Ján Jančár**

Brno, Spring 2018

*This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.*

## Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ján Jančár

**Advisor:** RNDr. Petr Švenda, Ph.D.
**Consultant:** Mgr. Marek Sýs, Ph.D.

## Acknowledgements

A great thanks to my advisor RNDr. Petr Švenda, Ph.D. for introducing me to elliptic curve cryptography head first and enabling the following research endeavour.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme 'Projects of Large Research, Development, and Innovations Infrastructures' (CESNET LM2015042), is greatly appreciated.

# Abstract

We present a summary of currently known security requirements of elliptic curve domain parameters together with a discussion of various methods for generating elliptic curve domain parameters, such as the exhaustive approach, verifiably random algorithms or a method using complex multiplication. We demonstrate a tool we built for fast generation of elliptic curve domain parameters using these methods.

We analyze potential elliptic curve cryptography implementation pitfalls, their consequences on security and present a tool for thorough blackbox testing of elliptic curve cryptography implementations on smart cards and in software libraries.

## Keywords

elliptic curve cryptography, elliptic curve domain parameters, ECDLP, ECDH, ECDSA, smart cards, JavaCard, cryptographic libraries

# Contents

# Introduction

Nowadays, elliptic curves see more and more use in public key cryptography, usually because of their performance and small key sizes. All standard elliptic curve based cryptographic algorithms require a set of public domain parameters, that specify a concrete curve over which computations are performed. These parameters are more complex than those of finite-field based algorithms such as DSA or Diffie-Hellman. Creating such parameters is non-trivial, both computationally and implementation complexity-wise. There are many requirements they need to follow, for them to be secure against known attacks and even implementation errors. That is why most protocols and implementations use parameters established in some of the public standards concerning the selection of secure elliptic curve parameters.

Many such standards exist with many curves defined in each one of them, however, reasons to generate elliptic curve domain parameters still remain. Generating curves with certain properties is useful for implementation testing. Some attacks require the attacker to pre-generate a set of curves with specific properties. Also, after the disclosure of the possible backdoor in the DUAL_EC_DRBG pseudo-random number generator, trust in some of the parameters was generally lost [1]. Thus there is a need for a fast tool that can generate a wide variety of elliptic curve domain parameters.

Given the complex nature of implementing elliptic curve cryptographic algorithms and the history of errors in implementations [2], a tool for thorough testing of elliptic curve cryptography implementations might uncover even more errors and possible vulnerabilities.

The goal of this thesis is to summarize current known security requirements of elliptic curve domain parameters, their generation methods, possible elliptic curve cryptography implementation pitfalls and implement a tool that generates elliptic curve domain parameters as well as a tool that is able to test elliptic curve cryptography implementations.

### Previous work

There is a sizable amount of previous work that can be categorized as:

- Standards which establish concrete elliptic curve generation algorithms, usually together with a set of generated and named parameters to be used. [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
- Publications which develop novel elliptic curve generation methods. Schoof's algorithm for counting points on elliptic curves [14] was the

first polynomial time algorithm, which made creation of curves of large point counts practical. Schoof-Elkies-Atkin (SEA) algorithm is an extension of Schoof's that significantly improved its efficiency. The introduction of the complex multiplication method for creating elliptic curves with a known number of points of cryptographic sizes created another method [15]. There was also work building on top of these generation methods to create methods that respect a set of known security requirements [16, 17].

- Publications which present new weaknesses on classes of elliptic curves, thus adding to the set of known security requirements. Also, publications that report new vulnerabilities in implementations and thus point to common implementation pitfalls [18].
- Software implementing algorithms for generating elliptic curve domain parameters. [19, 20, 21, 22]

### Our contribution

We built a command-line tool for generating elliptic curve domain parameters, called ecgen, which implements many different generation methods and checks for known security requirements:

- Random exhaustive approach for generating domain parameters.
- ANSI X9.62 verifiably random algorithm.
- Brainpool verifiably random algorithm.
- IEEE P1636 complex multiplication algorithm for generating domain parameters.
- Custom complex multiplication algorithm.

We also developed ECTester, a tool for black-box testing elliptic curve cryptography implementations on smart cards and in software libraries. This tool uses the domain parameters generated by ecgen to perform thorough implementation testing.

### Overview

We provide an overview of elliptic curves and their domain parameters in chapter 1. Popular forms of elliptic curves used in cryptography are described in section 1.1. Security requirements against known attacks are presented

in section 1.3. Section 1.4 describes vulnerabilities which may arise due to common implementation bugs, even when working with otherwise secure parameters.

In chapter 2 we outline the possible methods of generating elliptic curve domain parameters. The method usually called *random approach* along with point counting algorithms is described in section 2.1. Methods which claim to be *verifiably random* are summarized in section 2.2. The real randomness and security of these algorithms is analyzed in subsection 2.2.3. The *complex multiplication method* is introduced in section 2.3.

Section 2.4 presents a tool for generating elliptic curve domain parameters, called ecgen. It's features, architecture and implementation are described in subsection 2.4.1. Example results and timings are presented in subsection 2.4.2.

A tool for testing elliptic curve cryptography implementations in JavaCards and in software libraries, called ECTester, is presented in chapter 3. Description of the tool concerned with JavaCard testing is given in section 3.1. Testing of selected software libraries is shown in section 3.2. The platform neutral test suites are specified in section 3.3.

# 1 Elliptic curves

An elliptic curve is a non-singular plane algebraic curve of genus one, an elliptic curve over a field $\mathbb{K}$ is defined as follows:

$$E/\mathbb{K}: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1.1}$$

with $\mathcal{O} = \infty$ the point at infinity, and $a_1, \ldots, a_6 \in \mathbb{K}$. We can further define the *discriminant* of the curve, denoted $\Delta$ as follows [23]:

$$\begin{aligned} \Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{aligned} \tag{1.2}$$

The curve is non-singular, if and only if the discriminant $\Delta$ is nonzero.

A non-singular elliptic curve is determined, up to isomorphism over $\overline{\mathbb{K}}$, by its *j-invariant* which is given by:

$$j(E) = (d_2^2 - 24d_4)^3/\Delta \tag{1.3}$$

Thus we can introduce a change of variables that creates a new curve isomorphic over $\mathbb{K}$:

$$\begin{aligned} (x, y) \mapsto (u^2x + r, u^3y + u^2sx + t) \\ (u, r, s, t) \in \mathbb{K}^* \times \mathbb{K}^3 \end{aligned} \tag{1.4}$$

When we let $(u, r, s, t) \in \overline{\mathbb{K}}^* \times \overline{\mathbb{K}}^3$ then a change of variables creates a curve isomorphic over $\overline{\mathbb{K}}$, often called a *quadratic twist*. Under certain conditions, the curve can be isomorphic over $\mathbb{K}$ itself, meaning we obtained a trivial quadratic twist.

An elliptic curve is also an *abelian variety*, meaning that with operation defined as point addition of its elements and the point at infinity $\mathcal{O}$ it forms an abelian group. Furthermore, one can limit the points coordinates to any extension field $\mathbb{L}$ of field $\mathbb{K}$ and thus get the set of $\mathbb{L}$-rational points on $E/\mathbb{K}$ [24]:

$$E(\mathbb{L}) = \{(x, y) \in \mathbb{L}^2 : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\mathcal{O}\} \tag{1.5}$$

The structure of the group of $\mathbb{F}_q$-rational points of a curve defined over a finite field $\mathbb{F}_q$ is easy to describe. It is either cyclic or isomorphic to a product of two cyclic groups:

$$E(\mathbb{F}_q) \simeq \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \tag{1.6}$$

with $n_1|q-1$ and $n_1|n_2$ [24].

Given two-points $P=(x_1,y_1), Q=(x_2,y_2)$ in $E(\mathbb{L})$ we can perform point addition as follows:

$$\begin{aligned} P+Q &= (x_3,y_3) \\ \lambda &= \begin{cases} \dfrac{y_1-y_2}{x_1-x_2} & \text{if } P \neq \pm Q \\ \dfrac{3x_1^2+2a_2x_1+a_4-a_1y_1}{2y_1+a_1x_1+a_3} & \text{if } P = Q \end{cases} \\ x_3 &= \lambda^2+a_1\lambda-a_2-x_1-x_2 \\ y_3 &= \lambda(x_1-x_3)-y_1-a_1x_3 \end{aligned} \tag{1.7}$$

Then defining $P-P=\mathcal{O}$ gives a group of $\mathbb{L}$-rational points on $E/\mathbb{K}$ [23].

The geometric representation of point addition on an example curve $E/\mathbb{R}$ can be seen in fig. 1.1.

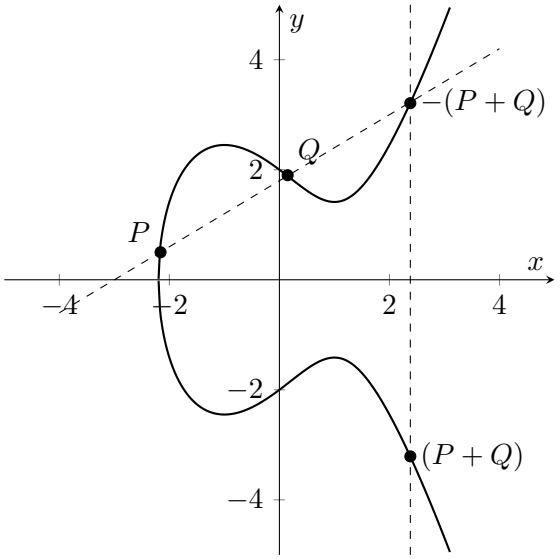


Figure 1.1: Diagram of point addition on $E/\mathbb{R}: \; y^2 = x^3 - 3x + 4$

For cryptographic uses, curves over finite fields $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$ with $p$ prime and $m \in \mathbb{N}$ are the most interesting. Point addition on these curves is not

so nicely visible in a plot of such a curve, however the same principle and addition laws apply.

Applying point addition multiple times naturally creates an operation of *scalar multiplication*, for $n \in \mathbb{N}^*$:

$$\begin{aligned} [n] : E(\mathbb{K}) &\to E(\mathbb{K}) \\ P &\mapsto [n]P = \underbrace{P + P + \ldots + P}_{n\text{times}} \end{aligned} \tag{1.8}$$

An example of a scalar multiplication algorithm is the double-and-add algorithms. It is similar to the square-and-multiply algorithm, used for exponentiation, however, it is in additive notation.

**Algorithm 1** Right-to-left double-and-add scalar multiplication [23]

```
▷ k = (k_{t-1}, ..., k_1, k_0)_2 is the scalar.
▷ P ∈ E(K) is the point to be multiplied.
function MULTIPLY(k, P)
    Q ← O
    for i from 0 to t − 1 do
        if k_i = 1 then
            Q ← Q + P
        P ← [2]P
    return Q
```

Looking at the kernel of scalar multiplication is also useful as it gives a group $E[n]$ of *n-torsion points* of the curve.

$$E[n] = \{Q \in E(\overline{\mathbb{K}}) \mid [n]Q = \mathcal{O}\} \tag{1.9}$$

If the characteristic of $\mathbb{K}$ is prime to $n$ the group is isomorphic to $\mathbb{Z}_n \times \mathbb{Z}_n$. Otherwise the group is either equal to $\{\mathcal{O}\}$ or isomorphic to $\mathbb{Z}_n$.

The Elliptic Curve Discrete Logarithm Problem (ECDLP) can be stated as follows (adapted from generalized discrete logarithm problem from Menezes et al. [25]):

**Definition 1** (Elliptic curve discrete logarithm problem)**.** Given a finite cyclic group of points $G$ of order $n$ of an elliptic curve over a finite field, a generator $A$ of $G$, and an element $B \in G$. Find the integer $x$, $0 \leq x \leq n-1$, such that $[x]A = B$.

The intractability of the ECDLP is key to all common elliptic curve cryptosystems. More specifically, the intractability of Elliptic Curve Diffie-

Hellman Problem (ECDHP), which is closely tied to ECDLP and is the following:

**Definition 2** (Elliptic curve Diffie-Hellman problem)**.** Given a finite cyclic group of points $G$ of an elliptic curve over a finite field, a generator $A$ of $G$, and group elements $[a]A$ and $[b]A$, find $[a+b]A$.

## 1.1 Equation forms

Equation 1.1 gives the full Weierstrass form of an elliptic curve. This is the most general form of elliptic curves. However several other forms, other representations which do not fundamentally alter the structure of the elliptic curve, exist and are used in cryptography. Such as Edwards form, Montgomery form, Hessian or other forms. These forms have different addition laws, some of them are complete and some offer better performance, with addition laws consisting of fewer operations.

We focus on the short Weierstrass form, as most standards specify curves in short Weierstrass form. Nonetheless we describe these forms in this section. For more information regarding actual implementation details and addition formulas we refer to the Explicit-Formulas Database [26].

### 1.1.1 Weierstrass form

The full Weierstrass equation is the most general form of an elliptic curve. It's discriminant and $j$-invariant is given by eqs. (1.2) and (1.3).

$$E/\mathbb{K}: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1.10}$$

In projective coordinates it leads to:

$$E/\mathbb{K}: \quad Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{1.11}$$

### 1.1.2 Short Weierstrass form

The full Weierstrass equation can always be simplified, depending on the characteristic of the underlying field to shorter and simpler forms [24].

If the underlying field has characteristic $\neq 2, 3$ the short Weierstrass form is the following:

$$\begin{aligned} E/\mathbb{F}_p: \quad y^2 &\equiv x^3 + a_4x + a_6 \\ \Delta &= -16(4a_4^3 + 27a_6^2) \\ j &= 1728a_4^3/4\Delta \end{aligned} \tag{1.12}$$

For underlying field of characteristic 2 the short Weierstrass form depends on the $j$-invariant of the curve.

$$\begin{aligned} E/\mathbb{F}_{2^m}: \quad y^2 + xy &\equiv x^3 + a_2x^2 + a_6 \\ \Delta &= a_6 \\ j &= 1/a_6 \end{aligned} \tag{1.13}$$

With $j$-invariant equal to 0 a curve over $\mathbb{F}_{2^m}$ is supersingular and not practically used in cryptography.

$$\begin{aligned} E/\mathbb{F}_{2^m}: \quad y^2 + a_3y &\equiv x^3 + a_4x + a_6 \\ \Delta &= a_3^4 \\ j &= 0 \end{aligned} \tag{1.14}$$

### 1.1.3 Montgomery form

The Montgomery form was first introduced to speed up the elliptic curve factorization method by Montgomery [27]. It allows for addition and doubling in projective coordinates using only a single coordinate. The addition formulas are also faster than short Weierstrass ones, requiring fewer operations.

However, not every curve over a prime field in short Weierstrass form can be transformed into Montgomery form [24]. It is possible if and only if :

- $x^3 + a_4x + a_6$ has at least one root $\alpha$ in $\mathbb{F}_p$
- $3\alpha^2 + a_4$ is a quadratic residue in $\mathbb{F}_p$

Then, the Montgomery form is given by:

$$E/\mathbb{F}_p: \quad By^2 = x^3 + Ax^2 + x \tag{1.15}$$

The fast, single-coordinate addition formulas from Montgomery form were adapted to the short Weierstrass form by Brier; Joye [28] for curves over $\mathbb{F}_p$ and by López; Dahab [29] for curves over $\mathbb{F}_{2^m}$.

### 1.1.4 Edwards form

Edwards form first appeared in Edwards [30]. While it technically is not an elliptic curve, because it has singularities at some points, removing those singularities makes it birationally equivalent to a Weierstrass curve [31].

For field $\mathbb{K}$ with characteristic $\neq 2$ the general Edwards form is the following:

$$\begin{aligned} E/\mathbb{K}: \quad x^2 + y^2 &= c^2(1 + dx^2y^2) \\ cd(1 - dc^4) &\neq 0 \end{aligned} \tag{1.16}$$

Originally introduced with $d = 1$ , and generalized to the aforementioned form by Bernstein; Lange [31]. However, all curves of that form are isomorphic to curves of the form:

$$E/\mathbb{K}: \quad x^2 + y^2 = 1 + dx^2y^2 \tag{1.17}$$

Not every Weierstrass curve is birationally equivalent to an Edwards curve. Given an elliptic curve $E/\mathbb{K}$ over a field $\mathbb{K}$ with characteristic $\neq 2$ having a point of order 4, there is an Edwards curve so that some quadratic twist of $E/\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to this Edwards curve.

## 1.2 Domain parameters

For two (or more) parties, to participate in an elliptic curve based cryptographic protocol, they need to share a set of public parameters uniquely identifying a group of points on an elliptic curve on which computations will be performed.

These parameters are similar to parameters for discrete logarithm problem based cryptographic methods. Several standards [4] [10] [6] define the form of these domain parameters, some even in ASN.1 notation. Luckily they mostly agree on the form.

The parameters differ for curves defined over a prime field and a binary field, we describe both.

### 1.2.1 Prime field

The short Weierstrass equation for an elliptic curve over a prime field is given:

$$E/\mathbb{F}_p: \quad y^2 \equiv x^3 + ax + b \mod p \tag{1.18}$$

where the discriminant of the curve:

$$\Delta = -16(4a^3 + 27b^2) \tag{1.19}$$

is nonzero, meaning the curve is non-singular.

Then the domain parameters of an elliptic curve over a prime field are usually a sextuple [10]:

$$(p, a, b, G, n, h)^1 \tag{1.20}$$

- $p$ is a prime, the characteristic of the finite field $\mathbb{F}_p$ over which the curve is defined.

1. Sometimes written as $(p, a, b, G, r, k)$.

- $a$ is an element of $\mathbb{F}_p$.
- $b$ is an element of $\mathbb{F}_p$.
- $G$ is a point on the curve $E/\mathbb{F}_p$ called the generator or sometimes the base-point.
- $n$ is the order of the generator $G$.
- $h$ is the quantity $\frac{1}{n}|E(\mathbb{F}_p)|$ also called the cofactor.

### 1.2.2 Binary field

The short Weierstrass equation for an elliptic curve over a binary field is given:

$$E/\mathbb{F}_{2^m}: \quad y^2 + xy \equiv x^3 + ax^2 + b \mod 2^m \tag{1.21}$$

where the discriminant of the curve:

$$\Delta = -16(4a^3 + 27b^2) \tag{1.22}$$

is nonzero, meaning the curve is nonsingular.

Then the domain parameters of an elliptic curve over a binary field are usually a septuple [10]:

$$(m, f(x), a, b, G, n, h)^2 \tag{1.23}$$

- $m$ is an integer specifying the field $\mathbb{F}_{2^m}$ over which the curve is defined.
- $f(x)$ is an irreducible binary polynomial of degree $m$ that specifies the polynomial basis representation of $\mathbb{F}_{2^m}$.
- $a$ is an element of $\mathbb{F}_{2^m}$.
- $b$ is an element of $\mathbb{F}_{2^m}$.
- $G$ is a point on the curve $E/\mathbb{F}_{2^m}$ called the generator or sometimes the base-point.
- $n$ is the order of the generator $G$.
- $h$ is the quantity $\frac{1}{n}|E(\mathbb{F}_{2^m})|$ also called the cofactor.

## 1.3 Security requirements

As proposals to use the group of rational points of an elliptic curve over a finite field for cryptography started appearing, attacks on ECDLP over some types of curves became known. Which means there are known subsets of elliptic curves over which ECDLP or ECDHP does not have the expected complexity. This prompted the creation of standards concerning the selection

2. Sometimes written as $(m, f(x), a, b, G, r, k)$

of secure parameters. These standards and proposals have differing security requirements on the curves and often specify concrete curves for common security levels.

This section presents a summary of known security requirements [18]. It only considers security requirements inherent to the curves, not implementation specific issues, in the context of ECDLP, ECDHP and ECDH.

In this section, we mostly consider the case of curves over a prime field $\mathbb{F}_p$ as those are much more popular than binary field curves.

### 1.3.1 Generator order

The fastest general algorithm for DLP and thus currently for ECDLP is the Pollard's rho algorithm. For ECDLP, it is in $\mathcal{O}(\sqrt{n})$ and it has an average running time of $\sqrt{\pi/4}\sqrt{n} \approx 0.886\sqrt{n}$ additions, where $n$ is the order of the cyclic group in which the discrete logarithm is being computed [18].

However, for some cyclic groups (those of composite order), faster algorithms exist, namely the Pohlig-Hellman decomposition for composite order cyclic groups.

**Algorithm 2** Pohlig-Hellman decomposition [32]

▷ $g$ is the generator of the group.
▷ $n$ is the order of the generator.
▷ $X$ is the element whose discrete logarithm is being computed.
▷ Multiplicative notation is used in this algorithm.
**function** Pohlig-Hellman$(g, n, X)$
    Find the prime factorization of $n = \prod_{i=1}^{k} p_i^{e_i}$
    **for** $i = 1$ **to** $k$ **do**
        $q \leftarrow p_i, \quad e \leftarrow e_i, \quad \gamma \leftarrow 1, \quad l_{-1} \leftarrow 0$
        $\overline{g} \leftarrow g^{n/q}$
        **for** $j = 0$ **to** $e - 1$ **do**
            $\gamma \leftarrow \gamma g^{l_{j-1} q^{j-1}}$
            $\overline{X} \leftarrow (X\gamma^{-1})^{n/q^{j+1}}$
            $l_j \leftarrow \log_{\overline{g}} \overline{X}$ ▷ Using another discrete logarithm algorithm.
        $x_i \leftarrow l_0 + l_1 q + \ldots + l_{e-1} q^{e-1}$
    Use a CRT solving algorithm (such as Gauss's) to solve for $x$, $0 \leq x \leq n - 1$ such that $x \equiv x_i \mod p_i^{e_i}$ for $1 \leq i \leq k$.
    **return** $x$

Suppose there is a cyclic group $G$ of order $n = \prod_{i=1}^{k} p_i^{e_i}$. Instead of looking for the discrete logarithm in the large group of order $n$, the Pohlig-

Hellman decomposition splits the problem into $k$ sub-problems and combines them using the Chinese remainder theorem. This is fast if the order $n$ is a relatively smooth number and degrades to whatever discrete logarithm algorithm is used in it if the order is prime. Thus the Pohlig-Hellman discrete logarithm algorithm is in $\mathcal{O}(\sqrt{p_k})$ where $p_k$ is the largest prime factor of $n$. See algorithm 2.

A similar use of the Chinese remainder theorem is even more usable in the scenario of the DHP and thus ECDHP. Again, for simplicity, assuming a cyclic group $G$ of order $n = \prod_{i=1}^{k} p_i$. Suppose that the attacker can solve DLP using a brute-force search easily up to some bound $B$. Then the attacker can find all points with different orders smaller than the bound $B$ and send them to the victim as his public key $P_i$. The victim computes $[x]P_i$, with $x$ being his private key, as part of the key agreement. Even if the result $[x]P_i$ is not available to the attacker directly, they can brute-force all the possible multiples of the point $P_i$ and check which one was computed from the behavior of the higher level protocol above the Diffie-Hellman key exchange. For example if the protocol specifies to use the derived shared secret $[x]P_i$ to key AES-CTR and encrypt data, an attacker can test decrypting the data using all possible multiples of $P_i$ until the decrypted data makes sense. Each of the victim's responses then reveals the value $x \mod p_i$ if the order of the point $P_i$ is $p_i$. Combining these leaks using CRT, the attacker can compute $x \mod n_r$ with $n_r = \prod_{i=1}^{r} p_i$, $r \leq k$ so that $p_r$ is the largest prime dividing $n$ that is smaller than the bound $B$. [33, 34]

### 1.3.2 Anomalous curves

Anomalous curves are prime field curves for which the following holds:

$$|E(\mathbb{F}_p)| = p \tag{1.24}$$

This is equal to them having the trace of Frobenius equal to 1, which means they are sometimes called trace one curves. Due to Satoh; Araki [35], Semaev [36] and Smart [37] a polynomial time algorithm is known to compute the discrete logarithm on anomalous curves. This algorithm is called an additive transfer, as it allows to transfer the generally hard ECDLP to the trivial DLP in the additive group of $\mathbb{F}_p$.

### 1.3.3 Multiplicative transfers

The *embedding degree* of a curve with generator order $l$ over a finite field $\mathbb{F}_p$ is the smallest $t \in \mathbb{N}$ such that:

$$l | p^t - 1 \tag{1.25}$$

The embedding degree can also be formulated as the order of $p$ in $\mathbb{Z}_l$.

ECDLP on every curve $E/\mathbb{F}_p$ can be transferred to DLP in the multiplicative group of $\mathbb{F}^*_{p^t}$, an extension field of the original curve field $\mathbb{F}_p$. The transfer works by establishing an isomorphism between $\langle G \rangle$, the subgroup of $E/\mathbb{F}_p$ generated by $G$ and the subgroup of $l$-th roots of unity in $\mathbb{F}_{p^t}$, with $l$ being the order of the generator $G$, using the Weil pairing. Which can be computed in probabilistic polynomial time [38].

Since index-calculus methods have sub-exponential times in such a multiplicative group, this attack, known as the MOV attack[3], provides a sub-exponential algorithm for ECDLP. However, this attack is only viable for curves with small embedding degree $t$ as even a sub-exponential algorithm in $\mathbb{F}^*_{p^t}$ with large $t$ has larger complexity than a direct ECDLP solving algorithm such as Pollard's rho. [39, 40]

This transfer also works on supersingular curves over prime fields, as they have order equal to $p + 1$ which obviously divides $p^2 - 1$, which means that the order $l$ of any subgroup divides $p^2 - 1$, which means the curve has embedding degree $t \leq 2$.

### 1.3.4 Extension field curves

Curves defined over an extension field $\mathbb{F}_{p^m}$ have structure that allows using index-calculus methods to solve ECDLP on their group of rational points. These methods have running time dependent on the size of the exponent $m$ of the ground field and on the order of the characteristic $p$ of the ground field in the multiplicative group $\mathbb{F}^*_m$.

Extension fields of characteristic 2 see the most use as ground fields of elliptic curves in cryptography as arithmetic in those fields can be very fast. However with the results of Diem [41] and Gaudry [42], discrete logarithm on such curves over a binary field $\mathbb{F}_{2^m}$ is susceptible to index calculus methods which might decrease the security level of ECDLP on the curve.

Furthermore, these results show that if extension field curves are to be used, then $m$ should be a prime such that the multiplicative order of 2 modulo $m$ is large, for the curve to retain it's security [24].

3. After it's authors Menezes, Okamoto and Vanstone.

### 1.3.5 Discriminant size

Complex multiplication discriminant of an elliptic curve $E/\mathbb{F}_p$, denoted $D$, is the largest square-free negative integer such that:

$$4p = t^2 - s^2 D \tag{1.26}$$

holds, for some $s \in \mathbb{N}$, where $|E(\mathbb{F}_p)| = p+1-t$. This should not be confused with the discriminant of the elliptic curve denoted $\Delta$.

Due to Duursma et al. [43], Wiener; Zuccherato [44], and Gallant et al. [45] there are some speedups to the discrete logarithm computation for curves with small discriminant. Namely, curves with small discriminant have computable endomorphisms, which can speed up the scalar-multiplication algorithm, however, they also speed up the discrete logarithm computation. This leads to a decrease of the security level of ECDLP on the curve.

Anomalous binary curves or sometimes called Koblitz curves are a small class of binary field curves that have coefficients in $\mathbb{F}_2$. The attacks presented in this subsection also apply to them, in fact, they originally only applied to them, however, they were generalized to curves of small complex multiplication discriminant.

## 1.4 Implementation issues

Even when all the known security requirements on the domain parameters are satisfied, the inherent complexity of elliptic curve arithmetic lends itself to implementation errors and issues.

Correct and bug-free implementation of elliptic curve arithmetic in the common short Weierstrass curve form is especially hard to achieve. The addition formulas have special cases and properties which make them hard to implement securely. Furthermore, considering side-channel attacks the requirements for a secure implementation are even higher.

In this section, we mostly consider the scenario of a cryptographic library or a device performing ECDH.

### 1.4.1 Small-subgroup attacks

The small-subgroup attacks are two attacks on implementations of general Diffie-Hellman and thus also work on elliptic curve Diffie-Hellman. [46]

The small-subgroup key confinement attack is possible if an implementation does not check that untrusted points, received when performing ECDH, lie on the correct subgroup of the curve, generated by the generator. This attack also requires that the curve does not have a prime order and that it has

many small-order subgroups. This attack was introduced by Van Oorschot; Wiener [33], further analyzed by Anderson; Vaudenay [34] and summarized and tested by Valenta et al. [46]. It uses a point of small order on the curve to force the ECDH derived shared key into a small subset of keys.

The small-subgroup key recovery attack, presented by Lim; Lee [47], has the same viability requirements as the small-subgroup key confinement attack, with the addition that the victim's key cannot be ephemeral. It can be thought of as an active Pohlig-Hellman attack where the attacker sends points on small order subgroups during ECDH. It works by repeatedly using the small-subgroup key confinement attack over various small-order subgroups of the curve group and then using the Chinese Remainder Theorem to reconstruct the victim's key. It is usable even if the victim's key cannot be uniquely determined, as it narrows the key search space so that Pollard's lambda algorithm can be used.

### 1.4.2 Invalid curve attacks

An ECDH implementation using affine coordinates with the short Weierstrass equation that does not check that untrusted points lie on the group of the intended curve may be susceptible to invalid curve attacks.

The affine point Weierstrass addition formulas don't usually contain the $b$ parameter of the curve. Thus, when performing ECDH an active attacker can supply a point $P$ on a curve with a different $b$ parameter as his public key. This way, an attacker 'sneaks' in the $b$ value of a different curve into victim's scalar multiplication. The victim then computes $[x]P$ with his private key $x$ on a different curve than the intended one. A curve with the field and the $a$ parameter equal to the correct curve and a different $b$ parameter. Suppose the point $P$ (on this different curve) has order $l$ that is small enough that ECDLP on a subgroup generated by $P$ is easily computable by brute-force search. This is similar to the small-subgroup attack presented above.

The shared secret $[x]P$ computed by the victim then leaks his private key $x \mod l$. Since an attacker can easily compute all possible multiples of $P$. Depending on the higher level protocol above ECDH, the attacker can check which of the multiples of $P$ was computed by the victim based on the protocol specific behavior of the victim.

The attacker can use this to supply many points of small order, on non-prime order curves. For efficiency, the attacker should use points of smallest prime order starting from $p_1 = 2$. This minimizes the size of discrete logarithm problems the attacker needs to solve. Assume the generator of the original curve has order $n$. Then the attacker needs to perform ECDH $k$

times, so that the product of $k$ smallest primes is larger than the order $n$:

$$m = \prod_{i=1}^{k} p_i > n \tag{1.27}$$

Since then using Chinese remainder theorem, the attacker can combine the value $x \mod m$ which reveals $x$ uniquely.

This attack was first introduced in a form of a fault attack by Biehl et al. [48] and expanded on by Antipa et al. [49] who also coined the term *invalid-curve* attack.

The attack was practically demonstrated by Jager et al. [2] on several popular SSL/TLS libraries which did not check that the incoming point during ECDH lies on the same curve as the local private key.

### 1.4.3 Twist security

Even if an implementation uses different addition formulas than those based on affine short Weierstrass form, such as those used in a single-coordinate ladder like the Montgomery ladder [27] or Brier-Joye ladder [28], it still might be susceptible to a variation of invalid-curve attacks.

Both the Montgomery and Brier-Joye ladder, which is a generalization of Montgomery's ladder, are scalar multiplication algorithms which use projective coordinates. However, the addition formulas used themselves do not involve the $y$ coordinate of either point [24].

Because of this, the addition formulas for a given curve work not only on the original curve and it's isomorphisms, but also work on all nontrivial quadratic twists of the original curve [50].

A nontrivial quadratic twist of a curve has a different amount of points than the original curve and thus might have a completely different group structure. Similar to invalid-curve attacks, an attacker can only use this if the implementation does not check that untrusted points lie on the group of the intended curve. This allows an attacker to supply points on the twist during the ECDH exchange, thus making the victim compute the scalar multiple on the twist. If the twist has lesser security than the original curve, the overall security of an implementation degrades to the security of the twist

For example, the generator order of the twist might be quite smooth, which speeds up the discrete logarithm computation using the Pohlig-Hellman decomposition. This means that an implementation that does not check that untrusted points in ECDH lie on the correct curve is as secure as the weaker of the two curves, the original and the twist.

### 1.4.4 Side-channel attacks

An implementation that does not leak secret data directly can still leak secret data through side-channels, such as timing or power consumption.

Timing, the duration of a given cryptographic operation, creates a side-channel when it is somehow dependent on the value of a secret used in the operation. This dependency may be created by something obvious as by conditional branching based on a secret value or a more hidden issue such as memory caching or stem directly from the definition of an algorithm. A naive implementation of a scalar multiplication algorithm may leak bits of the scalar through the amount of time it takes to finish. A timing attack may be executed if sufficiently precise measurement of time a given cryptographic operation takes is available to the attacker, even in the context of a network connection, but network jitter makes the attack harder. However, there is little in progress of timing attacks on EC systems since an attacker can usually only leak a few bits of the secret key through timing only [23].

Power consumption of a chip/device performing a cryptographic operation can also become a side-channel, that usually leaks much more information than the timing side-channel. This is because a power trace of the execution of a cryptographic operation contains much more information than just the duration of the operation. This attack can only be executed if an attacker has physical access to the device performing the cryptographic operation. Which makes it less general than a timing side-channel, however still perfectly applicable to smart cards and other hardware security devices. Power analysis attacks can be divided into two categories, *simple power analysis* (SPA) and *differential power analysis* (DPA).

#### Simple power analysis

*Simple power analysis* works with a single power trace of a cryptographic operation, an attacker manually analyses the power trace to gain information about a secret used in it. An attacker can also build a library of slices of power traces and their meaning, then use this library to automatically match a power trace to a sequence of operations/instructions. A naive implementation of, for example, the double-and-add scalar multiplication algorithm on an elliptic curve in short Weierstrass form, leaks the order of point doubling and point addition from which the private scalar can be easily recovered. This stems from the use of non-complete addition formulas which use different equations for point doubling and point addition, as they can be easily discerned on the power trace.

In general, any algorithm that processes a secret in some bitwise fashion and conditionally branches on its values is vulnerable to SPA. This can be counteracted by using a scalar multiplication algorithm that does not leak bits of the secret to SPA, such as Montgomery's ladder or Brier-Joye ladder, or using a curve form for which complete addition formulas are available as Jacobi, Edwards or Hessian form [24].

Differential power analysis

*Differential power analysis* uses statistical methods to recover small variations of power consumption correlated with secret data over a large set of power traces. This means that DPA is usable even if the sequence of instructions does not change depending on the secret data, but a single instruction's power consumption changes based on secret data being processed.

An attacker using DPA first collects a large amount of power traces of the device performing the cryptographic operation with a fixed secret and varying public data. Based on the knowledge of the underlying algorithm he selects an intermediate variable from the algorithm that depends on a part of the fixed secret and the varying public data. Given a guess $k'$ of the part of the fixed secret and public data $m$ he can compute the value of this intermediate variable. The collected traces are then grouped based on the value of this intermediate value given the guess $k'$ and $m_i$ for the $i$-th trace. The traces within a group are then averaged, the averages of groups are subtracted and this resulting differential trace is analyzed. For an incorrect guess $k'$ the differential trace should be flat, however, for a correct guess, the trace will display spikes in regions affected by the intermediate variable [23].

There are again many countermeasures against DPA, most rely on randomizing the values of intermediate variables of the algorithm, thus not allowing the attacker to actually compute any intermediate variable of the algorithm. For scalar multiplication on an elliptic curve, this might mean varying the scalar and computing the result of the multiplication as subtraction of two points [24].

### 1.4.5 Fault attacks

With physical access to a device computing a cryptographic operation it may be possible for an attacker to change the environment of the device in such a way and time so as to inject errors into the computation. Gaining the output of an erroneous computation and a valid one sometimes allows an attacker to gain some secret information [24].

Biehl et al. [48] first introduced the invalid-curve attack as a fault attack that injected faults into the scalar multiplication to obtain a result of scalar multiplication on a different curve. Ciet; Joye [51] showed that less precise faults can be used to achieve the same goal. Similarly Fouque et al. [50] presented fault attacks on the Montgomery ladder scalar multiplication algorithm.

### 1.4.6 Cofactor computation

Curves with a cofactor not equal to one may pose challenges to some protocols not adapted to deal with them. A bug coming from improper validation of points being on the prime order subgroup of a curve in CryptoNote based cryptocurrency ByteCoin was practically exploited to gain coins then valued at around 3 million US dollars [52, 53].

# 2 Generating domain parameters

Since the security of cryptographic protocols based on the ECDLP depends on the selected curves, their group's structure and order, it is necessary to be able to create curves that satisfy all posed security requirements. Generating elliptic curve domain parameters is also useful in testing elliptic curve cryptography implementations [54]. It also forms the basis of elliptic curve primality proving algorithms.

However, efficiently generating elliptic curve domain parameters with specific properties is not an easy task. The size of the parameter space is very large for curves with practical cryptographic applications. Of those tuples of parameters only some actually form an elliptic curve (are non-singular) and only some of them have the desired properties. The process is also complicated by the fact that most desired properties involve the curve's order or group structure and the computation of those, given a random elliptic curve and an underlying field, is a complex and slow process.

Thus, two main categories of methods are used practically [16].

The *exhaustive approach*, sometimes referred to as the *random approach* is a group of algorithms that share a common idea of generating domain parameters. These algorithms first generate a random curve, using algorithm-specific methods and requirements, then use a point-counting algorithm to determine the order of the curve's group. Once the order and structure is known the curve can be checked against the requirements specific to the algorithm, to decide whether it is a suitable curve. If it is not, the process is repeated with another random curve until a suitable one is reached.

This process may be modified to form an algorithm often called *verifiably random* the goal of which is to convince the user of the curve that it was generated randomly and not chosen from a set of curves vulnerable to some secret vulnerability. It usually uses one-way functions and nothing-up-my-sleeve numbers.

The *complex multiplication method* approaches the problem of finding a suitable curve from the opposite direction as the exhaustive approach. It uses the theory complex multiplication to find a curve with given suitable order and thus given group structure. However, this method by definition does not easily satisfy one known security requirement, the size of the complex multiplication discriminant, as this method gets slower as the complex multiplication discriminant grows.

## 2.1 Exhaustive approach

The exhaustive approach is based on a loop that first (randomly) creates the parameters which uniquely determine an elliptic curve over an underlying finite field. It then uses a point-counting algorithm to compute the group order. The rest of the curve's properties are computed. Then it checks whether it passes the posed requirements. If the curve passes it is output, if not the loop continues.

The parameters $a, b$ and the field $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$ uniquely determine an elliptic curve in short Weierstrass form. However, given those parameters, computing the number of points on the curve (the order of the curve group) and finding its generators is non-trivial.

Over a field of $q$ elements, by Hasse's theorem we have that:

$$||E(\mathbb{F}_q)| - q - 1| \leq 2\sqrt{q} \tag{2.1}$$

which gives some bounds on the number of points of $E/\mathbb{F}_q$.

The first deterministic polynomial time algorithm for counting points on elliptic curves over a prime field $\mathbb{F}_p$ was Schoof's [14]. Which computes the number of points $|E(\mathbb{F}_p)|$ through computing the trace of the Frobenius endomorphism $t$ and the property that $|E(\mathbb{F}_p)| = p + 1 - t$.

The Schoof-Elkies-Atkin algorithm is an improvement to the running time of Schoof's narrowing the search space for the trace of Frobenius by better usage of properties of certain primes [14].

The case of counting points in elliptic curves over a binary field $\mathbb{F}_{2^m}$ is quite different and the aforementioned algorithms are not usable or efficient for such a case. The first practically usable algorithm was Satoh's $p$-adic [55]. Another, much simpler, algorithm is the Arithmetic-Geometric-Mean (AGM) technique, originally applied to compute elliptic integrals and later applied to counting points by Mestre [56]. However, the number-theoretic complexity of these algorithms is well above the intended scope of this thesis and thus these algorithms are not discussed more.

### 2.1.1 Schoof's algorithm

Given a curve in short Weierstrass form:

$$E/\mathbb{F}_p : \quad y^2 = x^3 + ax + b \tag{2.2}$$

The curve order can be written as $|E(\mathbb{F}_p)| = p+1-t$ with $t$ the being the trace of the Frobenius endomorphism $\phi_p$. Where the Frobenius endomorphism

$\phi_p$ is a homomorphism from the curve to itself given by $(x, y) \mapsto (x^p, y^p)$. Applying Hasse's theorem gives $|t| \leq 2\sqrt{p}$.

Since the characteristic polynomial of the Frobenius endomorphism is $T^2 - tT + p$, we have:

$$\forall P \in E(\mathbb{F}_p): \quad \phi_p^2(P) - [t]\phi_p(P) + [p](P) = \mathcal{O} \tag{2.3}$$

However, this equation is not directly beneficial, since solving it for $t$ at a point $P$ would require computing the discrete logarithm of $\phi_p^2(P) + [p](P)$ with a base of $\phi_p(P)$.

Schoof's algorithm computes this trace $t$ modulo many small primes $l_1, \ldots, l_r$ such that the product $\prod_{i=1}^{r} l_i > 4\sqrt{p}$, the trace is then recovered through the Chinese Remainder Theorem.

By looking at the $E[l]$ group of $l$-torsion points of $E/\mathbb{F}_p$ we see that:

$$\forall P \in E[l]: \quad \phi_p^2(P) - [t_l]\phi_p(P) + [p_l](P) = \mathcal{O} \tag{2.4}$$

for some $t_l \equiv t \mod l$ and $p_l \equiv p \mod l$. However, directly computing $t_l$ from solving the discrete logarithm in this equation on $E/\mathbb{F}_p$ will not be possible for a large percentage of small primes $l$, since if $l$ does not divide $|E(\mathbb{F}_p)|$ then $E[l] \cap E(\mathbb{F}_p)$ is trivial.

By defining $\Psi_l(X) \in \mathbb{F}_p[X]$ as a polynomial with roots equal to the $x$ coordinates of points in $E[l]\backslash\{\mathcal{O}\}$, called the division polynomial, we see that this implies:

$$(X^{p^2}, Y^{p^2}) + [p_l](X, Y) \equiv [t_l](X^p, Y^p) \tag{2.5}$$

modulo polynomials $\Psi_l(X)$ and $Y^2 - X^3 - aX - b$, where the second polynomial makes sure $(X, Y)$ satisfies the curve equation. As $\Psi_l(X)$ is computable via a recursive formula, this gives us the number of points on the elliptic curve $E/\mathbb{F}_p$ after determining the trace $t$ from the division polynomials of the $r$ smallest primes using the Chinese Remainder Theorem [14] [24].

### 2.1.2 SEA algorithm

The Schoof-Elkies-Atkin algorithm is an improvement upon the original Schoof's algorithm. The general idea is the same, determine the number of points on the curve by computing the trace of the Frobenius endomorphism modulo many small primes and combine it with the Chinese Remainder Theorem.

The algorithm uses properties of certain primes to eliminate many values of $t_l \equiv t \mod l$. Given a small prime $l$ and an elliptic curve over $\mathbb{F}_q$, the prime $l$ is either an Atkin prime or an Elkies prime, where both types of primes

restrict the possible set of values $t_l$ in different ways. The type of prime $l$ depends on whether the complex multiplication discriminant $D = t^2 - 4q$ is a square in $\mathbb{F}_l$, in case it is, it is an Elkies prime, otherwise it is an Atkin prime. Determining this directly is not possible as it would involve computing the trace of the Frobenius endomorphism which is the goal of the whole algorithm. However, Atkin noted that the distinction can be made from certain modular polynomials.

For $l$ an Elkies prime, there exists a subgroup $C_l$ of $E[l]$ of order $l$ that is closed under the Frobenius endomorphism, so that $\forall P \in C_l : \phi_p(P) \in C_l$. Which means that $\phi_p$ acts as a multiplication on $C_l$, thus the polynomial $g_l = \prod_{\pm P \in C_l \setminus \{\mathcal{O}\}}(X - x_P)$ can be used in place of the division polynomial in Schoof's algorithm. This is a speedup as the degree o $g_l$ is significantly smaller than the degree of the $l$-th division polynomial and it is also efficiently computable.

For $l$ an Atkin prime, there are only $\varphi(r)$ possible values of $t_l \equiv t \mod l$, with $r$ being the order of the Frobenius endomorphism in $\mathrm{PGL}_2(\mathbb{F}_l)$, where $\varphi(r) < l$ which provides a speedup [24].

The SEA algorithm is implemented in the PARI/GP algebra system [57].

## 2.2 'Verifiably' random algorithms

A common situation in practical cryptography is one, where the user of some set of domain parameters does not, or rather would like to not need to, trust the creator of said parameter set. The rationale is that this creator of domain parameters might backdoor them by creating them in a way, where they have a vulnerability known only to the creator, and thus exploitable by the creator. It is plausible that such a vulnerability could exist in elliptic curve domain parameters as many classes of curves were discovered to be insecure as presented in an earlier section.

Thus, most standards establishing a set of named domain parameters use what could be called a *verifiably random* algorithm for generating these domain parameters, it consists of a generation algorithm and a verification one. Its goal is to provide assurance to the user of the domain parameters, that they were created randomly and are not part of some vulnerable class of curves. It does so via restricting the size of the search space for the parameters into one where it should be practically impossible for the creator to find a set of insecure parameters with some hidden properties.

The generation algorithm is a variation of the random approach algorithm. Instead of selecting parameters to try completely randomly, it starts with a seed value, usually a nothing-up-my-sleeve number or a random one,

and uses a one-way hash function to create candidate domain parameter values and update the seed.

This restricts the possible parameters to ones that are outputs of a hash function iterated over a nothing-up-my-sleeve number. Furthermore, the algorithm stops after the first set of parameters that define an elliptic curve that passes all the algorithm specific requirements. The seed works to restrict the search space, while the hash function works to iterate and move through the search space in a pseudo-random but deterministic way.

The verification algorithm receives a generated curve and a seed the curve was supposedly generated from. It iterates the seed in the same way the generation algorithm would, stopping when the generated curve parameters are reached and thus confirming the curve was generated using the claimed method along with the claimed seed.

There are a few verifiably random elliptic curve domain parameter generation algorithms, but they all share the common structure specified above. Historically this method of generating elliptic curve domain parameters was introduced in *ANSI X9.62* [58]. With changes it then appeared in *NIST FIPS 186-2* [59], *IEEE P1363* [5], *SECG2 v1* [60], *SECG2 v2* [10], *Brainpool* [6] and *Brainpool RFC 5639* [7].

### 2.2.1 ANSI X9.62 algorithm

The first 'verifiably' random algorithm for generating elliptic curve domain parameters was provided in *American National Standard X9.62-1998, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)* [58]. It uses a seed of at least 160 bits and the SHA-1 function for parameter iteration. It has two versions, one for generating elliptic curves over prime fields and one for elliptic curves over binary fields. We show the algorithms in the appendix, as algorithms 3 and 4.

### 2.2.2 Brainpool algorithm

The Brainpool algorithm was first published in *ECC Brainpool Standard Curves and Curve Generation* [6]. It presented 14 new curves, along with a verifiably random algorithm to generate them. Two of these curves see practical use in German passports and some are supported in the TLS protocol. The curves and the generation method was standardized in the form of an *IETF RFC 5639* [7].

However, as was later revealed by Bernstein et al. [61] some of the curves presented as generated by a verifiably random algorithm could not be gen-

erated by it. In fact, the two algorithms, the one presented in the original publication and the one in the RFC, slightly differ in the order of generated parameters. This discrepancy was not publicly known for a few years which questions the real utility and security of verifiably random algorithms for generating domain parameters when their users fail to verify them.

### 2.2.3 Manipulatability

The level of assurance in the randomness and security of domain parameters generated using 'verifiably' random algorithms was questioned by Bernstein et al. [61]. Even under what might be considered a strict acceptability model of what a user of a standard might accept as a valid verifiably random generation method, there is still significant flexibility left to the standard creator. The flexibility is in the choices of the generation method, its parameters and inputs.

When viewing the generation method and its inputs as fixed, the method indeed restricts the subset of possible domain parameters generated by it to a somewhat random subset of all domain parameters. However, when the generation method itself is allowed to vary, as it is when the standard creator publishes it them self, the freedom to choose a subset of domain parameters that the method can generate is significant. Bernstein et al. [61] gives that, assuming reasonable acceptance criteria of the user of a standard, a standard creator could possibly target a secret vulnerability with a probability of occurring in a random elliptic curve of $2^{-20}$.

The attack by which the standard creator finds a curve with a backdoor and a generation method that generates said curve which the user will accept is also presented in Bernstein et al. [61]. Its basis is simple, the standard creator enumerates all the acceptable generation methods, varying things like the hash functions used in them, seed update functions and other similar parameters. This is done until a generation method that generates a curve with a secret backdoor is found, which is them published, with arguments for its security against known attacks and how it was generated in a 'verifiably' random way.

## 2.3 Complex multiplication

The method of complex multiplication works in a significantly different way from the exhaustive approach, it uses elements from number theory and complex analysis to constructively obtain an elliptic curve with a given number of points. The methods used in the process are much more general than just

constructing specific elliptic curves. However, we focus only on the concrete process of constructing elliptic curves [24].

Suppose we want to construct an elliptic curve $E$ over $\mathbb{F}_p$ with the given number of $\mathbb{F}_p$-rational points $N$.

$$E/\mathbb{F}_p : \quad y^2 \equiv x^3 + ax + b \tag{2.6}$$

The quadratic twist of this curve is given by a quadratic non-residue $d \in \mathbb{F}_p$.

$$\widetilde{E}/\mathbb{F}_p : \quad dy^2 \equiv x^3 + ax + b \tag{2.7}$$

Then, introduce the $p$-th power Frobenius endomorphism:

$$\begin{aligned} \phi_p : E(\overline{\mathbb{F}}_p) &\to E(\overline{\mathbb{F}}_p) \\ (x, y) &\mapsto (x^p, y^p) \end{aligned} \tag{2.8}$$

The Frobenius endomorphism has the interesting property that it is the identity when considered over $E(\mathbb{F}_p)$. The order of the curve can then be stated as follows, with $t$ being the trace of the Frobenius endomorphism:

$$N = |E(\mathbb{F}_p)| = p + 1 - t \tag{2.9}$$

The characteristic polynomial of the $p$-th power Frobenius endomorphism is the following:

$$\phi^2 - t\phi + p \tag{2.10}$$

The discriminant of this polynomial is below, this is also defined as the CM discriminant of the curve $E/\mathbb{F}_p$.

$$D = t^2 - 4p \tag{2.11}$$

The fundamental discriminant $d$ is the square-free part of the CM discriminant:

$$D = s^2 d \tag{2.12}$$

Let $H_d(x) \in \mathbb{Z}[x]$, the Hilbert class polynomial, be the minimal polynomial of $j\left(\frac{d+\sqrt{d}}{2}\right)$, the modular $j$-function. Then, let $j_p \in \mathbb{Z}$ be a root of said polynomial modulo $p$, such that $H_d(j_p) \equiv 0 \mod p$. Then there is a elliptic curve over $\mathbb{F}_p$ with $j_p$ as its $j$-invariant. Furthermore, the amount of $\mathbb{F}_p$-rational points on this curve is one of $p + 1 + t$ or $p + 1 - t$.

We construct a curve with one of those orders from its $j$-invariant, $j_p$:

$$\begin{aligned} k_p &= \frac{j_p}{1728 - j_p} & a_p &= 3k_p \\ & & b_p &= 2k_p \\ E_{cm}/\mathbb{F}_p : \quad y^2 &\equiv x^3 + a_p x + b_p \end{aligned} \tag{2.13}$$

Now, either $E_{cm}/\mathbb{F}_p$ or its nontrivial quadratic twist $\widetilde{E}_{cm}/\mathbb{F}_p$ has the number of $\mathbb{F}_p$-rational points equal to $N$.

Constructing the Hilbert class polynomial is the most time-consuming part of the computation, however, this polynomial only depends on the value of the fundamental discriminant, so it can be precomputed. There have been significant advances in computing class polynomials for large fundamental discriminants. A simple algorithm for doing so is also given in the *IEEE P1363* [5] standard.

For more information on the theory of complex multiplication of elliptic curves we refer to Silverman [62], Bröker; Stevenhagen [63], and Cohen et al. [24].

## 2.4 ecgen

Ecgen [64] is a command-line tool for generating elliptic curve domain parameters.

A need for such a tool arose when the ECTester project needed custom elliptic curve domain parameters for implementation testing and no such general tool existed. Its development started in the GP language, which is an interpreted language that is run on top of the PARI/GP math and algebra library. However, the language features of GP did not suffice for the development of more features of the tool and it was then rewritten in ANSI C 99, while still using the very powerful PARI/GP library.

### 2.4.1 Features

Many methods of generating domain parameters are implemented because different methods are better suited based on the requested properties of the generated domain parameters. Most are based on the exhaustive approach.

#### Exhaustive approach

The main method of generating domain parameters in ecgen is the exhaustive approach, as described in section 2.1. Other methods usually use this algorithm as their basis. In ecgen, this method generates the parameters in fixed order:

`[seed]`, `field`, `a`, `b`, `curve`, `order`, `generators`, `points`

Multiple methods for creating each individual parameter are available and are combined at runtime based on the arguments. Creation of each

parameter might fail, which rewinds the process to start from the seed step. Examples of provided creation methods are:

- random `field` or `field` from input
- random equation parameters `(a, b)` or Koblitz equation parameters
- points of prime order only, composite order only or all possible orders

Multiple checks for each individual parameter are also available, such as:

- check for a prime order of a curve
- check for a specific cofactor
- check for the embedding degree
- check for the anomalous condition
- check for the size of the complex multiplication discriminant
- check for the group structure and amount of generators

This modular structure of generators and checks allows the exhaustive method to be very flexible and it is used throughout other methods as a subpart.

Furthermore, each generation attempt can be timed and timed-out by a watchdog alarm if it does not complete after a requested amount of time, this is useful to skip a set of parameter sometimes if for example the order computation algorithm gets stuck for an unusually long amount of time.

### 'Verifiably' random algorithms

Three verifiably random algorithms are implemented, from *ANSI X9.62* [58], *Brainpool* [6] and *Brainpool RFC 5639* [7]. All of them are realized through a customization of the aforementioned exhaustive approach.

### Invalid curves

The basis of the algorithm to generate invalid curves for a given curve is from Jager et al. [2]. This method computes invalid curves with points of prime orders up to a given bound or from a given interval if requested. The implemented method also differs from the one by Jager et al. [2] in that it is multi-threaded and seems to achieve much faster computation of all required points to a given bound than the provided timings in the original article.

### Anomalous curves

Anomalous curves are generated using the method proposed by Miyaji [15], which was intended to generate safe curves for use in cryptography, however as was shown before, trace one curves are not safe. This implemented method in fact uses complex multiplication, however, the actual complex multiplication related steps are precomputed and present in the original article.

### Complex multiplication

The implemented complex multiplication method is markedly different from the exhaustive approach and thus it does not use it. Two different ways of computing the class polynomial from the discriminant are implemented, one uses the built-in function that does so from the PARI/GP library, other implements the class polynomial computation algorithm from *IEEE P1363* [5].

The method for finding a small discriminant and a prime order field given some intended elliptic curve order is based on an algorithm by Bröker; Stevenhagen [65] with corrections, see appendix for algorithm 5.

## 2.4.2 Results

We both tested the tool and measured timings as well as used it in practice extensively. Here we present some interesting timings obtained. For actual example outputs of ecgen runs, see the appendix.

The following data was obtained on Intel® Core™i7-8550U CPU with 4 physical cores, with base clock at 1.80GHz, while using PARI/GP version `2.9.4`.

### Invalid curve generation

We ran and timed invalid curve generation for few standard curves. Each run of the generation process generated enough invalid curves and points on them that could be used to completely leak the static private key of a vulnerable ECDH implementation. As the generation process has a multi-threaded variant implemented, we also ran that using 2, 4 and 8 threads. The results of this benchmark can be seen in table 2.1.

| Curve | 1 thread | 2 threads | 4 threads | 8 threads |
|---|---|---|---|---|
| `secp128r1` | 141.68s | 106.41s | 72.23s | 64.26s |
| `sect163r1`* | 23.13s | 135.69s | - | - |

Table 2.1: Average run times to generate invalid curves for a given curve, 10 runs

Interesting behavior can be seen while generating invalid curves for a binary field curve, where using multiple threads harms the performance of the process. This might be due to a different point counting algorithm used by PARI/GP for binary field curves.

### Prime order curve generation

Exhaustive search using random parameters for a prime order curve over a prime field is an example of a generation process with very variable run time. To benchmark ecgen we ran this process for bit lengths of 128 and 192 bits while measuring the run time of the process. The histogram of 100 runs for each bit length is visible in figs. 2.1 and 2.2 respectively.

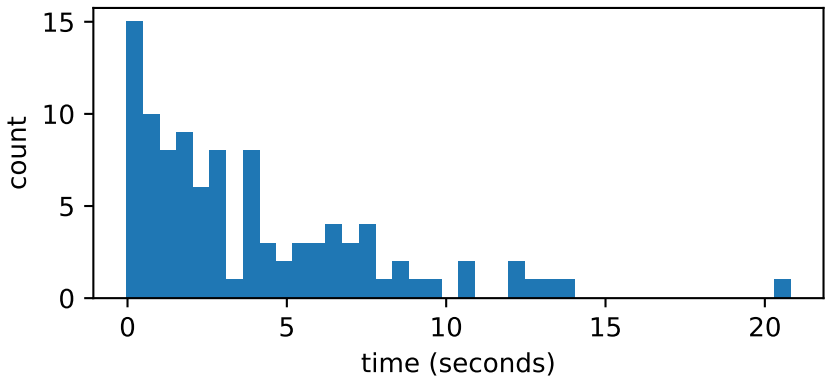


Figure 2.1: 128 bits, 100 samples
`ecgen -fp -prime -random 128`

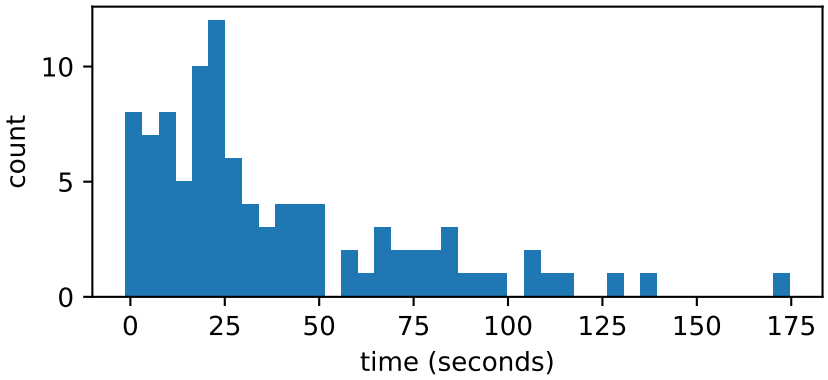


Figure 2.2: 192 bits, 100 samples
`ecgen -fp -prime -random 192`

This benchmark clearly demonstrates the need for the implemented time-out feature, where a generation of a random curve with given parameters can be timed-out after some time and new random parameters tried. For larger bit lengths some computations of a curve order using the SEA algorithm might take very long, while discarding the parameters and continuing with new random ones might arrive at a prime order curve, or any other property being sought, faster.

# 3 Testing black-box implementations

Elliptic curve cryptography algorithms present an implementation challenge. They require implementation of more complex algebraic objects than for example multiplicative group based cryptography algorithms. These additional layers of complexity, on top of finite-field arithmetic and such, provide ample places for error. Also, textbook definitions of some basic elliptic curve operations such as point addition or scalar-multiplication often have many edge cases. Thus, testing with just test-vectors for a given elliptic curve based primitive is often not enough. A comprehensive test suite based on real attacks and scenarios is necessary.

In this chapter, we present ECTester, a tool for testing elliptic curve implementations. Its development was started by RNDr. Petr Švenda Ph.D. and continued by me as a part of this thesis. It targets both hardware and software by working with the JavaCard smart card platform and with native or Java cryptography software libraries.

The tool consists of three parts. A JavaCard applet for testing smart cards, a corresponding reader utility for the host machine and a utility for testing software libraries.

## 3.1 JavaCard testing

JavaCard platform is the most popular platform for building systems based on programmable smart cards.

JavaCards provide a Java-like environment to run applications, called applets, on programmable JavaCard-based cards. However, the environment is only Java-like in that the feature set of Java it supports is not complete and is specific to a concrete JavaCard version. The cards communicate using the ISO/IEC 7816-4 APDU format [66] with a reader application running on a host machine.

There is also a standard JavaCard cryptographic API, which provides access to many cryptographic primitives, including ECDH and ECDSA, from JavaCard specification version 2.2. The primitives itself are implemented by the card manufacturer, usually using a specific cryptographic co-processor. This means their implementation is largely a black-box, only exposing the required JavaCard API. Because of this, a card user has to trust the manufacturer's implementation or possibly some third-party certification of the card, to use the algorithms present on the card.

Elliptic curves are a good choice for the smart card environment, because of their relatively short key sizes and interesting performance prospects.

Smart cards are often used in very high-risk scenarios, such as banking applications, passports and other electronic identification documents. Together with the black-box nature of their cryptographic implementations this makes smart cards and concretely JavaCards a prime target for comprehensive testing of elliptic curve implementations.

### 3.1.1 Applet

The applet is a part of ECTester that gets uploaded and ran on the card itself. It is a JavaCard applet aiming for compatibility with JavaCard API version at least 2.2.2.

Due to the constrained feature set of Java supported on JavaCards as well as limited performance the applet itself doesn't contain testing logic. Writing testing evaluation functionality for a JavaCard environment would be cumbersome and slow. Also, no timing measures could be taken as the JavaCard API doesn't specify any time sources.

Because of this, the applet only provides a stateful APDU-based command-response interface with low-level commands. This allows to construct complex testing logic on the host machine side while keeping the applet relatively simple. The applet works on four objects in total, two `KeyPairs`, one `KeyAgreement` object and one `Signature` object. All commands manipulate these objects in some way. It supports a total of thirteen commands listed below:

ALLOCATE_KA

Allocates a new `javacard.security.KeyAgreement` object into one or both of the `KeyAgreement` slots. The reader can specify the type of the requested `KeyAgreement` as the JavaCard platform provides a few elliptic curve based ones.

ALLOCATE_SIG

Allocates a new `javacard.security.Signature` object into one or both of the `Signature` slots. A specific `Signature` type can be requested by the reader.

ALLOCATE

Allocates one or both new `javacard.security.KeyPair` objects into the `KeyPair` slots. Both a bit-size of the requested `KeyPair` and type of the base field should be specified, either prime field or binary field.

CLEAR

Clears a particular `KeyPair` with the `clearKey()` call.

SET

Sets values sent from the reader into one or both `KeyPairs` and its parts. This can be a value of a private key, public key or some of the domain parameters, this command can also set multiple parameters at once and on both `KeyPairs` and key parts.

TRANSFORM

Transforms values of one or both `KeyPairs` and their key parts. The applet can perform several transforms, such as filling a value with random data, incrementing it, as well as point related transformations such as point compression or setting of infinity.

GENERATE

Generates one or both of the `KeyPairs` using the `genKeyPair()` call.

EXPORT

Exports values from one or both `KeyPairs` and their key parts. This can export both domain parameters as well as public and private key values.

ECDH

Performs ECDH using the allocated `KeyAgreement` and one or both `KeyPairs`. The reader specifies which public and private keys will be used, it also receives the result of the key agreement operation.

ECDH_DIRECT

Performs ECDH using the allocated `KeyAgreement`, one of the `KeyPair`'s private key and data sent by the reader as the public key. This is required to

properly test the `KeyAgreement` API, as it only requires a byte array as the public key value. Thus, passing the public key value through an `ECPublicKey` instance is not necessary. The result of the `KeyAgreement` is sent back to the reader.

ECDSA

Performs ECDSA signature and verifies it using the allocated `Signature` and one of the `KeyPairs`. The signature is performed over data supplied by the reader and sent back.

ECDSA_SIGN

Only performs the ECDSA signature using the allocated `Signature` and one of the `KeyPairs`, sends the signature back to the reader.

ECDSA_VERIFY

Verifies data and signature sent by the reader, using the allocated `Signature` and one of the `KeyPairs`, sends verification result back to the reader.

### 3.1.2 Reader tool

To communicate with an applet installed on a JavaCard we built a Java command line utility. This utility communicates with the applet through a reader plugged into the host machine, using the PC/SC support in the Java standard library.

The reader tool uses the commands provided by the applet to provide utility ECC functions on the card, such as generating elliptic curve keypairs and exporting them, exporting the default domain parameters, performing elliptic curve based key agreement such as ECDH or a signature scheme as ECDSA.

## 3.2 Software library testing

Testing many software libraries which support some set of elliptic curve cryptography is harder than testing many JavaCards, as the API between them differs heavily. To comprehensively test them, one would need to implement the complex test suites required for each library to test.

We overcome the heterogeneous nature of library APIs by creating a Java Cryptography Provider as well as implementing an elliptic curve related

subset of the Java Cryptography Architecture for each library and then build tests using the created Java Cryptography Provider. This obviously also covers all libraries that implement a Java Cryptography Provider, such as BouncyCastle or SunEC.

For libraries implemented in C/C++ or those that have a FFI with C we implement a small shim using the Java Native Interface. This shim uses the tested library to implement the minimal set of functionality required to test the implementation and create a Java Cryptography Provider. The shim is compiled into a shared object and packed into ECTester's JAR, which then tries loading it dynamically at runtime.

## 3.3 Test suites

ECTester provides few comprehensive test suites, each of which targets testing of different functionality or potential vulnerability.

### 3.3.1 Default

The default test suite tests basic support and performance of ECDH and ECDSA algorithms. Detects whether the device supports these algorithms, for what bit sizes and for what base fields.

### 3.3.2 Test vectors

Test vectors for ECDH and ECDSA were collected for standard named curves where available from the standard author or other related party. For JavaCard testing this only leaves ECDH and ECDHC, as JavaCard does not implement sufficient API to supply the required ECDSA random value $r$ during signing, it only implements classic ECDSA, thus making deterministic testing of ECDSA with test vectors impossible.

This provides test vectors from NIST for ECDHC [67] and SECG [68] and Brainpool [69] [8] for ECDH. These vectors are contained in the reader tool and sent to the applet during testing, where ECDH or ECDHC is performed, result is then sent back and checked against the saved test-vector.

### 3.3.3 Invalid curves

A test of implementation response to points on invalid curves requires generating these curves for a given used curve, this was done offline using the resources of Czech National Grid Infrastructure - MetaCentrum. The test

suite contains generated invalid curves for most popularly used standard named curves from SECG, NIST and Brainpool.

The test itself then proceeds exactly like an attack on a vulnerable implementation would. The reader requests the applet generates a keypair on some standard curve, then the reader starts sending ECDH commands where the applet uses the generated keypair's private key and the point on the invalid curve as the public key.

The ECDH operation should obviously fail, and not produce any result to the reader, since the provided public key point is not on the correct curve, which the keypair was generated over. If the operation does not fail, the implementation might be leaking bits of the private key. The test suite contains enough points on invalid curves to leak a whole private key during testing of a given curve.

This test suite is mainly intended for Weierstrass form based scalar multiplication algorithms. As for Montgomery or Brier-Joye addition formulas, which do not use the $y$ coordinate of points, the following twist test suite is analogous.

### 3.3.4 Curve twists

The $x$ coordinate only addition formulas, such as Montgomery or Brier-Joye ladder, and corresponding scalar-multiplication algorithms have attractive properties for implementation on smart cards. They are also immune to invalid curve attacks in some sense. By discarding the $y$ coordinate value the invalid $b$ value is effectively also discarded.

However, these algorithms will compute on the twist of the original curve if the point supplied has $x$ coordinate on the twist. For each $z$ in $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$ there is a pair of points on the original curve or on the twist with the $x$ coordinate equal to $z$.

This test suite uses generated points on twists on standard named curves. It requests the applet to generate a keypair on a standard curve then proceeds to send it points on its twist for use in ECDH. The operation should fail as the points are on the, possibly insecure, curve twist. A success means the card leaks bits of the private key if a twist insecure curve is used.

There are several twist insecure named curves, as during their standardization the choice was made to require point validation during ECDH which means a twist secure curve is not necessary for safe ECDH.

### 3.3.5 Composite curves

It is also of interest to test the behavior of a card on curves that are insecure, even if only to check that some unexpected errors or behavior does not arise. Composite order curves are insecure if their order is sufficiently smooth.

During this test suite a composite order curve is set, two keypairs are generated on it and ECDH is performed between them. The expected result of this test suite is hard to qualify, any sensible error or success of the operation means a success of the test.

### 3.3.6 Cofactor curves

Some standard named curves have the cofactor value not equal to 1. This means that while computations and key generation should happen on the prime order subgroup generated by the generator, there are other subgroups on the curve. This leads to the question of the behavior of implementations of ECDH when a point on such a cofactor subgroup is supplied.

The cofactor test suite contains the few standard named curves with a cofactor not equal to 1 and other generated curves with increasing cofactor values and subgroup count. It also contains generated points on these curves, which are not on the subgroup generated by the generator.

The test suite proceeds by setting these curves, generating keypairs on them and supplying the mentioned points on different subgroups for ECDH. This should fail and not provide a result to the reader, as success again means the card leaks some bits of the private key when a curve with cofactor not equal to 1 is used.

This leakage is however very small for standard named curves as the usual cofactor is only 4 or 8, but could be significant when used with some custom curve.

### 3.3.7 Wrong curves

The wrong curve test suite takes an approach similar to fuzzing, it contains domain parameters that do not specify correct elliptic curves. It tests behavior of the card when setting these wrong domain parameters by trying to set them, generate keypairs on them and even perform ECDH on them, if the keypairs were successfully generated.

This test suite does not have direct security implications but serves to check the card handles these wrong parameters gracefully.

## 3.4 Results

To share and provide test results obtained from ECTester we developed a website. It contains a support and performance summary for cards that were available to us to test. This summary is also reproduced in table 3.1.

An implementation guide for working with elliptic curve cryptography on the JavaCard platform is also included in the site as the platform and its implementation on real cards has many potential pitfalls for developers.

The site is built using the Jekyll static site builder. Browseable and interactive test run reports are generated from ECTester test run outputs in the YAML format. A part of such a test run report of a real card is visible in fig. 3.1.

| | **192b**$^*$ | | **256b** | | **384b** | | **521b** | |
|---|---|---|---|---|---|---|---|---|
| **Card** | **ECDH**$^\dagger$ | **ECDSA**$^\ddagger$ | **ECDH** | **ECDSA** | **ECDH** | **ECDSA** | **ECDH** | **ECDSA** |
| Athena IDProtect | 1,2,3,4<br>148ms | a,b,c<br>265ms | 1,2,3,4<br>206ms | a,b,c<br>365ms | 1,2,3,4<br>448ms | a,b,c<br>737ms | 1,2,3,4<br>937ms | a,b,c<br>1490ms |
| GD SmartCafe 6.0 | 1,2,3,4<br>128ms | a,b,c,d,e<br>270ms | 1,2,4,3<br>190ms | a,b,c,d,e<br>355ms | | | | |
| Infineon CJTOP | 1,2,3,4<br>137ms(1)<br>, 200ms | a,b,c,d,e<br>210ms(a)<br>280ms(e) | 1,2,3,4<br>166ms(1)<br>, 266ms | a,b,c,d,e<br>263ms(a)<br>329ms(e) | 1,2,3,4<br>241ms(1)<br>, 410ms | a,b,c,d,e<br>397ms(a)<br>464ms(e) | | |
| NXP JCOP31 v2.4.1 | 1,2<br>122ms | a 257ms | 1,2<br>189ms | a 392ms | | | | |
| NXP JCOP CJ2A081 | 1,2 75ms | a 151ms | 1,2<br>123ms | a 241ms | | | | |
| NXP JCOP v2.4.2 R3 | 1,2 99ms | a,b,c<br>180ms | 1,2<br>150ms | a,b,c<br>275ms | | | | |

$*$ Ground field size.

$\dagger$
1. ALG_EC_SVDP_DH
2. ALG_EC_SVDP_DHC
3. ALG_EC_SVDP_DH_PLAIN
4. ALG_EC_SVDP_DHC_PLAIN
5. ALG_EC_PACE_GM
6. ALG_EC_SVDP_DH_PLAIN_XY

$\ddagger$
a. ALG_ECDSA_SHA
b. ALG_ECDSA_SHA_224
c. ALG_ECDSA_SHA_256
d. ALG_ECDSA_SHA_384
e. ALG_ECDSA_SHA_512

Table 3.1: Results for $\mathbb{F}_p$ curves on tested cards. Timings are average over 100 operations. ECDSA timings include duration of signature and verification.

| | Test result | Description | Result |
|---|---|---|---|
| ⊟ | ✔ | Tests of 192b ALG_EC_FP support: Some. | SUCCESS |
| 2 | ✔ | Allocate both keypairs 192b ALG_EC_FP | SUCCESS |
| 2 | ✔ | Generate both keypairs | SUCCESS |
| 2 | ✔ | Allocate both keypairs 192b ALG_EC_FP | SUCCESS |
| 2 | ✔ | Set custom curve parameters on both keypairs | SUCCESS |
| 2 | ✔ | Generate both keypairs | SUCCESS |
| ⊟ | ✔ | KeyAgreement tests. | SUCCESS |
| ⊟ | ✔ | Test of the ALG_EC_SVDP_DH KeyAgreement. | SUCCESS |
| 4 | ✔ | Allocate KeyAgreement(ALG_EC_SVDP_DH) object | SUCCESS |
| 4 | ✔ | ALG_EC_SVDP_DH of local pubkey and remote privkey(unchanged point) | SUCCESS |
| 4 | ✖ | ALG_EC_SVDP_DH of local pubkey and remote privkey(COMPRESSED point) | FAILURE |
| 4 | ✔ | Mean =137477828 ns, Median =137503097 ns, Mode =135560799 ns | SUCCESS |
| ⊞ | ✔ | Test of the ALG_EC_SVDP_DHC KeyAgreement. | SUCCESS |
| ⊞ | ✔ | Test of the ALG_EC_SVDP_DH_PLAIN KeyAgreement. | SUCCESS |
| ⊞ | ✔ | Test of the ALG_EC_SVDP_DHC_PLAIN KeyAgreement. | SUCCESS |
| 3 | ✖ | Allocate KeyAgreement(ALG_EC_PACE_GM) object | FAILURE |
| 3 | ✖ | Allocate KeyAgreement(ALG_EC_SVDP_DH_PLAIN_XY) object | FAILURE |
| ⊞ | ✔ | Signature tests. | SUCCESS |

**Legend**

- ☰: Displays either a toggle icon for a test that is composed of several tests, or a depth of a simple test.
- **Test result:** The high-level result of a test of a group of tests.
- **Description:** A short description of what the test or a group of tests did.
- **Result:** The concrete result of a test or a group of tests. Can be one of:
  - `SUCCESS` - ✔ - The test was expected to pass and it did.
  - `FAILURE` - ✖ - The test was expected to pass but it failed.
  - `UXSUCCESS` - ✖ - The test was expected to fail but it succeeded.
  - `XFAILURE` - ✔ - The test was expected to fail and it did.
  - `ERROR` - ⚠ - There was an *unexpected* error while running the test.

Figure 3.1: A partial example result of a default test suite ran on a card, showing support and performance of tested algorithms, available on the ECTester website.

# Conclusions and future work

This work explored the security of elliptic curves as used in cryptography, their domain parameters and implementations.

Known security requirements on elliptic curve domain parameters were surveyed and summarized. They are quite complex and evolved over the years from when elliptic curves were first considered for uses in public key cryptography. The need for standards concerning the generation of elliptic curve parameters is clear, as it can not be expected of a general user of elliptic curve cryptography to collect, understand and abide by all known security requirements.

Elliptic curve domain parameter generation algorithms were analyzed. Three main classes of algorithms were described. The exhaustive approach is the simplest one, being pretty obvious in structure. However, counting rational points on elliptic curves over finite fields is non-trivial and clearly the most complicated part of the exhaustive approach. Verifiably random algorithms are trying to give assurance, but only give handwaving arguments for doing so and fail to protect against anything but a simple attacker, a motivated standard creator might still backdoor the generated domain parameters. Finally, complex multiplication allows to construct curves with given orders, which is quite useful, however it is very complex.

Potential implementation pitfalls in elliptic curve cryptography were assessed along with their security implications. There is quite a lot of them, most stemming from the higher complexity of elliptic curve cryptography, where some public key cryptography is built on finite fields and multiplicative groups, elliptic curve cryptography builds another algebraic structure on top of those and uses that.

The ECTester tool was developed from the prototype level to a versatile, practically usable, open-source tool for black-box elliptic curve cryptography testing. In the process it was completely rewritten to a new architecture, many more test suites were added along with new output formats and options. These new formats were used to build a site for browsing and analyzing the test results. The tool was continuously used to test available cards.

To accommodate the need for domain parameters and data with specific properties by ECTester, the ecgen tool was built. Algorithms from all three classes of generation methods were implemented. Most of the data needed by ECTester was then generated on the MetaCentrum grid infrastructure.

There is still significant work to be done, both on projects presented in this thesis and on new paths stemming from them. There are interesting

ideas for test suites that are still not implemented in ECTester. Furthermore, automatic test analysis and inferring of conclusions about a test run from its results is a work in progress. It also might be possible to detect what kind of curve model, addition formulas and scalar multiplication algorithm an implementation uses based on its behavior with various domain parameters and specifically crafted data. Designing and implementing this in ECTester would significantly extend its utility. Finally, testing more cards and cryptographic libraries might uncover unknown vulnerabilities in them.

# Appendices

## Selected algorithms

**Algorithm 3** ANSI X9.62 prime field algorithm [4]

**function** ANSI_X962_Prime($p$)
  $t \leftarrow \lfloor \log_2 p \rfloor$;  $s \leftarrow \lfloor (t-1)/160 \rfloor$;  $h \leftarrow t - 160s$
  $SEED \leftarrow$ random bit string of at least 160 bits
  $g \leftarrow |SEED|$
  $H \leftarrow$ SHA-1($SEED$)
  $c_0 \leftarrow$ bit string of $h$ rightmost bits of $H$
  $W_0 \leftarrow c_0$ with leftmost bit set to 0
  **for** $i = 1$ **to** $s$ **do**
    $W_i \leftarrow$ SHA-1$((SEED + i) \mod 2^g)$
  $W \leftarrow W_0 \| W_1 \| \ldots \| W_s$
  $r \leftarrow \sum_{i=1}^{t} w_i 2^{t-i}$  ▷ with $w_i$ being the $i$-th bit of $W$ from the left
  $(a, b) \leftarrow$ elements of $\mathbb{F}_p$, so that $rb^2 \equiv a^3 \mod p$
  **if** $4a^3 + 27b^2 \equiv 0 \mod p$ **then**
    **goto** 3
  **return** $SEED, a, b$

**Algorithm 4** ANSI X9.62 binary field algorithm [4]

**function** ANSI_X962_Binary($q = 2^m$)
  $t \leftarrow m$;  $s \leftarrow \lfloor (t-1)/160 \rfloor$;  $h \leftarrow t - 160s$
  $SEED \leftarrow$ random bit string of at least 160 bits
  $g \leftarrow |SEED|$
  $H \leftarrow$ SHA-1($SEED$)
  $b_0 \leftarrow$ bit string of $h$ rightmost bits of $H$
  **for** $i = 1$ **to** $s$ **do**
    $b_i \leftarrow$ SHA-1$((SEED + i) \mod 2^g)$
  $b \leftarrow b_0 \| b_1 \| \ldots \| b_s \in \mathbb{F}_{2^m}$
  **if** $b = 0$ **then**
    **goto** 3
  $a \leftarrow$ random element from $\mathbb{F}_{2^m}$
  **return** $SEED, a, b$

**Algorithm 5** Corrected complex multiplication algorithm from Bröker; Stevenhagen [65]

1: ▷ $N$ the desired (prime) order of the elliptic curve.
2: **function** Complex_multiplication($N$)
3: $r \leftarrow 0$
4: $S \leftarrow$ empty table
5: For all odd primes $p \in [r \log N, (r+1) \log N]$ that satisfy $\left(\frac{N}{p}\right) = 1$, compute $p^* = (-1)^{(p-1)/2} p$ and add $(p^*, \sqrt{p^*} \mod N)$ into the table S.
6: **for** each product $(D, \sqrt{D} \mod N) = (\prod_i p_i^*, \prod_i \sqrt{p_i^*} \mod N)$ **where** $D < ((r+1) \log N)^2$ **and** $D \equiv 5 \mod 8$ **do**
7: Use $\sqrt{D} \mod N$ and Cornacchia's algorithm to compute $x, y \in \mathbb{N}$ such that $x^2 - Dy^2 = 4N$
8: Test whether $p = N + 1 \pm x$ is prime. If it is, construct the Hilbert class polynomial $H_D(X) \in \mathbb{Z}[X]$ and compute its roots $j_i$ modulo $p$. If roots are found, half of them correspond to $j$-invariants of a curve with $N$ points and other half to its twist.
9: **if** $j$-invariant found **then**
10: **return** elliptic curve over $\mathbb{F}_p$ with $j$-invariant of $j_i$
11: $r \leftarrow r + 1$
12: **goto** 5

### Example generation results

ecgen --fp --random --prime 128


```
[{
    "field": {
        "p": "0xf905b3cebba4c9f406e79fe0e4e85f0b"
    },
    "a": "0xc9779c85106011d6a1c3067a3cc504d0",
    "b": "0x55fc08ef10f38524986ffd0972e3fc82",
    "order": "0xf905b3cebba4c9f5e5b44feafbfd50b3",
    "subgroups": [
        {
            "x": "0x82dd0e57527318d716c7b422553a5a10",
            "y": "0xd8deba8184831298731e62974c088820",
            "order": "0xf905b3cebba4c9f5e5b44feafbfd50b3",
            "cofactor": "0x1",
            "points": [
                {
                    "x": "0x82dd0e57527318d716c7b422553a5a10",
                    "y": "0xd8deba8184831298731e62974c088820",
                    "order": "0xf905b3cebba4c9f5e5b44feafbfd50b3"
                }
            ]
        }
    ]
}]
```

ecgen --f2m --random --cofactor=2 128


```
[{
    "field": {
        "m": "0x80",
        "e1": "0x7",
        "e2": "0x2",
        "e3": "0x1"
    },
    "a": "0xd1964726a0f377f52d38c0b746b64700",
    "b": "0x0178e45b662ce3a361bb091ae23f1f1b",
    "order": "0x1000000000000000022c0a2803e83193a",
    "subgroups": [
        {
            "x": "0xa41834f94800c270624f893278c7081f",
            "y": "0x0c997b71443ab553f2235c90471eb593",
            "order": "0x800000000000000000116051401f418c9d",
            "cofactor": "0x2",
            "points": [
                {
                    "x": "0xa41834f94800c270624f893278c7081f",
                    "y": "0x0c997b71443ab553f2235c90471eb593",
                    "order": "0x800000000000000000116051401f418c9d"
                }
            ]
        },
        {
            "x": "0xa3da3e5d6e60d2abaf769c43e0c3bbf5",
            "y": "0x99bdafb8a47b3b8d0bc7bea6e5ad9c88",
            "order": "0x1000000000000000022c0a2803e83193a",
            "cofactor": "0x1",
            "points": [
                {
                    "x": "0x00000000000000000000000000000000",
                    "y": "0x00c06618be20746b1ded08b857741bc8",
                    "order": "0x2"
                },
                {
                    "x": "0xa41834f94800c270624f893278c7081f",
                    "y": "0x0c997b71443ab553f2235c90471eb593",
                    "order": "0x800000000000000000116051401f418c9d"
                }
            ]
        }
    ]
}]
```

ecgen --fp --anomalous --random 128


```
[{
    "field": {
        "p": "0xaa04ed4b4d0f5d623ea49b3bfd84b8b7"
    },
    "a": "0x7c8b20d2322c0f56d8efc5f3302740bc",
    "b": "0x95fc1fff2d840a8a2a3fddf7c936ce25",
    "order": "0xaa04ed4b4d0f5d623ea49b3bfd84b8b7",
    "subgroups": [
        {
            "x": "0x6319388a49de962eacbf8a538def78b8",
            "y": "0xa73db44263c3797ff1bbe4a1652718b9",
            "order": "0xaa04ed4b4d0f5d623ea49b3bfd84b8b7",
            "cofactor": "0x1",
            "points": [
                {
                    "x": "0x6319388a49de962eacbf8a538def78b8",
                    "y": "0xa73db44263c3797ff1bbe4a1652718b9",
                    "order": "0xaa04ed4b4d0f5d623ea49b3bfd84b8b7"
                }
            ]
        }
    ]
}]
```

ecgen --fp --order=27252415630768662218630641864537700524 7 128

```
[{
    "field": {
        "p": "0xcd063e4a3b4947eacdcf05d10331b09b"
    },
    "a": "0x1da7d71e08858af1136269110d7e3ae3",
    "b": "0xe25828e1f77a750eec9d96eef281c51d",
    "order": "0xcd063e4a3b4947ec901cd5739ba8b6bf",
    "subgroups": [
        {
            "x": "0x89ba8b69ae2fb21e45af248f24c72f4b",
            "y": "0x72ff9254496fd49b587d441dd2f6ab57",
            "order": "0xcd063e4a3b4947ec901cd5739ba8b6bf",
            "cofactor": "0x1"
        }
    ]
}]
```

## Data attachment

- `ecgen`
  Sources of the ecgen tool. Also available on `https://github.com/J08nY/ecgen`.
- `ECTester`
  Sources of the ECTester tool. Also available on `https://github.com/crocs-muni/ECTester`.
- `ECTester_site`
  A static version of the ECTester site. Also hosted on `https://crocs-muni.github.io/ECTester`.

# Bibliography


1. SCHNEIER, Bruce. *The NSA Is Breaking Most Encryption on the Internet* [online]. 2013 [visited on 2018-05-16]. Available from: `https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html#c1675929`.
2. JAGER, Tibor; SCHWENK, Jörg; SOMOROVSKY, Juraj. Practical Invalid Curve Attacks on TLS-ECDH. In: *Proceedings of the 20th European Symposium on Computer Security*. Cham: Springer, 2015, vol. 9326, pp. 407–425. ESORICS '15. Available from DOI: `10.1007/978-3-319-24174-6_21`.
3. *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION 186-4 Digital Signature Standard (DSS)*. 2013-01. Available also from: `https://csrc.nist.gov/publications/detail/fips/186/4/final`. Standard. National Institute for Standards and Technology.
4. *American National Standard X9.62-2005, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)*. 2005. Standard. Accredited Standards Committee X9.
5. *IEEE Standard Specifications for Public-Key Cryptography*. 2000. Standard. IEEE Std 1363-2000 Working Group.
6. *ECC Brainpool Standard Curves and Curve Generation*. 2005. Available also from: `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`. Standard. ECC Brainpool.
7. LOCHTER, Manfred; MERKLE, Johannes. *RFC 5639: Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation* [Internet Requests for Comments]. RFC Editor, 2010-03. Available also from: `https://tools.ietf.org/html/rfc5639`. RFC.
8. MERKLE, Johannes; LOCHTER, Manfred. *RFC 7027: Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS)* [Internet Requests for Comments]. RFC Editor, 2013-10. Available also from: `https://tools.ietf.org/html/rfc7027`. RFC.
9. *SEC 1: Elliptic Curve Cryptography*. 2009-05. Available also from: `http://www.secg.org/sec1-v2.pdf`. Standard. Standards for Efficient Cryptography Group.
10. *SEC 2: Recommended Elliptic Curve Domain Parameters*. 2010-01. Available also from: `http://www.secg.org/sec2-v2.pdf`. Standard. Standards for Efficient Cryptography Group.
11. BERNSTEIN, Daniel J. Curve25519: new Diffie-Hellman speed records. In: *Proceedings of the 9th International Conference on Theory and Practice in Public Key Cryptography*. Berlin, Heidelberg: Springer, 2006-04, pp. 207–228. PKC '06. Available from DOI: `10.1007/11745853_14`.
12. HAMBURG, Mike. *Ed448-Goldilocks, a new elliptic curve* [Cryptology ePrint Archive, Report 2015/625]. 2015. Available also from: `http://eprint.iacr.org/2015/625`.
13. BOS, Joppe W.; COSTELLO, Craig; LONGA, Patrick; NAEHRIG, Michael. *Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis* [Cryptology ePrint Archive, Report 2014/130]. 2014. Available also from: `https://eprint.iacr.org/2014/130`.
14. SCHOOF, René. Counting points on elliptic curves over finite fields. *Journal de Théorie des Nombres Bordeaux*. 1995, vol. 7, no. 1, pp. 219–254. Available also from: `http://www.numdam.org/item?id=JTNB_1995__7_1_219_0`.

15. MIYAJI, Atsuko. Elliptic curves over $\mathbb{F}_p$ suitable for cryptosystems. In: *Proceedings of the 3rd International Conference on the Theory and Applications of Cryptology and Information Security*. Berlin, Heidelberg: Springer, 1992-12, pp. 477–491. ASIACRYPT '92. Available from DOI: 10.1007/3-540-57220-1_86.
16. BAIER, Harald; BUCHMANN, Jonannes. *Generation Methods of Elliptic Curves*. 2002-08. Available also from: https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1030_Buchmann.evaluation.pdf. report. Information-technology Promotion Agency of Japan.
17. BAIER, Harald. *Efficient algorithms for generating elliptic curves over finite fields suitable for use in cryptography*. 2002. PhD thesis. Technische Universität.
18. BERNSTEIN, Daniel J.; LANGE, Tanja. *SafeCurves: choosing safe curves for elliptic-curve cryptography* [online]. 2017-01 [visited on 2018-05-16]. Available from: https://safecurves.cr.yp.to/.
19. CERTIVOX. *MIRACL: Multiprecision Integer and Rational Arithmetic Cryptographic Library* [online]. 2018 [visited on 2018-05-16]. Available from: https://github.com/miracl/MIRACL.
20. KONSTANTINOU, Elisavet; STAMATIOU, Yannis; ZAROLIAGIS, Christos. *ECC-LIB : A Library for Elliptic Curve Cryptography* [online]. 2018 [visited on 2018-05-16]. Available from: https://www.ceid.upatras.gr/webpages/faculty/zaro/software/ecc-lib/.
21. ENGE, Andreas. *CM - Version 0.3* [online]. 2018 [visited on 2018-05-10]. Available from: http://www.multiprecision.org/cm/home.html.
22. *LiDIA - A library for computational number theory* [online]. 2018 [visited on 2018-05-16]. Available from: https://github.com/mkoeppe/LiDIA.
23. HANKERSON, Darrel; MENEZES, Alfred J.; VANSTONE, Scott. *Guide to Elliptic Curve Cryptography*. New York, USA: Springer, 2004. ISBN 9780387218465. Available from DOI: 10.1007/b97644.
24. COHEN, Henri; FREY, Gerhard; AVANZI, Roberto M.; DOCHE, Christophe; LANGE, Tanja; NGUYEN, Kim; VERCAUTEREN, Frederik. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005-07-19. Discrete Mathematics and It's Applications, no. 34. ISBN 9781584885184.
25. MENEZES, Alfred J.; VANSTONE, Scott A.; VAN OORSCHOT, Paul C. *Handbook of Applied Cryptography*. 1st ed. Boca Raton, USA: CRC Press, 1996-10. ISBN 9780849385230. Available also from: http://cacr.uwaterloo.ca/hac/.
26. BERNSTEIN, Daniel J.; LANGE, Tanja. *Explicit-Formulas Database* [online]. 2017 [visited on 2018-05-16]. Available from: https://hyperelliptic.org/EFD/.
27. MONTGOMERY, Peter L. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*. 1987, vol. 48, no. 177, pp. 243–264. Available from DOI: 10.1090/S0025-5718-1987-0866113-7.
28. BRIER, Eric; JOYE, Marc. Weierstraß elliptic curves and side-channel attacks. In: *Proceedings of the 5th International Conference on Theory and Practice in Public Key Cryptography*. Berlin, Heidelberg: Springer, 2002-02, pp. 335–345. PKC '02. Available from DOI: 10.1007/3-540-45664-3_24.
29. LÓPEZ, Julio; DAHAB, Ricardo. Fast Multiplication on Elliptic Curves over GF($2^m$) without Precomputation. In: *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems*. 1999-08, pp. 316–327. CHES '99. Available from DOI: 10.1007/3-540-48059-5_27.
30. EDWARDS, Harold. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*. 2007, vol. 44, no. 3, pp. 393–422. Available from DOI: 10.1090/S0273-0979-07-01153-6.

31. BERNSTEIN, Daniel J; LANGE, Tanja. Faster Addition and Doubling on Elliptic Curves. In: *Proceedings of the 13th International Conference on the Theory and Application of Cryptology and Information Security*. Berlin, Heidelberg: Springer, 2007-12, pp. 29–50. ASIACRYPT '07. Available from DOI: 10.1007/978-3-540-76900-2_3.
32. POHLIG, Stephen; HELLMAN, Martin. An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance. *IEEE Transactions on Information Theory*. 1978-01, vol. 24, no. 1, pp. 106–110. Available also from: https://www-ee.stanford.edu/~hellman/publications/28.pdf.
33. VAN OORSCHOT, Paul C.; WIENER, Michael J. On Diffie-Hellman Key Agreement with Short Exponents. In: *Proceedings of the 15th International Conference on the Theory and Application of Cryptographic Techniques*. Berlin, Heidelberg: Springer, 1996-05, pp. 332–343. EUROCRYPT '96. Available from DOI: 10.1007/3-540-68339-9_29.
34. ANDERSON, Ross; VAUDENAY, Serge. Minding Your P's and Q's. In: *Proceedings of the 5th International Conference on the Theory and Applications of Cryptology and Information Security*. Berlin, Heidelberg: Springer, 1996-11, pp. 26–35. ASIACRYPT '96. Available from DOI: 10.1007/BFb0034832.
35. SATOH, Takakazu; ARAKI, Kiyomichi. Fermat Quotients and the Polynomial Time Discrete Log Algorithm for Anomalous Elliptic Curves. *Commentarii mathematici Universitatis Sancti Pauli*. 1998-06, vol. 47, no. 1, pp. 81–92. Available from DOI: 10.14992/00009878.
36. SEMAEV, Igor A. Evaluation of discrete logarithms in a group of $p$-torsion points of an elliptic curve in characteristic $p$. *Mathematics of Computation*. 1998-01, vol. 67, no. 221, pp. 353–356. Available from DOI: 10.1090/S0025-5718-98-00887-4.
37. SMART, Nigel P. *The Discrete Logarithm Problem on Elliptic Curves of Trace One* [HP Laboratories Bristol]. 1997. Available also from: http://www.hpl.hp.com/techreports/97/HPL-97-128.pdf.
38. MENEZES, Alfred J.; VANSTONE, Scott; OKAMOTO, Tatsuaki. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. *IEEE Transactions on Information Theory*. 1993-09, vol. 39, no. 5, pp. 1639–1646. Available from DOI: 10.1109/18.259647.
39. FREY, Gerhard; RÜCK, Hans-Georg. A Remark Concerning $m$-divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*. 1994-04, vol. 62, no. 206, pp. 865–874. Available from DOI: 10.1090/S0025-5718-1994-1218343-6.
40. SEMAEV, Igor A. On the computation of logarithms on elliptic curves. *Discrete Mathematics and Applications*. 1996, vol. 6, no. 1, pp. 69–76. Available also from: http://mi.mathnet.ru/eng/dm516.
41. DIEM, Claus. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica*. 2011, vol. 147, no. 1, pp. 75–104. Available from DOI: 10.1112/S0010437X10005075.
42. GAUDRY, Pierrick. *Index calculus for abelian varieties and the elliptic curve discrete logarithm problem* [Cryptology ePrint Archive, Report 2004/073]. 2004. Available also from: https://eprint.iacr.org/2004/073.
43. DUURSMA, Iwan M.; GAUDRY, Pierrick; MORAIN, François. Speeding Up the Discrete Log Computation on Curves with Automorphisms. In: *Proceedings of the 7th International Conference on the Theory and Applications of Cryptology and Information Security*. Berlin, Heidelberg: Springer, 1999-11, pp. 103–121. ASIACRYPT '99. Available from DOI: 10.1007/978-3-540-48000-6_10.

44. WIENER, Michael J.; ZUCCHERATO, Robert J. Faster Attacks on Elliptic Curve Cryptosystems. In: *Proceedings of the 5th International Workshop on Selected Areas in Cryptography*. Berlin, Heidelberg: Springer, 1998-08, pp. 190–200. SAC '98. Available from DOI: 10.1007/3-540-48892-8_15.
45. GALLANT, Robert; LAMBERT, Robert; VANSTONE, Scott. Improving the Parallelized Pollard Lambda Search on Anomalous Binary Curves. *Mathematics of Computation*. 2000-10, vol. 69, no. 232, pp. 1699–1705. Available from DOI: 10.1090/S0025-5718-99-01119-9.
46. VALENTA, Luke; ADRIAN, David; SANSO, Antonio; COHNEY, Shaanan; FRIED, Joshua; HASTINGS, Marcella; HALDERMAN, J. Alex; HENINGER, Nadia. *Measuring small subgroup attacks against Diffie-Hellman* [Cryptology ePrint Archive, Report 2016/995]. 2016. Available also from: http://eprint.iacr.org/2016/995.
47. LIM, Chae Hoon; LEE, Pil Joong. A key recovery attack on discrete log-based schemes using a prime order subgroup. In: *Proceedings of the 17th Annual International Cryptology Conference*. Berlin, Heidelberg: Springer, 1997-08, pp. 249–263. CRYPTO '97. Available from DOI: 10.1007/BFb0052240.
48. BIEHL, Ingrid; MEYER, Bernd; MÜLLER, Volker. Differential Fault Attacks on Elliptic Curve Cryptosystems. In: *Proceedings of the 20th Annual International Cryptology Conference*. Berlin, Heidelberg: Springer, 2000-08, pp. 131–146. CRYPTO '00. Available from DOI: 10.1007/3-540-44598-6_8.
49. ANTIPA, Adrian; BROWN, Daniel R. L.; MENEZES, Alfred J.; STRUIK, René; VANSTONE, Scott A. Validation of Elliptic Curve Public Keys. In: *Proceedings of the 6th International Conference on Theory and Practice in Public Key Cryptography*. Berlin, Heidelberg: Springer, 2003-01, pp. 211–223. PKC '03. Available from DOI: 10.1007/3-540-36288-6_16.
50. FOUQUE, Pierre-Alain; LERCIER, Reynald; RÉAL, Denis; VALETTE, Frédéric. Fault attack on elliptic curve Montgomery ladder implementation. In: *Proceedings of the 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2008, pp. 92–98. FDTC'08. Available from DOI: 10.1109/FDTC.2008.15.
51. CIET, Mathieu; JOYE, Marc. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Designs, codes and cryptography*. 2005, vol. 36, no. 1, pp. 33–43. Available from DOI: 10.1007/s10623-003-1160-8.
52. SPAGNI, Riccardo "fluffypony". *Disclosure of a Major Bug in CryptoNote Based Currencies* [online]. 2017-05 [visited on 2018-05-16]. Available from: https://getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html.
53. GEIJERSTAM, Peter AF. *Bytecoin was exploited* [online]. 2017-05 [visited on 2018-05-20]. Available from: http://shellcode.se/hacking/bytecoin_exploited/.
54. ŠVENDA, Petr; JANČÁR, Ján. *ECTester* [online]. 2018 [visited on 2018-05-16]. Available from: https://github.com/crocs-muni/ECTester.
55. SATOH, Takakazu; SKJERNAA, Berit; TAGUCHI, Yuichiro. Fast computation of canonical lifts of elliptic curves and its application to point counting. *Finite Fields and Their Applications*. 2003, vol. 9, no. 1, pp. 89–101. Available from DOI: 10.1016/S1071-5797(02)00013-8.
56. MESTRE, Jean-Francois. *Lettre adressée à Gaudry et Harley*. 2000. Available also from: https://webusers.imj-prg.fr/~jean-francois.mestre/lettreGaudryHarley.ps.
57. THE PARI GROUP. *PARI/GP version* 2.9.4 [online]. University Bordeaux, 2018 [visited on 2018-05-16]. Available from: https://pari.math.u-bordeaux.fr.

58. *American National Standard X9.62-1998, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)*. 1998. Preliminary draft. Accredited Standards Committee X9.
59. *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION 186-2 Digital Signature Standard (DSS)*. 2000-01. Available also from: `http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf`. Standard. National Institute for Standards and Technology.
60. *SEC 2: Recommended Elliptic Curve Domain Parameters*. 2000-09. Available also from: `http://www.secg.org/SEC2-Ver-1.0.pdf`. Standard (superseded). Standards for Efficient Cryptography Group.
61. BERNSTEIN, Daniel J.; CHOU, Tung; CHUENGSATIANSUP, Chitchanok; HÜLSING, Andreas; LANGE, Tanja; NIEDERHAGEN, Ruben; VREDENDAAL, Christine van. *How to manipulate curve standards: a white paper for the black hat* [Cryptology ePrint Archive, Report 2014/571]. 2014. Available also from: `http://eprint.iacr.org/2014/571`.
62. SILVERMAN, Joseph H. *The Arithmetic of Elliptic Curves*. 2nd ed. New York, USA: Springer, 2009. Graduate Texts in Mathematics, no. 106. ISBN 9780387094946. Available from DOI: `10.1007/978-0-387-09494-6`.
63. BRÖKER, Reinier; STEVENHAGEN, Peter. Efficient CM-constructions of elliptic curves over finite fields. *Mathematics of Computation*. 2007, vol. 76, no. 260, pp. 2161–2180. Available from DOI: `10.1090/S0025-5718-07-01980-1`.
64. JANČÁR, Ján. *ecgen* [online]. 2018 [visited on 2018-05-16]. Available from: `https://github.com/J08nY/ecgen`.
65. BRÖKER, Reinier; STEVENHAGEN, Peter. Constructing elliptic curves of prime order. *Contemporary Mathematics*. 2008, vol. 463, pp. 17–28. Available also from: `https://arxiv.org/abs/0712.2022`.
66. *ISO/IEC 7816-4:2013 Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*. 2013. Available also from: `https://www.iso.org/standard/54550.html`. Standard. ISO/IEC.
67. *SP 800-56A Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive Testing*. 2011-04. Available also from: `https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Algorithm-Validation-Program/documents/components/ecccdhvs.pdf`. Standard. National Institute for Standards and Technology.
68. *GEC 2: Test Vectors for SEC 1*. 1999-09. Available also from: `http://read.pudn.com/downloads168/doc/772358/TestVectorsforSEC%201-gec2.pdf`. Draft. Standards for Efficient Cryptography Group.
69. HARKINS, Dan. *RFC 6932: Brainpool Elliptic Curves for the Internet Key Exchange (IKE) Group Description Registry* [Internet Requests for Comments]. RFC Editor, 2013-05. Available also from: `https://tools.ietf.org/html/rfc6932`. RFC.