

---

## 1 Operational notes
































2 Document updated on **February 4, 2022**.

3 The following colors are **not** part of the final product, but serve as highlights in the edit-  
4 ing/review process:

- 5 • text that needs attention from the Subject Matter Experts: Mirco, Anna,& Jan
- 6 • terms that have not yet been defined in the book
- 7 • text that needs advice from the communications/marketing team: Aaron & Shane
- 8 • text that needs to be completed or otherwise edited (by Sylvia)

















































9 NB: This PDF only includes the Statements chapter

# 10 **Todo list**

















































11	 zero-knowledge proofs . . . . .	12
12	 played with . . . . .	12
13	 finite field . . . . .	12
14	 elliptic curve . . . . .	12
15	 Update reference when content is finalized . . . . .	12
16	 methatical . . . . .	12
17	 numerical . . . . .	12
18	 a list of additional exercises . . . . .	13
19	 think about them . . . . .	13
20	 add some more informal explanation of absolute value . . . . .	14
21	 We haven't really talked about what a ring is at this point . . . . .	14
22	 What's the significance of this distinction? . . . . .	15
23	 reverse . . . . .	15
24	 Turing machine . . . . .	15
25	 polynomial time . . . . .	15
26	 sub-exponentially, with $\mathcal{O}((1 + \varepsilon)^n)$ and some $\varepsilon > 0$ . . . . .	15
27	 Add text . . . . .	16
28	 $\mathbb{Q}$ of fractions . . . . .	16
29	 Division in the usual sense is not defined for integers . . . . .	16
30	 Add more explanation of how this works . . . . .	17
31	 pseudocode . . . . .	18
32	 modular arithmetics . . . . .	18
33	 actual division . . . . .	18
34	 multiplicative inverses . . . . .	18
35	 factional numbers . . . . .	18
36	 exponentiation function . . . . .	20
37	 See XXX . . . . .	20
38	 once they accept that this is a new kind of calculations, its actually not that hard . . . . .	20
39	 perform Euclidean division on them . . . . .	20
40	 This Sage snippet should be described in more detail. . . . .	21
41	 prime fields . . . . .	23
42	 residue class rings . . . . .	23
43	 Algorithm sometimes floated to the next page, check this for final version . . . . .	23
44	 Add a number and title to the tables . . . . .	25
45	 (-1) should be (-a)? . . . . .	26
46	 we have . . . . .	28
47	 rephrase . . . . .	32
48	 subtrahend . . . . .	33
49	 minuend . . . . .	33

50	what does this mean? . . . . .	37
51	TODO: Elliptic Curve asymmetric cryptography examples. Private key, generator,	
52	public key. . . . .	68
53	add reference . . . . .	68
54	rephrase . . . . .	70
55	add reference . . . . .	70
56	add reference . . . . .	71
57	jubjub . . . . .	71
58	add reference . . . . .	71
59	add reference . . . . .	72
60	add reference . . . . .	72
61	add reference . . . . .	73
62	add reference . . . . .	73
63	add reference . . . . .	74
64	add reference . . . . .	75
65	add reference . . . . .	75
66	add reference . . . . .	76
67	add reference . . . . .	77
68	add reference . . . . .	77
69	add reference . . . . .	77
70	add reference . . . . .	77
71	add reference . . . . .	78
72	add reference . . . . .	78
73	add reference . . . . .	78
74	add reference . . . . .	78
75	add reference . . . . .	78
76	add reference . . . . .	78
77	add reference . . . . .	79
78	add reference . . . . .	79
79	add reference . . . . .	79
80	add reference . . . . .	79
81	add reference . . . . .	79
82	add reference . . . . .	80
83	add reference . . . . .	80
84	add reference . . . . .	80
85	add reference . . . . .	81
86	add reference . . . . .	81
87	add reference . . . . .	81
88	add reference . . . . .	82
89	add reference . . . . .	82
90	add reference . . . . .	82
91	add reference . . . . .	82
92	add reference . . . . .	82
93	add reference . . . . .	82
94	add reference . . . . .	82
95	add reference . . . . .	82
96	add reference . . . . .	82
97	add reference . . . . .	82





---

98	 add reference . . . . .	83
99	 add reference . . . . .	83
100	 add reference . . . . .	83
101	 add reference . . . . .	85
102	 add reference . . . . .	85
103	 add reference . . . . .	85
104	 add reference . . . . .	86
105	 add reference . . . . .	87
106	 add reference . . . . .	87
107	 add reference . . . . .	87
108	 add reference . . . . .	87
109	 add reference . . . . .	87
110	 add reference . . . . .	87
111	 add reference . . . . .	88
112	 add reference . . . . .	89
113	 add reference . . . . .	89
114	 add reference . . . . .	89
115	 add reference . . . . .	89
116	 add reference . . . . .	90
117	 add reference . . . . .	91
118	 add reference . . . . .	91
119	 add reference . . . . .	91
120	 add reference . . . . .	92
121	 add reference . . . . .	92
122	 add reference . . . . .	92
123	 add reference . . . . .	92
124	 add reference . . . . .	92
125	 add reference . . . . .	93
126	 add reference . . . . .	93
127	 add reference . . . . .	93
128	 add reference . . . . .	93
129	 add reference . . . . .	93
130	 add reference . . . . .	93
131	 add reference . . . . .	94
132	 add reference . . . . .	94
133	 add reference . . . . .	94
134	 add reference . . . . .	94
135	 add reference . . . . .	94
136	 add reference . . . . .	95
137	 add reference . . . . .	95
138	 add reference . . . . .	96
139	 add reference . . . . .	96
140	 add reference . . . . .	97
141	 add reference . . . . .	97
142	 add reference . . . . .	97
143	 add reference . . . . .	98
144	 add reference . . . . .	98
145	 add reference . . . . .	98

---

146	 add reference . . . . .	98
147	 add reference . . . . .	100
148	 add reference . . . . .	101
149	 add reference . . . . .	101
150	 add reference . . . . .	101
151	 add reference . . . . .	101
152	 add reference . . . . .	103
153	 add reference . . . . .	103
154	 add reference . . . . .	104
155	 add reference . . . . .	104
156	 add reference . . . . .	104
157	 oberse . . . . .	104
158	 add reference . . . . .	104
159	 add reference . . . . .	104
160	 add reference . . . . .	105
161	 add reference . . . . .	105
162	 add reference . . . . .	105
163	 add reference . . . . .	106
164	 add reference . . . . .	106
165	 add reference . . . . .	106
166	 add reference . . . . .	107
167	 add reference . . . . .	108
168	 add reference . . . . .	109
169	 add reference . . . . .	109
170	 add reference . . . . .	110
171	 add reference . . . . .	110
172	 add reference . . . . .	111
173	 add reference . . . . .	112
174	 add reference . . . . .	112
175	 add reference . . . . .	112
176	 add reference . . . . .	113
177	 add reference . . . . .	114
178	 add reference . . . . .	114
179	 add reference . . . . .	115
180	 add reference . . . . .	115
181	 add reference . . . . .	115
182	 add reference . . . . .	116
183	 add reference . . . . .	116
184	 add reference . . . . .	116
185	 add reference . . . . .	118
186	 add reference . . . . .	118
187	 add reference . . . . .	118
188	 add reference . . . . .	118
189	 add reference . . . . .	119
190	 add reference . . . . .	119
191	 add reference . . . . .	119
192	 add reference . . . . .	119
193	 add reference . . . . .	119

---

194	 add reference . . . . .	120
195	 add reference . . . . .	120
196	 add reference . . . . .	120
197	 add reference . . . . .	120
198	 add reference . . . . .	121
199	 add reference . . . . .	121

200

# MoonMath manual

201

TechnoBob and the Least Scruples crew

202

February 4, 2022

# Contents

204	<b>1</b>	<b>Introduction</b>	<b>5</b>
205	1.1	Target audience . . . . .	5
206	1.2	The Zoo of Zero-Knowledge Proofs . . . . .	6
207		To Do List . . . . .	8
208		Points to cover while writing . . . . .	8
209	<b>2</b>	<b>Preliminaries</b>	<b>9</b>
210	2.1	Preface and Acknowledgements . . . . .	9
211	2.2	Purpose of the book . . . . .	9
212	2.3	How to read this book . . . . .	10
213	2.4	Cryptological Systems . . . . .	10
214	2.5	SNARKS . . . . .	10
215	2.6	complexity theory . . . . .	10
216	2.6.1	Runtime complexity . . . . .	10
217	2.7	Software Used in This Book . . . . .	11
218	2.7.1	Sagemath . . . . .	11
219	<b>3</b>	<b>Arithmetics</b>	<b>12</b>
220	3.1	Introduction . . . . .	12
221	3.1.1	Aims and target audience . . . . .	12
222	3.1.2	The structure of this chapter . . . . .	13
223	3.2	Integer Arithmetics . . . . .	13
224		Euclidean Division . . . . .	16
225		The Extended Euclidean Algorithm . . . . .	18
226	3.3	Modular arithmetic . . . . .	19
227		Congurency . . . . .	20
228		Modular Arithmetics . . . . .	20
229		The Chinese Remainder Theorem . . . . .	23
230		Modular Inverses . . . . .	26
231	3.4	Polynomial Arithmetics . . . . .	29
232		Polynomial Arithmetics . . . . .	33
233		Euklidean Division . . . . .	34
234		Prime Factors . . . . .	36
235		Lange interpolation . . . . .	37
236	<b>4</b>	<b>Algebra</b>	<b>40</b>
237	4.1	Groups . . . . .	40
238		Commutative Groups . . . . .	41
239		Finite groups . . . . .	43



240		Generators . . . . .	43
241		The discrete Logarithm problem . . . . .	43
242	4.1.1	Cryptographic Groups . . . . .	44
243		The discret logarithm assumption . . . . .	45
244		The decisional Diffi Hellman assumption . . . . .	47
245		The computational Diffi Hellman assumption . . . . .	47
246		Cofactor Clearing . . . . .	48
247	4.1.2	Hashing to Groups . . . . .	48
248		Hash functions . . . . .	48
249		Hashing to cyclic groups . . . . .	50
250		Hashing to modular arithmetics . . . . .	51
251		Pederson Hashes . . . . .	54
252		MimC Hashes . . . . .	55
253		Pseudo Random Functions in DDH-A groups . . . . .	55
254	4.2	Commutative Rings . . . . .	55
255		Hashing to Commutative Rings . . . . .	58
256	4.3	Fields . . . . .	58
257		Prime fields . . . . .	59
258		Square Roots . . . . .	60
259		Exponentiation . . . . .	62
260		Hashing into Prime fields . . . . .	62
261		Extension Fields . . . . .	62
262		Hashing into extension fields . . . . .	65
263	4.4	Projective Planes . . . . .	65
264	<b>5</b>	<b>Elliptic Curves</b>	<b>68</b>
265	5.1	Elliptic Curve Arithmetics . . . . .	68
266	5.1.1	Short Weierstraß Curves . . . . .	68
267		Affine short Weierstraß form . . . . .	69
268		Affine compressed representation . . . . .	73
269		Affine group law . . . . .	74
270		Scalar multiplication . . . . .	78
271		Projective short Weierstraß form . . . . .	81
272		Projective Group law . . . . .	83
273		Coordinate Transformations . . . . .	83
274	5.1.2	Montgomery Curves . . . . .	83
275		Affine Montgomery Form . . . . .	85
276		Affine Montgomery coordinate transformation . . . . .	86
277		Montgomery group law . . . . .	88
278	5.1.3	Twisted Edwards Curves . . . . .	88
279		Twisted Edwards Form . . . . .	89
280		Twisted Edwards group law . . . . .	90
281	5.2	Elliptic Curves Pairings . . . . .	91
282		Embedding Degrees . . . . .	91
283		Elliptic Curves over extension fields . . . . .	93
284		Full Torsion groups . . . . .	94
285		Torsion-Subgroups . . . . .	96
286		The Weil Pairing . . . . .	98

287	5.3	Hashing to Curves . . . . .	101
288		Try and increment hash functions . . . . .	101
289	5.4	Constructing elliptic curves . . . . .	103
290		The Trace of Frobenius . . . . .	104
291		The $j$ -invariant . . . . .	105
292		The Complex Multiplication Method . . . . .	106
293		The <i>BLS6_6</i> pen& paper curve . . . . .	114
294		Hashing to the pairing groups . . . . .	121
295	9	Exercises and Solutions	186

## Chapter 5

# Elliptic Curves

Generally speaking, elliptic curves are “curves” defined in geometric planes like the Euclidean or the projective plane over some given field. One of the key features of elliptic curves over finite fields from the point of view of cryptography is their set of points has a group law, such that the resulting group is finite and cyclic and it is believed that the discrete logarithm problem on these groups is hard.

A special class of elliptic curves are so called **pairing friendly curve**, which have a notation of a group pairing as defined in XXX. This pairing has cryptographicall nice prperties. Those curve are useful in the development of SNAKS, since they allow to compute so called R1CS-satisfiability "in the exponent" **MIRCO: (THIS HAS TO BE REWRITTEN WITH WAY MORE DETAIL)**

In this chapter we introduce epileptic curves as they are used in pairing based approaches to the construction of SNARKs. The elliptic curves we consider are all defined over prime fields or prime field extensions and the reader should be familiar with the contend of the previous section on those fields.

In its most generality elliptic curves are defined as a smooth projective curve of genus 1 defined over some field  $\mathbb{F}$  with a distinguished  $\mathbb{F}$ -rational point, but this definition is not very useful for the introductory character of this book. We will therefore look at 3 more practical definitions in the following sections, by introducing Weierstraß, Montgomery and Edwards curves. All of them are useful in cryptography and necessary to understand for the continuation of the book.

## 5.1 Elliptic Curve Arithmetics

### 5.1.1 Short Weierstraß Curves

In this section we introduce the so called short Weierstraß curves, which are the most general types of curves over finite fields of characteristic greater then 3.

We start with their representation in affine space. This representation has the advantage that affine points are just pairs of numbers which is more convenient to work with for the beginner. However, it has the disadvantage that a special "point at infinity" that is not a point on the curve, is necessary to describe the group structure. We introduce the elliptic curve group law and describe elliptic curve scalar multiplication, which is nothing but an instantiation of the exponential map from general cyclic groups.

Then we look at the projective representation of short Weierstraß curves. It has the advantage that no special symbol is necessary to represent the point at infinity but comes with

TODO:  
Elliptic  
Curve  
asymmet-  
ric cryp-  
tography  
examples.  
Private  
key, gen-  
erator,  
public  
key.

add refer-  
ence

the drawback that projective points are classes of numbers, which might be a bit unusual for a beginner.

We finish this section with an explicit equivalence that transforms affine representations into projective once and vice versa.

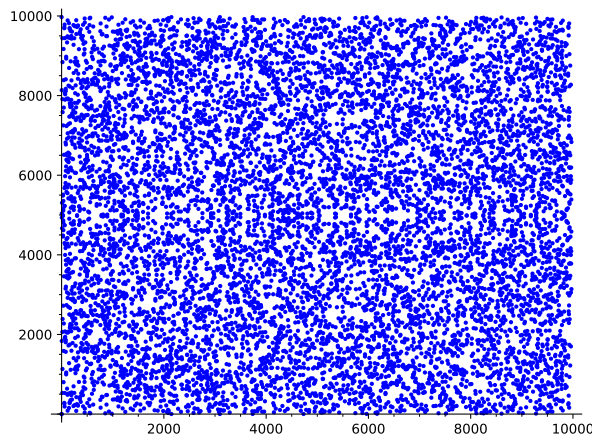
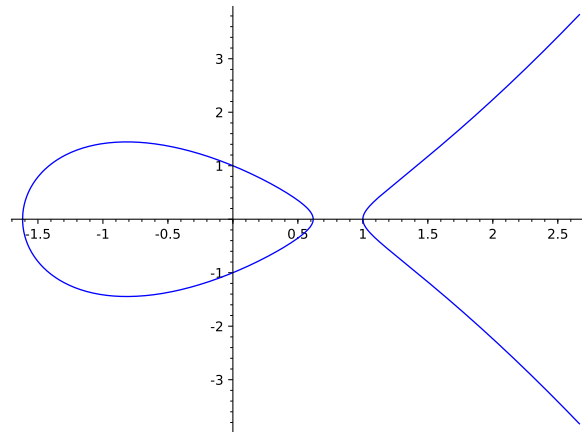
**Affine short Weierstraß form** Probably the least abstract and most straight forward way to introduce elliptic curves for non-mathematicians and beginners is the so called affine representation of a short Weierstraß curve. To see what this is, let  $\mathbb{F}$  be a finite field of order  $q$  and  $a, b \in \mathbb{F}$  two field elements such that  $4a^3 + 27b^2 \bmod q \neq 0$ . Then a **short Weierstraß elliptic curve**  $E(\mathbb{F})$  over  $\mathbb{F}$  in its affine representation is the set

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + a \cdot x + b\} \cup \{\mathcal{O}\} \quad (5.1)$$

of all pairs of field elements  $(x, y) \in \mathbb{F} \times \mathbb{F}$ , that satisfy the short Weierstraß cubic equation  $y^2 = x^3 + a \cdot x + b$ , together with a distinguished symbol  $\mathcal{O}$ , called the **point at infinity**.

*Notation and Symbols 7.* In the literature, the set  $E(\mathbb{F})$ , which includes the sybol  $\mathcal{O}$  is often called the set of **rational points** of the elliptic curve, in which case the curve itself is usually written as  $E/\mathbb{F}$ . However, in what follows we will frequently identify an elliptic curve with its set of rational points and therefore use the symbol  $E(\mathbb{F})$  instead. This is possible in our case, since we only really care about the group structure of the curve in consideration.

The term "curve" appears, because in the ordinary 2 dimensional plane  $\mathbb{R}^2$ , the set of all points  $(x, y)$  that satisfy  $y^2 = x^3 + a \cdot x + b$  looks like a curve. We should note however, that visualizing elliptic curves over finite fields as "curves" has its limitations and we will therefore not stress the geometric picture too much, but focus on the computational properties instead. To understand the visual difference, consider the following two elliptic curves:



Both elliptic curves are defined by the same short Weierstraß equation  $y^2 = x^3 - 2x + 1$ , but the first curve is defined in the real affine plane  $\mathbb{R}^2$ , that is the pair  $(x, y)$  contains real numbers, while the second one is defined in the affine plane  $\mathbb{F}_{9973}^2$ , which means that both  $x$  and  $y$  are from the prime field  $\mathbb{F}_{9973}$ . Every blue dot represents a pair  $(x, y)$  that is solution to  $y^2 = x^3 - 2x + 1$  and as we can see the second curve hardly looks like a geometric structure one would naturally call a curve. So the geometric intuitions from  $\mathbb{R}^2$  is kind of obfuscated in curves over finite fields.

The identity  $6 \cdot (4a^3 + 27b^2) \bmod q \neq 0$  ensures that the curve is non-singular, which basically means that the curve has no cusps or self-intersections.

When dealing with elliptic curves, computations can quickly become cumbersome and tedious. So, on the one hand, the reader is advised to do as many computations in a pen-and-paper style as possible. This helps a lot to get a deeper understanding for the details. On the other hand side however, computations are sometimes simply too large to be done by hand and one might get lost in the details. Fortunately Sage is very helpful in dealing with elliptic curves. It is there a goal of this book to introduce the reader to the great elliptic curve capabilities of Sage.

One we to define elliptic curves and work is them goes like this:

rephrase

```
Sage: F5 = GF(5) # define the base field
Sage: a = F5(2) # parameter a
Sage: b = F5(4) # parameter b
Sage: # check non-singularity
Sage: F5(6) * (F5(4) * a^3 + F5(27) * b^2) != F5(0)
Sage: # short Weierstraß curve
Sage: E = EllipticCurve(F5, [a, b]) # y^2 == x^3 + ax + b
Sage: P = E(0, 2) # 2^2 == 0^3 + 2*0 + 4
Sage: P.xy() # affine coordinates
Sage: INF = E(0) # point at infinity
Sage: try: # point at infinity has no affine coordinates
.....:     INF.xy()
.....: except ZeroDivisionError:
.....:     pass
Sage: P = E.plot() # create a plotted version
```

The following three examples will give a more practical understanding of what an elliptic curve is and how we can compute them. The reader is advised to read them carefully and ideally to parallel the computation themselves. We will repeatedly build on these example in this chapter and use the second example at various places in this book.

*Example 65.* To provide the reader with a small example of an elliptic curve, where all computation can be done in a pen-and-paper style, consider the prime field  $\mathbb{F}_5$  from example (XXX). The reader who had worked through the examples and exercises in the previous section knows this prime field well.

add reference

To define an elliptic curve over  $\mathbb{F}_5$ , we have to choose two numbers  $a$  and  $b$  from that field. Assuming we choose  $a = 1$  and  $b = 1$  then  $4a^3 + 27b^2 \equiv 1 \pmod{5}$  from which follows that the corresponding elliptic curve  $E_1(\mathbb{F}_5)$  is given by the set of all pairs  $(x, y)$  from  $\mathbb{F}_5$  that satisfy the equation  $y^2 = x^3 + x + 1$ , together with the special symbol  $\mathcal{O}$ , which represents the "point at infinity".

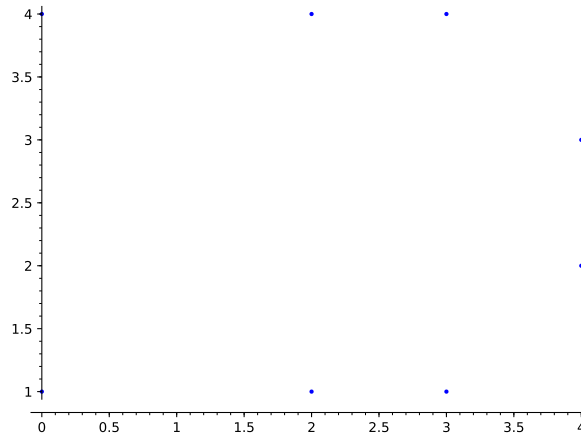
To get a better understanding of that curve, observe that if we choose arbitrarily the pair  $(x, y) = (1, 1)$ , we see that  $1^2 \neq 1^3 + 1 + 1$  and hence  $(1, 1)$  is not an element of the curve  $E_1(\mathbb{F}_5)$ . On the other hand choosing for example  $(x, y) = (2, 1)$  gives  $1^2 = 2^3 + 2 + 1$  and hence

the pair  $(2, 1)$  is an element of  $E_1(\mathbb{F}_5)$  (Remember that all computations are done in modulo 5 arithmetics).

Now since the set  $\mathbb{F}_5 \times \mathbb{F}_5$  of all pairs  $(x, y)$  from  $\mathbb{F}_5$  contains only  $5 \cdot 5 = 25$  pairs, we can compute the curve, by just inserting every possible pair  $(x, y)$  into the short Weierstraß equation  $y^2 = x^3 + x + 1$ . If the equation holds, the pair is a curve point, if not that means that the point is not on the curve. Combining the result of this computation with the point at infinity gives the curve as:

$$E_1(\mathbb{F}_5) = \{\mathcal{O}, (0, 1), (2, 1), (3, 1), (4, 2), (4, 3), (0, 4), (2, 4), (3, 4)\}$$

So our elliptic curve is a set of 9 elements. 8 of which are pairs of numbers and one special symbol  $\mathcal{O}$ . Visualizing  $E_1$  gives:



In the development of SNARKS it is sometimes necessary to do elliptic curve cryptograph "in a circuit", which basically means that the elliptic curves needs to be implemented in a certain SNARK-friendly way. We will look at what this means in XXX. To be able to do this efficiently it is desirable to have curves with special properties. The following example is a pen-and-paper version of such a curve, that parallels the definition of a cryptographically secure curve called **Baby-jubjub**, which is extensively used in real world SNARKs. The interested reader is advised to read this example carefully as we will use it and build on it in various places throughout the book.

*Example 66 (Pen-jubjub).* Consider the prime field  $\mathbb{F}_{13}$  from exercise XXX. If we choose  $a = 8$  and  $b = 8$  then  $4a^3 + 27b^2 \equiv 6 \pmod{13}$  and the corresponding elliptic curve is given by all pairs  $(x, y)$  from  $\mathbb{F}_{13}$  such that  $y^2 = x^3 + 8x + 8$  holds. We write  $PJJ_{13}$  for this curve and call it the **Pen-jubjub** curve.

Now, since the set  $\mathbb{F}_{13} \times \mathbb{F}_{13}$  of all pairs  $(x, y)$  from  $\mathbb{F}_{13}$  contains only  $13 \cdot 13 = 169$  pairs, we can compute the curve, by just inserting every possible pair  $(x, y)$  into the short Weierstraß equation  $y^2 = x^3 + 8x + 8$ . We get

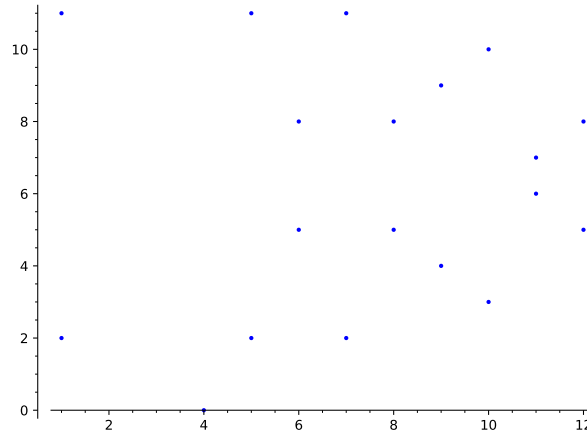
$$PJJ_{13} = \{\mathcal{O}, (1, 2), (1, 11), (4, 0), (5, 2), (5, 11), (6, 5), (6, 8), (7, 2), (7, 11), (8, 5), (8, 8), (9, 4), (9, 9), (10, 3), (10, 10), (11, 6), (11, 7), (12, 5), (12, 8)\}$$

As we can see the curve consist of 20 points. 19 points from the affine plane and the point at infinity. To get a visual impression of the  $PJJ_{13}$  curve, we might plot all of its points (except the point at infinity) in the  $\mathbb{F}_{13} \times \mathbb{F}_{13}$  affine plane. We get:

add reference

jubjub

add reference



As we will see in what follows this curve is kind of special as it is possible to represent it in two alternative forms, called the Montgomery and the twisted Edwards form (See XXX and XXX).

Now that we have seen two pen-and-paper friendly elliptic curves, let us look at a curve that is used in actual cryptography. Cryptographically secure elliptic curve are not qualitatively different from the curves we looked at so far. The only difference is that the prime number modulus of the prime field is much larger. Typical examples use prime numbers, which have binary representations in the size of more than double the size of the desired security level. So if for example a security of 128 bit is desired, a prime modulus of binary size  $\geq 256$  is chosen. The following example provides such a curve.

*Example 67* (Bitcoin's Secp256k1 curve). To give an example of a real world, cryptographically secure curve, let us look at curve Secp256k1, which is famous for being used in the public key cryptography of Bitcoin. The prime field  $\mathbb{F}_p$  of Secp256k1 is defined by the prime number

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$

which has a binary representation that need 256 bits. This implies that the  $\mathbb{F}_p$  approximately contains  $2^{256}$  many elements. So the underlying field is large. To get an image of how large the base field is, consider that the number  $2^{256}$  is approximately in the same order of magnitude as the estimated number of atoms in the observable universe.

Curve Secp256k1 is then defined by the parameters  $a, b \in \mathbb{F}_p$  with  $a = 0$  and  $b = 7$ . Since  $4 \cdot 0^3 + 27 \cdot 7^2 \mod p = 1323$ , those parameters indeed define an elliptic curve given by

$$\text{Secp256k1} = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p \mid y^2 = x^3 + 7\}$$

Clearly Secp256k1 is a curve, too large to do computations by hand, since it can be shown that Secp256k1 contains

$$r = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

many elements, where  $r$  is a prime number that also has a binary representation of 256 bits. Cryptographically secure elliptic curves are therefore not useful in pen-and-paper computations. Fortunately Sage handles large curve efficiently:

```
Sage: p = 115792089237316195423570985008687907853269984665640564039457584
Sage: # Hexadecimal representation
Sage: p.str(16)
Sage: p.is_prime()
```

add reference

add reference

```

441 Sage: p.nbits()
442 Sage: Fp = GF(p)
443 Sage: Secp256k1 = EllipticCurve(Fp, [0, 7])
444 Sage: r = Secp256k1.order() # number of elements
445 Sage: r.str(16)
446 Sage: r.is_prime()
447 Sage: r.nbits()

```

448 *Exercise 35.* Look-up the definition of curve BLS12-381, implement it in Sage and computes  
 449 its order.

450 **Affine compressed representation** As we have seen in example XXX, cryptographically se-  
 451 cure elliptic curves are defined over large prime fields, where elements of those fields typically  
 452 need more than 255 bits storage on a computer. Since elliptic curve points consists of pairs of  
 453 those field element, they need double that amount of storage.

add refer-  
ence

454 To reduce the amount of space needed to represent a curve point note however, that up to  
 455 a sign the  $y$ -coordinate of a curve point can be computed from the  $x$ -coordinate, by simply  
 456 inserting  $x$  into the Weierstraß equation and then computing the roots of the result. This gives  
 457 two results and it follows that we can represent a curve point in **compressed form** by simply  
 458 storing the  $x$ -coordinate together with a single sign bit only, the latter of which deterministically  
 459 decides which of the two roots to choose. In case that the  $y$ -coordinate is zero, both sign bit give  
 460 the same result.

461 For example one convention could be to always choose the root closer to 0, when the sign  
 462 bit is 0 and the root closer to the order of  $\mathbb{F}$  when the sign bit is 1.

*Example 68* (Pen-jubjub). To understand the concept of compressed curve points a bit better  
 consider the *PJJ\_13* curve from example XXX again. Since this curve is defined over the prime  
 field  $\mathbb{F}_{13}$  and numbers between 0 and 13 need approximately 4 bits to be represented, each  
*PJJ\_13* -point need 8-bits of storage in uncompressed form, while it would need only 5 bits in  
 compressed form. To see how this works, recall that in uncompressed form we have

add refer-  
ence

$$PJJ_{13} = \{\mathcal{O}, (1, 2), (1, 11), (4, 0), (5, 2), (5, 11), (6, 5), (6, 8), (7, 2), (7, 11), \\ (8, 5), (8, 8), (9, 4), (9, 9), (10, 3), (10, 10), (11, 6), (11, 7), (12, 5), (12, 8)\}$$

Using the technique of point compression, we can replace the  $y$ -coordinate in each  $(x, y)$  pair by  
 a sign bit, indicating, wheather or not  $y$  is closer to 0 or to 13. So  $y$  values in the range  $[0, \dots, 6]$   
 having sign bit 0 and  $y$ -values in the range  $[7, \dots, 12]$  having sign bit 1. Applying this to the  
 points in *PJJ\_13* gives the compressed representation:

$$PJJ_{13} = \{\mathcal{O}, (1, 0), (1, 1), (4, 0), (5, 0), (5, 1), (6, 0), (6, 1), (7, 0), (7, 1), \\ (8, 0), (8, 1), (9, 0), (9, 1), (10, 0), (10, 1), (11, 0), (11, 1), (12, 0), (12, 1)\}$$

463 Note that the numbers  $7, \dots, 12$  are the negatives (additive inverses) of the numbers  $1, \dots, 6$  in  
 464 modular 13 arithmetics and that  $-0 = 0$ . Calling the compression bit a "sign bit" therefore  
 465 makes sense.

466 To recover the uncompressed point of say  $(5, 1)$ , we insert the  $x$ -coordinate 5 into the Weier-  
 467 straß equation and get  $y^2 = 5^3 + 8 \cdot 5 + 8 = 4$ . As expected 4 is a quadratic residue in  $\mathbb{F}_{13}$  with  
 468 roots  $\sqrt{4} = \{2, 11\}$ . Now since the sign bit of the point is 1, we have to choose the root closer  
 469 to the modulus 13 which is 11. The uncompressed point is therefore  $(5, 11)$ .



Looking at the previous examples, compression rate looks not very impressive. The following example therefore looks at the Secp256k1 curve to show that compression is actually useful.

*Example 69.* Consider the Secp256k1 curve from example XXX again. The following code involves Sage to generate a random affine curve point, we then apply our compression method

add reference

```
Sage: P = Secp256k1.random_point().xy()
Sage: P
Sage: # uncompressed affine point size
Sage: ZZ(P[0]).nbits()+ZZ(P[1]).nbits()
Sage: # compute the compression
Sage: if P[1] > Fp(-1)/Fp(2):
.....:     PARITY = 1
.....: else:
.....:     PARITY = 0
Sage: PCOMPRESSED = [P[0], PARITY]
Sage: PCOMPRESSED
Sage: # compressed affine point size
Sage: ZZ(PCOMPRESSED[0]).nbits()+ZZ(PCOMPRESSED[1]).nbits()
```

**Affine group law** One of the key properties of an elliptic curve is that it is possible to define a group law on the set of its rational points, such that the point at infinity serves as the neutral element and inverses are reflections on the  $x$ -axis.

The origin of this law can be understood in a geometric picture and is known as the **chord-and-tangent rule**. In the affine representation of a short Weierstraß curve, the rule can be described in the following way:

- (Point addition) Let  $P, Q \in E(\mathbb{F}) \setminus \{\mathcal{O}\}$  with  $P \neq Q$  be two distinct points on an elliptic curve, that are both not the point at infinity. Then the sum of  $P$  and  $Q$  is defined as follows: Consider the line  $l$  which intersects the curve in  $P$  and  $Q$ . If  $l$  intersects the elliptic curve at a third point  $R'$ , define the sum  $R = P \oplus Q$  of  $P$  and  $Q$  as the reflection of  $R'$  at the  $x$ -axis. If it does not intersect the curve at a third point define the sum to be the point at infinity  $\mathcal{O}$ . It can be shown, that no such chord-line will intersect the curve in more than three points, so addition is not ambiguous.
- (Point doubling) Let  $P \in E(\mathbb{F}) \setminus \{\mathcal{O}\}$  be a point on an elliptic curve, that is not the point at infinity. Then the sum of  $P$  with itself (the doubling) is defined as follows: Consider the line which is tangent to the elliptic curve at  $P$ , if this line intersects the elliptic curve at a second point  $R'$ . The sum  $2P = P + P$  is then the reflection of  $R'$  at the  $x$ -axis. If it does not intersect the curve at a third point define the sum to be the point at infinity  $\mathcal{O}$ . It can be shown, It can be shown, that no such tangent-line will intersect the curve in more than two points, so addition is not ambiguous.
- (Point at infinity) We define the point at infinity  $\mathcal{O}$  as the neutral element of addition, that is we define  $P + \mathcal{O} = P$  for all points  $P \in E(\mathbb{F})$ .

It can be shown that the points of an elliptic curve form a commutative group with respect to the tangent and chord rule, such that  $\mathcal{O}$  acts the neutral element and the inverse of any element  $P \in E(\mathbb{F})$  is the reflection of  $P$  on the  $x$ -axis.

To translate the geometric description into algebraic equations, first observe that for any two given curve points  $(x_1, y_1), (x_2, y_2) \in E(\mathbb{F})$ , it can be shown that the identity  $x_1 = x_2$  implies  $y_2 = \pm y_1$ , which shows that the following rules are a complete description of the affine addition law.

- (Neutral element) Point at infinity  $\mathcal{O}$  is the neutral element.
- (Additive inverse ) The additive inverse of  $\mathcal{O}$  is  $\mathcal{O}$  and for any other curve point  $(x, y) \in E(\mathbb{F}) \setminus \{\mathcal{O}\}$ , the additive inverse is given by  $(x, -y)$ .
- (Addition rule) For any two curve points  $P, Q \in E(\mathbb{F})$  addition is defined by one of the following three cases:
  1. (Adding the neutral element) If  $Q = \mathcal{O}$ , then the sum is defined as  $P \oplus Q = P$ .
  2. (Adding inverse elements) If  $P = (x, y)$  and  $Q = (x, -y)$ , the sum is defined as  $P \oplus Q = \mathcal{O}$ .
  3. (Adding non self-inverse equal points) If  $P = (x, y)$  and  $Q = (x, y)$  with  $y \neq 0$ , the sum  $2P = (x', y')$  is defined by

$$x' = \left( \frac{3x^2 + a}{2y} \right)^2 - 2x \quad , \quad y' = \left( \frac{3x^2 + a}{2y} \right)^2 (x - x') - y$$

4. (Adding non inverse different points) If  $P = (x_1, y_1)$  and  $Q = (x_2, y_2)$  such that  $x_1 \neq x_2$ , the sum  $R = P + Q$  with  $R = (x_3, y_3)$  is defined by

$$x_3 = \left( \frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2 \quad , \quad y_3 = \left( \frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3) - y_1$$

Note that short Weierstraß curve points  $P$  with  $P = (x, 0)$  are inverse to themselves, which implies  $2P = \mathcal{O}$  in this case.

*Notation and Symbols 8.* Let  $\mathbb{F}$  be a field and  $E(\mathbb{F})$  be an elliptic curve over  $\mathbb{F}$ . We write  $\oplus$  for the group law on  $E(\mathbb{F})$  and  $(E(\mathbb{F}), \oplus)$  for the group of rational points.

As we can see, it is very efficient to compute inverses on elliptic curves. However, computing the addition of elliptic curve points in the affine representation needs to consider many cases and involves extensive finite field divisions. As we will see in the next paragraph this can be simplified in projective coordinates.

To get some practical impression of how the group law on an elliptic curve is computed, let's look at some actual cases:

*Example 70.* Consider the elliptic curve  $E_1(\mathbb{F}_5)$  from example XXX again. As we have seen, the set of rational points contains 9 elements and is given by

$$E_1(\mathbb{F}_5) = \{\mathcal{O}, (0, 1), (2, 1), (3, 1), (4, 2), (4, 3), (0, 4), (2, 4), (3, 4)\}$$

We know that this set defines a group, so we can add any two elements from  $E_1(\mathbb{F}_5)$  to get a third element.

To give an example consider the elements  $(0, 1)$  and  $(4, 2)$ . Neither of these elements is the neutral element  $\mathcal{O}$  and since the  $x$ -coordinate of  $(0, 1)$  is different from the  $x$ -coordinate of  $(4, 2)$ , we know that we have to use the chord rule, that is rule number 4 from XXX to compute

add reference

add reference

the sum  $(0, 1) \oplus (4, 2)$ . We get

$$\begin{aligned} x_3 &= \left( \frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2 && \# \text{ insert points} \\ &= \left( \frac{2 - 1}{4 - 0} \right)^2 - 0 - 4 && \# \text{ simplify in } \mathbb{F}_5 \\ &= \left( \frac{1}{4} \right)^2 + 1 = 4^2 + 1 = 1 + 1 = 2 \end{aligned}$$

$$\begin{aligned} y_3 &= \left( \frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3) - y_1 && \# \text{ insert points} \\ &= \left( \frac{2 - 1}{4 - 0} \right) (0 - 2) - 1 && \# \text{ simplify in } \mathbb{F}_5 \\ &= \left( \frac{1}{4} \right) \cdot 3 + 4 = 4 \cdot 3 + 4 = 2 + 4 = 1 \end{aligned}$$

So in our elliptic curve  $E_1(\mathbb{F}_5)$  we get  $(0, 1) \oplus (4, 2) = (2, 1)$  and indeed the pair  $(2, 1)$  is an element of  $E_1(\mathbb{F}_5)$  as expected. On the other hand we have  $(0, 1) \oplus (0, 4) = \mathcal{O}$ , since both points have equal  $x$ -coordinates and inverse  $y$ -coordinates rendering them as inverse to each other. Adding the point  $(4, 2)$  to itself, we have to use the tangent rule, that is rule 3 from XXX. We get

add reference

$$\begin{aligned} x' &= \left( \frac{3x^2 + a}{2y} \right)^2 - 2x && \# \text{ insert points} \\ &= \left( \frac{3 \cdot 4^2 + 1}{2 \cdot 2} \right)^2 - 2 \cdot 4 && \# \text{ simplify in } \mathbb{F}_5 \\ &= \left( \frac{3 \cdot 1 + 1}{4} \right)^2 + 3 \cdot 4 = \left( \frac{4}{4} \right)^2 + 2 = 1 + 2 = 3 \end{aligned}$$

$$\begin{aligned} y' &= \left( \frac{3x^2 + a}{2y} \right)^2 (x - x') - y && \# \text{ insert points} \\ &= \left( \frac{3 \cdot 4^2 + 1}{2 \cdot 2} \right)^2 (4 - 3) - 2 && \# \text{ simplify in } \mathbb{F}_5 \\ &= 1 \cdot 1 + 3 = 4 \end{aligned}$$

537 So in our elliptic curve  $E_1(\mathbb{F}_5)$  we get the doubling  $2 \cdot (4, 2)$ , that is  $(4, 2) \oplus (4, 2) = (3, 4)$  and  
 538 indeed the pair  $(3, 4)$  is an element of  $E_1(\mathbb{F}_5)$  as expected. The group  $E_1(\mathbb{F}_5)$  has no self inverse  
 539 points other than the neutral element  $\mathcal{O}$ , since no point has 0 as its  $y$ -coordinate. We can invoke  
 540 Sage to double check the computations.

```
541 Sage: F5 = GF(5)
542 Sage: E1 = EllipticCurve(F5, [1, 1])
543 Sage: INF = E1(0) # point at infinity
544 Sage: P1 = E1(0, 1)
545 Sage: P2 = E1(4, 2)
```

```

546 Sage: P3 = E1(0,4)
547 Sage: R1 = E1(2,1)
548 Sage: R2 = E1(3,4)
549 Sage: R1 == P1+P2
550 Sage: INF == P1+P3
551 Sage: R2 == P2+P2
552 Sage: R2 == 2*P2
553 Sage: P3 == P3 + INF

```

*Example 71 (Pen-jubjub).* Consider the *PJJ\_13*-curve from example XXX again and recall that its group of rational points is given by

add reference

$$PJJ\_13 = \{\mathcal{O}, (1,2), (1,11), (4,0), (5,2), (5,11), (6,5), (6,8), (7,2), (7,11), (8,5), (8,8), (9,4), (9,9), (10,3), (10,10), (11,6), (11,7), (12,5), (12,8)\}$$

In contrast to the group from the previous example, this group contains a self inverse point, which is different from the neutral element, given by  $(4,0)$ . To see what this means, observe that we can not add  $(4,0)$  to itself using the tangent rule 3 from XXX, as the  $y$ -coordinate is zero. Instead we have to use rule 2, since  $0 = -0$ . We therefore get  $(4,0) \oplus (4,0) = \mathcal{O}$  in *PJJ\_13*. The point  $(4,0)$  is therefore inverse to itself, as adding it to itself gives the neutral element.

add reference

```

560 Sage: F13 = GF(13)
561 Sage: MJJ = EllipticCurve(F13, [8,8])
562 Sage: P = MJJ(4,0)
563 Sage: INF = MJJ(0) # Point at infinity
564 Sage: INF == P+P
565 Sage: INF == 2*P

```

*Example 72.* Consider the *Secp256k1* curve from example XXX again. The following code involves Sage to generate a random affine curve point, we then apply our compression method

add reference

```

568 Sage: P = Secp256k1.random_point()
569 Sage: Q = Secp256k1.random_point()
570 Sage: INF = Secp256k1(0)
571 Sage: R1 = -P
572 Sage: R2 = P + Q
573 Sage: R3 = Secp256k1.order()*P
574 Sage: P.xy()
575 Sage: Q.xy()
576 Sage: (ZZ(R1[0]).str(16), ZZ(R1[1]).str(16))
577 Sage: R2.xy()
578 Sage: R3 == INF
579 Sage: P[1]+R1[1] == Fp(0) # -(x,y) = (x,-y)

```

*Exercise 36.* Consider the *PJJ\_13*-curve from example XXX.

add reference

1. Compute the inverse of  $(10,10)$ ,  $\mathcal{O}$ ,  $(4,0)$  and  $(1,2)$ .
2. Compute the expression  $3 \cdot (1,11) - (9,9)$ .
3. Solve the equation  $x + 2(9,4) = (5,2)$  for some  $x \in PJJ\_13$
4. Solve the equation  $x \cdot (7,11) = (8,5)$  for  $x \in \mathbb{Z}$

**Scalar multiplication** As we have seen in the previous section, elliptic curves  $E(\mathbb{F})$  have the structure of a commutative group associated to them. It can moreover be shown, that this group is finite and cyclic, whenever the field is finite.

To understand the elliptic curve scalar multiplication, recall from XXX that every finite cyclic group of order  $q$  has a generator  $g$  and an associated exponential map  $g^{(\cdot)} : \mathbb{Z}_q \rightarrow \mathbb{G}$ , where  $g^n$  is the  $n$ -fold product of  $g$  with itself.

add reference

Now, elliptic curve scalar multiplication is then nothing but the exponential map, written in additive notation. To be more precise let  $\mathbb{F}$  be a finite field,  $E(\mathbb{F})$  an elliptic curve of order  $r$  and  $P$  a generator of  $E(\mathbb{F})$ . Then the **elliptic curve scalar multiplication** with base  $P$  is given by

$$[\cdot]P : \mathbb{Z}_r \rightarrow E(\mathbb{F}); m \mapsto [m]P$$

where  $[0]P = \mathcal{O}$  and  $[m]P = P + P + \dots + P$  is the  $m$ -fold sum of  $P$  with itself. Elliptic curve scalar multiplication is therefore nothing but an instantiation of the general exponential map, when using additive instead of multiplicative notation. This map is a homomorph of groups, which means that  $[n + m]P = [n]P \oplus [m]P$ .

As with all finite, cyclic groups the inverse of the exponential map exist and is usually called the **elliptic curve discrete logarithm map**. However, elliptic curve are believed to be XXX-groups, which means that we don't know of any efficient way to actually compute this map.

add reference

Scalar multiplication and its inverse, the elliptic curve discrete logarithm, define the elliptic curve discrete logarithm **problem**, which consists of finding solutions  $m \in \mathbb{Z}_r$ , such that

$$P = [m]Q \tag{5.2}$$

holds. Any solution  $m$  is usually called a **discrete logarithm** relation between  $P$  and  $Q$ . If  $Q$  is a generator of the curve, then there is a discrete logarithm relation between  $Q$  and any other point, since  $Q$  generates the group by repeatedly adding  $Q$  to itself. So for generator  $Q$  and point  $P$ , we know some discrete logarithm relation exist. However, since elliptic curves are believed to be XXX-groups, finding actual relations  $m$  is computationally hard, with runtimes approximately in the size of the order of the group. In practice, we often need the assumption that a discrete logarithm relation exists, but that at the same time no-one knows this relation.

add reference

One useful property of the exponential map in regard to the examples in this book, is that it can be used to greatly simplify pen-and-paper computations. As we have seen in example XXX, computing the elliptic curve addition law takes quit a bit of effort, when done without a computer. However, when  $g$  is a generator of small pen-and-paper elliptic curve group of order  $r$ , we can use the exponential map to write the group as

add reference

$$\mathbb{G} = \{[1]g \rightarrow [2]g \rightarrow [3]g \rightarrow \dots \rightarrow [r-1]g \rightarrow \mathcal{O}\} \tag{5.3}$$

using cofactor clearing, which implies that  $[r]g = \mathcal{O}$ . "Logarithmic ordering" like this greatly simplifies complicated elliptic curve addition to the much simpler case of modular  $r$  addition. So in order to add two curve points  $P$  and  $Q$ , we only have to look up their discrete log relations with the generator, say  $P = [n]g$  and  $Q = [m]g$  and compute the sum as  $P \oplus Q = [n + m]g$ . This is, of course, only possible for small groups which we can organize as in XXX.

add reference

In the following example we will look at some implications of the fact that elliptic curves are finite cyclic groups. We will apply the fundamental theorem of finite cyclic groups and look how it reflects on the curves in consideration.

*Example 73.* Consider the elliptic curve group  $E_1(\mathbb{F}_5)$  from example XXX. Since it is a finite cyclic group of order 9 and the prime factorization of 9 is  $3 \cdot 3$ , we can use the fundamental

add reference

theorem of finite cyclic groups to reason about all its subgroups. In fact since the only prime factor of 9 is 3, we know that  $E_1(\mathbb{F}_5)$  has the following subgroups:

- $\mathbb{G}_1 = E_1(\mathbb{F}_5)$  is a subgroup of order 9. By definition any group is a subgroup of itself.
- $\mathbb{G}_2 = \{(2, 1), (2, 4), \mathcal{O}\}$  is a subgroup of order 3. This is the subgroup associated to the prime factor 3.
- $\mathbb{G}_3 = \{\mathcal{O}\}$  is a subgroup of order 1. This is the trivial subgroup.

Moreover since  $E_1(\mathbb{F}_5)$  and all its subgroups are cyclic, we know from XXX, that they must have generators. For example the curve point  $(2, 1)$  is a generator of the order 3-subgroup  $\mathbb{G}_2$ , since every element of  $\mathbb{G}_2$  can be generated, by repeatedly adding  $(2, 1)$  to itself:

$$\begin{aligned}[1](2, 1) &= (2, 1) \\ [2](2, 1) &= (2, 4) \\ [3](2, 1) &= \mathcal{O}\end{aligned}$$

Since  $(2, 1)$  is a generator we know from XXX, that it gives rise to an exponential map from the finite field  $\mathbb{F}_3$  onto  $\mathbb{G}_2$  defined by scalar multiplication

$$[\cdot](2, 1) : \mathbb{F}_3 \rightarrow \mathbb{G}_2 : x \mapsto [x](2, 1)$$

To give an example of a generator that generates the entire group  $E_1(\mathbb{F}_5)$  consider the point  $(0, 1)$ . Applying the tangent rule repeatedly we compute with some effort:

$$\begin{array}{ll} [0](0, 1) = \mathcal{O} & [1](0, 1) = (0, 1) \\ [2](0, 1) = (4, 2) & [3](0, 1) = (2, 1) \\ [4](0, 1) = (3, 4) & [5](0, 1) = (3, 1) \\ [6](0, 1) = (2, 4) & [7](0, 1) = (4, 3) \\ [8](0, 1) = (0, 4) & [9](0, 1) = \mathcal{O} \end{array}$$

Again, since  $(2, 1)$  is a generator we know from XXX, that it gives rise to an exponential map. However, since the group order is not a prime number, the exponential maps, does not map a from any field but from the residue class ring  $\mathbb{Z}_9$  only:

$$[\cdot](0, 1) : \mathbb{Z}_9 \rightarrow \mathbb{G}_1 : x \mapsto [x](0, 1)$$

Using the generator  $(0, 1)$  and its associated exponential map, we can write  $E(\mathbb{F}_1)$  i logarithmic order with respect to  $(0, 1)$  as explained in XXX. We get

$$E_1(\mathbb{F}_5) = \{(0, 1) \rightarrow (4, 2) \rightarrow (2, 1) \rightarrow (3, 4) \rightarrow (3, 1) \rightarrow (2, 4) \rightarrow (4, 3) \rightarrow (0, 4) \rightarrow \mathcal{O}\}$$

indicating that the first element is a generator and the  $n$ -th element is the scalar product of  $n$  and the generator. To see how logarithmic orders like this simplify the computations in small elliptic curve groups, consider example XXX again. In that example we use the chord and tangent rule to compute  $(0, 1) \oplus (4, 2)$ . Now in the logarithmic order of  $E_1(\mathbb{F})$  we can compute that sum much easier, since we can directly see that  $(0, 1) = [1](0, 1)$  and  $(4, 2) = [2](0, 1)$ . We can then deduce  $(0, 1) \oplus (4, 2) = (2, 1)$  immediately, since  $[1](0, 1) \oplus [2](0, 1) = [3](0, 1) = (2, 1)$ .

To give another example, we can immediately see that  $(3, 4) \oplus (4, 3) = (4, 2)$ , without doing any expensive elliptic curve addition, since we know  $(3, 4) = [4](0, 1)$  as well as  $(4, 3) =$

[7](0, 1) from the logarithmic representation of  $E_1(\mathbb{F}_5)$  and since  $4 + 7 = 2$  in  $\mathbb{Z}_9$ , the result must be  $[2](0, 1) = (4, 2)$ .

Finally we can use  $E_1(\mathbb{F}_5)$  as an example to understand the concept of cofactor clearing from XXX. Since the order of  $E_1(\mathbb{F}_5)$  is 9 we only have a single factor, which happen to be the cofactor as well. Cofactor clearing then implies that we can map any element from  $E_1(\mathbb{F}_5)$  onto its prime factor group  $\mathbb{G}_2$  by scalar multiplication with 3. For example taking the element  $(3, 4)$  which is not in  $\mathbb{G}_2$  and multiplying it with 3, we get  $[3](3, 4) = (2, 1)$ , which is an element of  $\mathbb{G}_2$  as expected.

add reference

In the following example we will look at the subgroups of our pen-jubjub curve, define generators and compute the logarithmic order for pen-and-paper computations. Then we have another look at the principle of cofactor clearing.

*Example 74.* Consider the pen-jubjub curve  $PJJ\_13$  from example XXX again. Since the order of  $PJJ\_13$  is 20 and the prime factorization of 20 is  $2^2 \cdot 5$ , we know that the  $PJJ\_13$  contains a "large" prime order subgroup of size 5 and a small prime order subgroup of size 2.

add reference

To compute those groups, we can apply the technique of cofactor clearing in a try and repeat loop. We start the loop by arbitrarily choose an element  $P \in PJJ\_13$ . Then we multiply that element with the cofactor of the group, we want to compute. If the result is  $\mathcal{O}$ , we try a different element and repeat the process until the result is different from the point at infinity.

To compute a generator for the small prime order subgroup  $(PJJ\_13)_2$ , first observe that the cofactor is 10, since  $20 = 2 \cdot 10$ . We then arbitrarily choose the curve point  $(5, 11) \in PJJ\_13$  and compute  $[10](5, 11) = \mathcal{O}$ . Since the result is the point at infinity, we have to try another curve point, say  $(9, 4)$ . We get  $[10](9, 4) = (4, 0)$  and we can deduce that  $(4, 0)$  is a generator of  $(PJJ\_13)_2$ . Logarithmic order of then gives

$$(PJJ\_13)_2 = \{(4, 0) \rightarrow \mathcal{O}\}$$

as expected, since we know from example XXX that  $(4, 0)$  is self inverse, with  $(4, 0) \oplus (4, 0) = \mathcal{O}$ . Double checking the computations using Sage:

add reference

```
Sage: F13 = GF(13)
Sage: PJJ = EllipticCurve(F13, [8, 8])
Sage: P = PJJ(5, 11)
Sage: INF = PJJ(0)
Sage: 10*P == INF
Sage: Q = PJJ(9, 4)
Sage: R = PJJ(4, 0)
Sage: 10*Q == R
```

We can apply the same reasoning to the "large" prime order subgroup  $(PJJ\_13)_5$ , which contains 5 elements. To compute a generator for this group, first observe that the associated cofactor is 4, since  $20 = 5 \cdot 4$ . We choose the curve point  $(9, 4) \in PJJ\_13$  again and compute  $[4](9, 4) = (7, 11)$  and we can deduce that  $(7, 11)$  is a generator of  $(PJJ\_13)_5$ . Using the generator  $(7, 11)$ , we compute the exponential map  $[\cdot](7, 11) : \mathbb{F}_5 \rightarrow PJJ\_13$  and get

$$\begin{aligned} [0](7, 11) &= \mathcal{O} \\ [1](7, 11) &= (7, 11) \\ [2](7, 11) &= (8, 5) \\ [3](7, 11) &= (8, 8) \\ [4](7, 11) &= (7, 2) \end{aligned}$$



We can use this computation to write the large order prime group  $(PJJ\_13)_5$  of the pen-jubjub curve in logarithmic order, which we will use quite frequently in what follows. We get:

$$(PJJ\_13)_5 = \{(7, 11) \rightarrow (8, 5) \rightarrow (8, 8) \rightarrow (7, 2) \rightarrow \mathcal{O}\}$$

From this, we can immediately see that for example  $(8, 8) \oplus (7, 2) = (8, 5)$ , since  $3 + 4 = 2$  in  $\mathbb{F}_5$ .

From the previous two examples, the reader might get the impression, that elliptic curve computation can be largely replaced by modular arithmetics. This however, is not true in general, but only an arefact of small groups where it is possible to write the entire group in a logarithmic order. The following example gives some understanding, why this is not possible in cryptographically secure groups

*Example 75.* SEKTP BICOIN. DISCRET LOG HARDNESS PROHIBITS ADDITION IN THE FIELD...

**Projective short Weierstraß form** As we have seen in the previous section, describing elliptic curves as pairs of points that satisfy a certain equation is relatively straight forward. However, in order to define a group structure on the set of points, we had to add a special point at infinity to act as the neutral element.

Recalling from the definition of projective planes XXX we know, that points at infinity are handled as ordinary points in projective geometry. It make therefore sense to look at the definition of a short Weierstraß curve in projective geometry.

To see what a short Weierstraß curve in projective coordinates is, let  $\mathbb{F}$  be a finite field of order  $q$  and characteristic  $> 3$ ,  $a, b \in \mathbb{F}$  two field elements such that  $4a^3 + 27b^2 \bmod q \neq 0$  and  $\mathbb{FP}^2$  the projective plane over  $\mathbb{F}$ . Then a **short Weierstraß elliptic curve** over  $\mathbb{F}$  in its projective representation is the set

$$E(\mathbb{FP}^2) = \{[X : Y : Z] \in \mathbb{FP}^2 \mid Y^2 \cdot Z = X^3 + a \cdot X \cdot Z^2 + b \cdot Z^3\} \quad (5.4)$$

of all points  $[X : Y : Z] \in \mathbb{FP}^2$  from the projective plane, that satisfy the **homogenous** cubic equation  $Y^2 \cdot Z = X^3 + a \cdot X \cdot Z^2 + b \cdot Z^3$ .

To understand how the point at infinity is unified in this definition, recall from XXX that, in projective geometry points at infinity are given by homogeneous coordinates  $[X : Y : 0]$ . Inserting representatives  $(x_1, y_1, 0) \in [X : Y : 0]$  from those classes into the defining homogenous cubic equations gives

$$\begin{aligned} y_1^2 \cdot 0 &= x_1^3 + a \cdot x_1 \cdot 0^2 + b \cdot 0^3 \\ 0 &= x_1^3 \end{aligned} \quad \Leftrightarrow$$

which shows that the only point at infinity that is also a point on a projective short Weierstraß curve is the class

$$[0, 1, 0] = \{(0, y, 0) \mid y \in \mathbb{F}\}$$

This point is the projective representation of  $\mathcal{O}$ . The projective representation of a short Weierstraß curve therefore has the advantage to not need a special symbol to represent the point at infinity  $\mathcal{O}$  from the affine definition.

*Example 76.* To get an intuition of how an elliptic curve in projective geometry looks, consider curve  $E_1(\mathbb{F}_5)$  from example (XXX). We know that in its affine representation, the set of rational



points is given by

$$E_1(\mathbb{F}_5) = \{\mathcal{O}, (0, 1), (2, 1), (3, 1), (4, 2), (4, 3), (0, 4), (2, 4), (3, 4)\}$$

690 which is defined as the set of all pairs  $(x, y) \in \mathbb{F}_5 \times \mathbb{F}_5$ , such that the affine short Weierstraß  
691 equation  $y^2 = x^3 + ax + b$  with  $a = 1$  and  $b = 1$  is satisfied.

To find the projective representation of a short Weierstraß curve with the same parameters  $a = 1$  and  $b = 1$ , we have to compute the set of projective points  $[X : Y : Z]$  from the projective plane  $\mathbb{F}_5\mathbb{P}^2$ , that satisfy the homogenous cubic equation

$$y_1^2 z_1 = x_1^3 + 1 \cdot x_1 z_1^2 + 1 \cdot z_1^3$$

692 for any representative  $(x_1, y_1, z_1) \in [X : Y : Z]$ . We know from XXX, that the projective plane  
693  $\mathbb{F}_5\mathbb{P}^2$  contains  $5^2 + 5 + 1 = 31$  elements, so we can take the effort and insert all elements into  
694 equation XXX and see if both sides match.

For example, consider the projective point  $[0 : 4 : 1]$ . We know from XXX, that this point in the projective plane represents the line

$$[0 : 4 : 1] = \{(0, 0, 0), (0, 4, 1), (0, 3, 2), (0, 2, 3), (0, 1, 4)\}$$

in the three dimensional space  $\mathbb{F}^3$ . To check whether or not  $[0 : 4 : 1]$  satisfies XXX, we can insert any representative, that is we can insert any element from XXX. Each element satisfies the equation if and only if any other satisfies the equation. So we insert  $(0, 4, 1)$  and get

$$1^2 \cdot 1 = 0^3 + 1 \cdot 0 \cdot 1^2 + 1 \cdot 1^3$$

which tells us that the affine point  $[0 : 4 : 1]$  is indeed a solution. And as we can see, would just as well insert any other representative. For example inserting  $(0, 3, 2)$  also satisfies XXX, since

$$3^2 \cdot 2 = 0^3 + 1 \cdot 0 \cdot 2^2 + 1 \cdot 2^3$$

695 To find the projective representation of  $E_1$ , we first observe that the projective line at infinity  
696  $[1 : 0 : 0]$  is not a curve point on any projective short Weierstraß curve since it can not satisfy  
697 XXX for any parameter  $a$  and  $b$ . So we can exclude it from our consideration.

698 Moreover a point at infinity  $[X : Y : 0]$  can only satisfy equation XXX for any  $a$  and  $b$ , if  
699  $X = 0$ , which implies that the only point at infinity relevant for short Weierstraß elliptic curves  
700 is  $[0 : 1 : 0]$ , since  $[0 : k : 0] = [0 : 1 : 0]$  for all  $k$  from the finite field. So we can exclude all points  
701 at infinity except the point  $[0 : 1 : 0]$ .

So all points that remain are the affine points  $[X : Y : 1]$ . Inserting all of them into XXX we get the set of all projective curve points as

$$E_1(\mathbb{F}_5\mathbb{P}^2) = \{[0 : 1 : 0], [0 : 1 : 1], [2 : 1 : 1], [3 : 1 : 1], [4 : 2 : 1], [4 : 3 : 1], [0 : 4 : 1], [2 : 4 : 1], [3 : 4 : 1]\}$$

702 If we compare this with the affine representation we see that there is a 1:1 correspondence  
703 between the points in the affine representation XXX and the affine points in projective geometry  
704 and that the point  $[0 : 1 : 0]$  represents the additional point  $\mathcal{O}$  in the projective representation.

705 *Exercise 37.* Compute the projective representation of the pen-jubjub curve and the logarithmic  
706 order of its large prime order subgroup with respect to the generator  $(7, 11)$ .

**Projective Group law** As we have seen in XXX, one of the key properties of an elliptic curve is that it comes with a definition of a group law on the set of its rational points, described geometrically by the chord and tangent rule. This rule was kind of intuitive, with the exception of the distinguished point at infinity, which appeared whenever the chord or the tangent did not have a third intersection point with the curve.

add reference

One of the key features of projective coordinates is now, that in projective space it is guaranteed that any chord will always intersect the curve in three points and any tangent will intersect in two points including the tangent point. So the geometric picture simplifies as we don't need to consider external symbols and associated cases.

Again, it can be shown that the points of an elliptic curve in projective space form a commutative group with respect to the tangent and chord rule, such that the projective point  $[0 : 1 : 0]$  is the neutral element and the additive inverse of a point  $[X : Y : Z]$  is given by  $[X : -Y : Z]$ . The addition law is then usually described by the following algorithm, that minimizes the number of needed additions and multiplications in the base field.

*Exercise 38.* Compare that affine addition law for short Weierstraß curves with the projective addition rule. Which branch in the projective rule corresponds to which case in the affine law?

**Coordinate Transformations** As we have seen in example XXX, there was a close relation between the affine and the projective representation of a short Weierstraß curve. This was no accident. In fact from a mathematical point of view projective and affine short Weierstraß curves describe the same thing as there is a one-to-one correspondence (an isomorphism) between both representations for any given parameters  $a$  and  $b$ .

add reference

To specify the isomorphism, let  $E(\mathbb{F})$  and  $E(\mathbb{F}\mathbb{P}^2)$  be an affine and a projective short Weierstraß curve defined for the same parameters  $a$  and  $b$ . Then the map

$$\Phi : E(\mathbb{F}) \rightarrow E(\mathbb{F}\mathbb{P}^2) : \begin{array}{ll} (x, y) & \mapsto [x : y : 1] \\ \mathcal{O} & \mapsto [0 : 1 : 0] \end{array} \quad (5.5)$$

maps points from the affine representation to points from the projective representation of a short Weierstraß curve, that is if the pair of points  $(x, y)$  satisfies the affine equation  $y^2 = x^3 + ax + b$ , then all homogeneous coordinates  $(x_1, y_1, z_1) \in [x : y : 1]$  satisfy the projective equation  $y_1^2 \cdot z_1 = x_1^3 + ay_1 \cdot z_1^2 + b \cdot z_1^3$ . The inverse is given by the map

$$\Phi^{-1} : E(\mathbb{F}\mathbb{P}^2) \rightarrow E(\mathbb{F}) : [X : Y : Z] \mapsto \begin{cases} (\frac{X}{Z}, \frac{Y}{Z}) & \text{if } Z \neq 0 \\ \mathcal{O} & \text{if } Z = 0 \end{cases} \quad (5.6)$$

Note the only projective point  $[X : Y : Z]$  with  $Z \neq 0$  that satisfies XXX is given by the class  $[0 : 1 : 0]$ .

add reference

One key feature of  $\Phi$  and its inverse is, that it respects the group structure, which means that  $\Phi((x_1, y_1) \oplus (x_2, y_2))$  is equal to  $\Phi(x_1, y_1) \oplus \Phi(x_2, y_2)$ . The same holds true for the inverse map  $\Phi^{-1}$ .

Maps with these properties are called **group isomorphisms** and from a mathematical point of view the existence of  $\Phi$  implies, that both definition are equivalent and implementations can choose freely between both representations.

## 5.1.2 Montgomery Curves

History and use of them (optimized scalar multiplication)

**Algorithm 6** Projective Weierstraß Addition Law

---

**Require:**  $[X_1 : Y_1 : Z_1], [X_2 : Y_2 : Z_2] \in E(\mathbb{F}_{\mathbb{P}}^2)$

**procedure** ADD-RULE( $[X_1 : Y_1 : Z_1], [X_2 : Y_2 : Z_2]$ )

**if**  $[X_1 : Y_1 : Z_1] == [0 : 1 : 0]$  **then**

$[X_3 : Y_3 : Z_3] \leftarrow [X_2 : Y_2 : Z_2]$

**else if**  $[X_2 : Y_2 : Z_2] == [0 : 1 : 0]$  **then**

$[X_3 : Y_3 : Z_3] \leftarrow [X_1 : Y_1 : Z_1]$

**else**

$U_1 \leftarrow Y_2 \cdot Z_1$

$U_2 \leftarrow Y_1 \cdot Z_2$

$V_1 \leftarrow X_2 \cdot Z_1$

$V_2 \leftarrow X_1 \cdot Z_2$

**if**  $V_1 == V_2$  **then**

**if**  $U_1 \neq U_2$  **then**  $[X_3 : Y_3 : Z_3] \leftarrow [0 : 1 : 0]$

**else**

**if**  $Y_1 == 0$  **then**  $[X_3 : Y_3 : Z_3] \leftarrow [0 : 1 : 0]$

**else**

$W \leftarrow a \cdot Z_1^2 + 3 \cdot X_1^2$

$S \leftarrow Y_1 \cdot Z_1$

$B \leftarrow X_1 \cdot Y_1 \cdot S$

$H \leftarrow W^2 - 8 \cdot B$

$X' \leftarrow 2 \cdot H \cdot S$

$Y' \leftarrow W \cdot (4 \cdot B - H) - 8 \cdot Y_1^2 \cdot S^2$

$Z' \leftarrow 8 \cdot S^3$

$[X_3 : Y_3 : Z_3] \leftarrow [X' : Y' : Z']$

**end if**

**end if**

**else**

$U = U_1 - U_2$

$V = V_1 - V_2$

$W = Z_1 \cdot Z_2$

$A = U^2 \cdot W - V^3 - 2 \cdot V^2 \cdot V_2$

$X' = V \cdot A$

$Y' = U \cdot (V^2 \cdot V_2 - A) - V^3 \cdot U_2$

$Z' = V^3 \cdot W$

$[X_3 : Y_3 : Z_3] \leftarrow [X' : Y' : Z']$

**end if**

**end if**

**return**  $[X_3 : Y_3 : Z_3]$

**end procedure**

**Ensure:**  $[X_3 : Y_3 : Z_3] == [X_1 : Y_1 : Z_1] \oplus [X_2 : Y_2 : Z_2]$

---

**Affine Montgomery Form** To see what a Montgomery curve in affine coordinates is, let  $\mathbb{F}$  be a finite field of characteristic  $> 2$  and  $A, B \in \mathbb{F}$  two field elements such that  $B \neq 0$  and  $A^2 \neq 4$ . Then a **Montgomery elliptic curve**  $M(\mathbb{F})$  over  $\mathbb{F}$  in its affine representation is the set

$$M(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid B \cdot y^2 = x^3 + A \cdot x^2 + x\} \cup \{\mathcal{O}\} \quad (5.7)$$

of all pairs of field elements  $(x, y) \in \mathbb{F} \times \mathbb{F}$ , that satisfy the Montgomery cubic equation  $B \cdot y^2 = x^3 + A \cdot x^2 + x$ , together with a distinguished symbol  $\mathcal{O}$ , called the **point at infinity**.

Despite the fact that Montgomery curves look different than short Weierstraß curve, they are in fact just a special way to describe certain short Weierstraß curves. In fact every curve in affine Montgomery form can be transformed into an elliptic curve in Weierstraß form. To see that assume that a curve in Montgomery form  $By^2 = x^3 + Ax^2 + x$  is given. The associated Weierstraß form is then

$$y^2 = x^3 + \frac{3 - A^2}{3B^2} \cdot x + \frac{2A^3 - 9A}{27B^3}$$

On the other hand, an elliptic curve  $E(\mathbb{F})$  over base field  $\mathbb{F}$  in Weierstraß form  $y^2 = x^3 + ax + b$  can be converted to Montgomery form if and only if the following conditions hold:

- The number of points on  $E(\mathbb{F})$  is divisible by 4
- The polynomial  $z^3 + az + b \in \mathbb{F}[z]$  has at least one root  $z_0 \in \mathbb{F}$
- $3z_0^2 + a$  is a quadratic residue in  $\mathbb{F}$ .

When these conditions are satisfied, then for  $s = (\sqrt{3z_0^2 + a})^{-1}$  the equivalent Montgomery curve is defined by the equation

$$sy^2 = x^3 + (3z_0s)x^2 + x$$

If those properties are met it is therefore possible to transform certain Weierstraß curve into Montgomery form. In the following example we will look at our pen-jubjub curve again and show that it is actually a Montgomery curve.

*Example 77.* Consider the prime field  $\mathbb{F}_{13}$  and the pen-jubjub curve  $PJJ\_13$  from example XXX. To see that it is a Montgomery curve, we have to check the properties from XXX:

Since the order of  $PJJ\_13$  is 20, which is divisible by 4, the first requirement is met. Next, since  $a = 8$  and  $b = 8$ , we have to check if the polynomial  $P(z) = z^3 + 8z + 8$  has a root in  $\mathbb{F}_{13}$ . We simply evaluate  $P$  at all numbers  $z \in \mathbb{F}_{13}$  and find that  $P(4) = 0$ , so a root is given by  $z_0 = 4$ . In a last step we have to check, that  $3 \cdot z_0^2 + a$  has a root in  $\mathbb{F}_{13}$ . We compute

$$\begin{aligned} 3z_0^2 + a &= 3 \cdot 4^2 + 8 \\ &= 3 \cdot 3 + 8 \\ &= 9 + 8 \\ &= 4 \end{aligned}$$

To see if 4 is a quadratic residue, we can use Euler's criterion XXX to compute the Legendre symbol of 4. We get:

$$\left( \frac{4}{13} \right) = 4^{\frac{13-1}{2}} = 4^6 = 1$$

so 4 indeed has a root in  $\mathbb{F}_{13}$ . In fact computing a root of 4 in  $\mathbb{F}_{13}$  is easy, since the integer root 2 of 4 is also one of its roots in  $\mathbb{F}_{13}$ . The other root is given by  $13 - 4 = 9$ .

add reference

add reference

add reference

Now since all requirements are met, we have shown that  $PJJ\_13$  is indeed a Montgomery curve and we can use XXX to compute its associated Montgomery form. We compute

add reference

$$\begin{aligned}
 s &= \left( \sqrt{3 \cdot z_0^2 + 8} \right)^{-1} \\
 &= 2^{-1} && \# \text{ Fermat's little theorem} \\
 &= 2^{13-2} && \# 2048 \bmod 13 = 7 \\
 &= 7
 \end{aligned}$$

The defining equation for the Montgomery form of our pen-jubjub curve is then given by the following equation

$$\begin{aligned}
 sy^2 &= x^3 + (3z_0s)x^2 + x && \Rightarrow \\
 7 \cdot y^2 &= x^3 + (3 \cdot 4 \cdot 7)x^2 + x && \Leftrightarrow \\
 7 \cdot y^2 &= x^3 + 6x^2 + x
 \end{aligned}$$

So we get the defining parameters as  $B = 7$  and  $A = 6$  and we can write the pen-jubjub curve in its affine Montgomery representation as

$$PJJ\_13 = \{(x, y) \in \mathbb{F}_{13} \times \mathbb{F}_{13} \mid 7 \cdot y^2 = x^3 + 6x^2 + x\} \cup \{\mathcal{O}\}$$

Now that we have the abstract definition of our pen-jubjub curve in Montgomery form, we can compute the set of points, by inserting all pairs  $(x, y) \in \mathbb{F}_{13} \times \mathbb{F}_{13}$  similar to how we computed the curve points in its Weierstraß representation. We get

$$\begin{aligned}
 PJJ\_13 = \{ &\mathcal{O}, (0, 0), (1, 4), (1, 9), (2, 4), (2, 9), (3, 5), (3, 8), (4, 4), (4, 9), \\
 &(5, 1), (5, 12), (7, 1), (7, 12), (8, 1), (8, 12), (9, 2), (9, 11), (10, 3), (10, 10) \}
 \end{aligned}$$

```

761 Sage: F13 = GF(13)
762 Sage: L_MPJJ = []
763 .....: for x in F13:
764 .....:     for y in F13:
765 .....:         if F13(7)*y^2 == x^3 + F13(6)*x^2 + x:
766 .....:             L_MPJJ.append((x, y))
767 Sage: MPJJ = Set(L_MPJJ)
768 Sage: # does not compute the point at infinity

```

**Affine Montgomery coordinate transformation** Comparing the Montgomery representation of the previous example with the Weierstraß representation of the same curve, we see that there is a 1:1 correspondence between the curve points in both examples. This is no accident. In fact if  $M_{A,B}$  is a Montgomery curve and  $E_{a,b}$  a Weierstraß curve with  $a = \frac{3-A^2}{3B^2}$  and  $b = \frac{2A^2-9A}{27B^3}$  then the function

$$\Phi : M_{A,B} \rightarrow E_{a,b} : (x, y) \mapsto \left( \frac{3x+A}{3B}, \frac{y}{B} \right) \quad (5.8)$$

maps all points in Montgomery representation onto the points in Weierstraß representation. This map is a 1:1 correspondence (an isomorphism) and its inverse map is given by

$$\Phi^{-1} : E_{a,b} \rightarrow M_{A,B} : (x, y) \mapsto (s \cdot (x - z_0), s \cdot y) \quad (5.9)$$

where  $z_0$  is a root of the polynomial  $z^3 + az + b \in \mathbb{F}[z]$  and  $s = (\sqrt{3z_0^2 + a})^{-1}$ . Using this map, it is therefore possible for implementations of Montgomery curves to freely transit between the Weierstraß and the Montgomery representation. Note however, that according to XXX not every Weierstraß curve is a Montgomery curve, as all of the properties from XXX have to be satisfied. The map  $\Phi^{-1}$  therefore does not always exist.

*Example 78.* Consider our pen-jubjub curve again. In example XXX we derive its Weierstraß representation and in example XXX we derive its Montgomery representation.

To see how the coordinate transformation  $\Phi$  works in this example, let's map points from the Montgomery representation onto points from the Weierstraß representation. Inserting for example the point  $(0,0)$  from the Montgomery representation XXX into  $\Phi$  gives

$$\begin{aligned}\Phi(0,0) &= \left( \frac{3 \cdot 0 + A}{3B}, \frac{0}{B} \right) \\ &= \left( \frac{3 \cdot 0 + 6}{3 \cdot 7}, \frac{0}{7} \right) \\ &= \left( \frac{6}{8}, 0 \right) \\ &= (4,0)\end{aligned}$$

So the Montgomery point  $(0,0)$  maps to the self inverse point  $(4,0)$  of the Weierstraß representation. On the other hand we can use our computations of  $s = 7$  and  $z_0 = 4$  from XXX, to compute the inverse map  $\Phi^{-1}$ , which maps point on the Weierstraß representation to points on the Montgomery form. Inserting for example  $(4,0)$  we get

$$\begin{aligned}\Phi^{-1}(4,0) &= (s \cdot (4 - z_0), s \cdot 0) \\ &= (7 \cdot (4 - 4), 0) \\ &= (0,0)\end{aligned}$$

So as expected, the inverse map maps the Weierstraß point back to where it came from on the Montgomery form. We can invoke Sage to proof that our computation of  $\Phi$  is correct:

```
Sage: # Compute PHI of Montgomery form:
Sage: L_PHI_MPJJ = []
Sage: for (x,y) in L_MPJJ: # LMJJ as defined previously
.....:     v = (F13(3)*x + F13(6)) / (F13(3)*F13(7))
.....:     w = y/F13(7)
.....:     L_PHI_MPJJ.append((v,w))
Sage: PHI_MPJJ = Set(L_PHI_MPJJ)
Sage: # Computation Weierstraß form
Sage: C_WPJJ = EllipticCurve(F13,[8,8])
Sage: L_WPJJ = [P.xy() for P in C_WPJJ.points() if P.order() > 1]
Sage: WPJJ = Set(L_WPJJ)
Sage: # check PHI(Montgomery) == Weierstraß
Sage: WPJJ == PHI_MPJJ
Sage: # check the inverse map PHI^(-1)
Sage: L_PHIINV_WPJJ = []
Sage: for (v,w) in L_WPJJ:
```

```

801 .....:      x = F13(7) * (v-F13(4))
802 .....:      y = F13(7) * w
803 .....:      L_PHIINV_WPJJ.append((x, y))
804 Sage: PHIINV_WPJJ = Set(L_PHIINV_WPJJ)
805 Sage: MPJJ == PHIINV_WPJJ

```

**Montgomery group law** So we see that Montgomery curves a special cases of short Weierstraß curves. As such they have a group structure defined on the set of their points, which can also be derived from a chord and tangent rule. In accordance with short Weierstraß curves, it can be shown that the identity  $x_1 = x_2$  implies  $y_2 = \pm y_1$ , which shows that the following rules are a complete description of the affine addition law.

- (Neutral element) Point at infinity  $\mathcal{O}$  is the neutral element.
- (Additive inverse ) The additive inverse of  $\mathcal{O}$  is  $\mathcal{O}$  and for any other curve point  $(x, y) \in M(\mathbb{F}_q) \setminus \{\mathcal{O}\}$ , the additive inverse is given by  $(x, -y)$ .
- (Addition rule) For any two curve points  $P, Q \in M(\mathbb{F}_q)$  addition is defined by one of the following cases:
  1. (Adding the neutral element) If  $Q = \mathcal{O}$ , then the sum is defined as  $P + Q = P$ .
  2. (Adding inverse elements) If  $P = (x, y)$  and  $Q = (x, -y)$ , the sum is defined as  $P + Q = \mathcal{O}$ .
  3. (Adding non self-inverse equal points) If  $P = (x, y)$  and  $Q = (x, y)$  with  $y \neq 0$ , the sum  $2P = (x', y')$  is defined by

$$x' = \left( \frac{3x_1^2 + 2Ax_1 + 1}{2By_1} \right)^2 \cdot B - (x_1 + x_2) - A, \quad y' = \frac{3x_1^2 + 2Ax_1 + 1}{2By_1} (x_1 - x') - y_1$$

4. (Adding non inverse differen points) If  $P = (x_1, y_1)$  and  $Q = (x_2, y_2)$  such that  $x_1 \neq x_2$ , the sum  $R = P + Q$  with  $R = (x_3, y_3)$  is defined by

$$x' = \left( \frac{y_2 - y_1}{x_2 - x_1} \right)^2 B - (x_1 + x_2) - A, \quad y' = \frac{y_2 - y_1}{x_2 - x_1} (x_1 - x') - y_1$$

### 5.1.3 Twisted Edwards Curves

As we have seen in XXX both Weierstraß and Montgomery curves have somewhat complicated addition and doubling laws as many cases have to be distinguished. Those cases translate to branches in computer programs.

In the context of SNARK development two computational models for bounded computations, called **circuits** and **rank-1 constraint systems**, are used and program-branches are undesirably costly, when implemented in those models. It is therefore advantageous to look for curves with an addition/doubling rule, that requires no branches and as few field operations as possible.

Twisted Edwards curves are particular useful here as a subclass of these curves has a compact and easy to implement addition law that works for all point, including the point at infinity. Implementing that rule therefore needs no branching.

add reference

**Twisted Edwards Form** To see what an affine **twisted Edwards curve** looks like, let  $\mathbb{F}$  be a finite field of characteristic  $> 2$  and  $a, d \in \mathbb{F} \setminus \{0\}$  two non zero field elements with  $a \neq d$ . Then a **twisted Edwards elliptic curve** in its affine representation is the set

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid a \cdot x^2 + y^2 = 1 + d \cdot x^2 y^2\} \quad (5.10)$$

of all pairs  $(x, y)$  from  $\mathbb{F} \times \mathbb{F}$ , that satisfy the twisted Edwards equation  $a \cdot x^2 + y^2 = 1 + d \cdot x^2 y^2$ . A twisted Edwards curve is called an Edwards curve (non twisted), if the parameter  $a$  is equal to 1 and is called a **SNARK-friendly** twisted Edwards curve if the parameter  $a$  is a quadratic residue and the parameter  $d$  is a quadratic non residue.

As we can see from the definition, affine twisted Edwards curve look somewhat different from Weierstraß curves as their affine representation does not need a special symbol to represent the point at infinity. In fact we will see that the pair  $(0, 1)$  is always a point on any twisted Edwards curve and that it takes the role of the point at infinity.

Despite the different looks however, twisted Edwards curves are equivalent to Montgomery curves in the sense that for every twisted Edwards curve there is a Montgomery curve and a way to map the points of one curve in a 1:1 correspondence onto the other and vice versa. To see that assume that a curve in twisted Edwards form  $a \cdot x^2 + y^2 = 1 + d \cdot x^2 y^2$  is given. The associated Montgomery curve is then defined by the Montgomery equation

$$\frac{4}{a-d} y^2 = x^3 + \frac{2(a+d)}{a-d} \cdot x^2 + x \quad (5.11)$$

On the other hand a Montgomery curve  $By^2 = x^3 + Ax^2 + x$  with  $B \neq 0$  and  $A^2 \neq 4$  can give rise to a twisted Edwards curve defined by the equation

$$\left(\frac{A+2}{B}\right)x^2 + y^2 = 1 + \left(\frac{A-2}{B}\right)x^2 y^2 \quad (5.12)$$

Recalling from XXX that Montgomery curves are just a special class of Weierstraß, we now know that twisted Edwards curve are special Weierstraß curves too. So the more general way to describe elliptic curves are Weierstraß curves.

*Example 79.* Consider the pen jubjub curve from example XXX again. We know from XXX that it is a Montgomery curve and since Montgomery curves are equivalent to twisted Edwards curve, we want to write that curve in twisted Edwards form. We use XXX and compute the parameters  $a$  and  $d$  as

$$\begin{aligned} a &= \frac{A+2}{B} & \# \text{ insert } A=6 \text{ and } B=7 \\ &= \frac{8}{7} = 3 \end{aligned}$$

$$\begin{aligned} d &= \frac{A-2}{B} \\ &= \frac{4}{7} = 8 \end{aligned}$$

$$\# 7^{-1} = 2$$

So we get the defining parameters as  $a = 3$  and  $d = 8$ . Since our goal is to use this curve later on in implementations of pen-and-paper SNARKs, let's show that tiny-jubjub is moreover a **SNARK-friendly** twisted Edwards curve. To see that, we have to show that  $a$  is a quadratic

add reference

add reference

add reference

add reference



residue and  $d$  is a quadratic non residue. We therefore compute the Legendre symbols of  $a$  and  $d$  using the Euler criterium. We get

$$\left(\frac{3}{13}\right) = 3^{\frac{13-1}{2}} = 3^6 = 1$$

$$\left(\frac{8}{13}\right) = 8^{\frac{13-1}{2}} = 8^6 = 12 = -1$$

which proofs that tiny-jubjub is SNARK-friendly. We can write the tiny-jubjub curve in its affine twisted Edwards representation as

$$TJJ_{13} = \{(x, y) \in \mathbb{F}_{13} \times \mathbb{F}_{13} \mid 3 \cdot x^2 + y^2 = 1 + 8 \cdot x^2 \cdot y^2\}$$

Now that we have the abstract definition of our pen-jubjub curve in twisted Edwards form, we can compute the set of points, by inserting all pairs  $(x, y) \in \mathbb{F}_{13} \times \mathbb{F}_{13}$  similar to how we computed the curve points in its Weierstraß or Edwards representation. We get

$$PJJ_{13} = \{(0, 1), (0, 12), (1, 2), (1, 11), (2, 6), (2, 7), (3, 0), (5, 5), (5, 8), (6, 4), (6, 9), (7, 4), (7, 9), (8, 5), (8, 8), (10, 0), (11, 6), (11, 7), (12, 2), (12, 11)\}$$

```

852 Sage: F13 = GF(13)
853 Sage: L_EPJJ = []
854 .....: for x in F13:
855 .....:     for y in F13:
856 .....:         if F13(3)*x^2 + y^2 == 1+ F13(8)*x^2*y^2:
857 .....:             L_EPJJ.append( (x, y) )
858 Sage: EPJJ = Set(L_EPJJ)

```

**Twisted Edwards group law** As we have seen, twisted Edwards curves are equivalent to Montgomery curves and as such also have a group law. However, in contrast to Montgomery and Weierstraß curves, the group law of SNARK-friendly twisted Edwards curves can be described by single computation, that works in all cases, no matter if we add the neutral element, inverse, or if have to double a point. To see how the group law looks like, first observe that the point  $(0, 1)$  is a solution to  $a \cdot x^2 + y^2 = 1 + d \cdot x^2 \cdot y^2$  for any curve. The sum of any two points  $(x_1, y_1), (x_2, y_2)$  on an Edwards curve  $E(\mathbb{F})$  is then given by

$$(x_1, y_1) \oplus (x_2, y_2) = \left( \frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - a x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right)$$

859 and it can be shown that the point  $(0, 1)$  serves as the neutral element and the inverse of a point  
860  $(x_1, y_1)$  is given by  $(-x_1, y_1)$ .

*Example 80.* Lets look at the tiny-jubjub curve in Edwards form from example XXX again. As we have seen, this curve is given by

$$PJJ_{13} = \{(0, 1), (0, 12), (1, 2), (1, 11), (2, 6), (2, 7), (3, 0), (5, 5), (5, 8), (6, 4), (6, 9), (7, 4), (7, 9), (8, 5), (8, 8), (10, 0), (11, 6), (11, 7), (12, 2), (12, 11)\}$$

add reference

To get an understanding of the twisted Edwards addition law, let's first add the neutral element  $(0, 1)$  to itself. We apply the group law XXX and get

add reference

$$\begin{aligned}(0, 1) \oplus (0, 1) &= \left( \frac{0 \cdot 1 + 1 \cdot 0}{1 + 8 \cdot 0 \cdot 0 \cdot 1 \cdot 1}, \frac{1 \cdot 1 - 3 \cdot 0 \cdot 0}{1 - 8 \cdot 0 \cdot 0 \cdot 1 \cdot 1} \right) \\ &= (0, 1)\end{aligned}$$

So as expected, adding the neutral element to itself gives the neutral element again. Now let's add the neutral element to some other curve point. We get

$$\begin{aligned}(0, 1) \oplus (8, 5) &= \left( \frac{0 \cdot 5 + 1 \cdot 8}{1 + 8 \cdot 0 \cdot 8 \cdot 1 \cdot 5}, \frac{1 \cdot 5 - 3 \cdot 0 \cdot 8}{1 - 8 \cdot 0 \cdot 8 \cdot 1 \cdot 5} \right) \\ &= (8, 5)\end{aligned}$$

Again as expected adding the neutral element to any element will give the element again. Given any curve point  $(x, y)$ , we know that the inverse is given by  $(-x, y)$ . To see how the addition of a point to its inverse works out we therefore compute

$$\begin{aligned}(5, 5) \oplus (8, 5) &= \left( \frac{5 \cdot 5 + 5 \cdot 8}{1 + 8 \cdot 5 \cdot 8 \cdot 5 \cdot 5}, \frac{5 \cdot 5 - 3 \cdot 5 \cdot 8}{1 - 8 \cdot 5 \cdot 8 \cdot 5 \cdot 5} \right) \\ &= \left( \frac{12 + 1}{1 + 5}, \frac{12 - 3}{1 - 5} \right) \\ &= \left( \frac{0}{6}, \frac{12 + 10}{1 + 8} \right) \\ &= \left( 0, \frac{9}{9} \right) \\ &= (0, 1)\end{aligned}$$

861 So adding a curve point to its inverse gives the neutral element, as expected. As we have seen  
862 from these examples the twisted Edwards addition law handles edge cases particularly nice and  
863 in a unified way.

## 864 5.2 Elliptic Curves Pairings

865 As we have seen in XXX some groups comes with the notation of a so called pairing map,  
866 which is a non-degenerate bilinear map, from two groups into another group.

add reference

867 In this section, we discuss **pairings on elliptic curves**, which form the basis of several zk-  
868 SNARKs and other zero knowledge proof schemes. The SNARKs derived from pairings have  
869 the advantage of constant-sized proof sizes, which is crucial to blockchains.

870 We start out by defining elliptic curve pairings and discussing a simple application which  
871 bears some resemblance to the more advanced SNARKs. We then introduce the pairings arising  
872 from elliptic curves and describe Miller's algorithm which makes these pairings practical rather  
873 than just theoretically interesting.

874 Elliptic curves have a few structures, like the Weil or the Tate map, that qualifies as pairing.

875 **Embedding Degrees** As we will see in what follows, every elliptic curve gives rise to a pair-  
876 ing map. However, as we will see in example XXX, not every such pairing is efficiently com-  
877 putable. So in order to distinguish curves with efficiently computable pairings from the rest, we  
878 need to start with an introduction to the so called **embedding degree** of a curve.

add reference

To understand this term, let  $\mathbb{F}$  be a finite field,  $E(\mathbb{F})$  an elliptic curve over  $\mathbb{F}$ , and  $n$  a prime number that divides the order of  $E(\mathbb{F})$ . The embedding degree of  $E(\mathbb{F})$  with respect to  $n$  is then the smallest integer  $k$  such that  $n$  divides  $q^k - 1$ .

Fermat's little theorem XXX implies, that every curve has at least **some** embedding degree  $k$ , since at least  $k = n - 1$  is always a solution to the congruency  $q^k \equiv 1 \pmod{n}$  which implies that the remainder of the integer division of  $q^k - 1$  by  $n$  is 0.

*Example 81.* To get a better intuition of the embedding degree, let's consider the elliptic curve  $E_1(\mathbb{F}_5)$  from example XXX. We know from XXX that the order of  $E_1(\mathbb{F}_5)$  is 9 and since the only prime factor of 9 is 3, we compute the embedding degree of  $E_1(\mathbb{F}_5)$  with respect to 3.

To find that embedding degree we have to find the smallest integer  $k$ , such that 3 divides  $q^k - 1 = 5^k - 1$ . We try and increment until we find a proper  $k$ .

$k = 1: 5^1 - 1 = 4$	not divisible by 3
$k = 2: 5^2 - 1 = 24$	divisible by 3

So we know that the embedding degree of  $E_1(\mathbb{F}_5)$  is 2 relative to the the prime factor 3.

*Example 82.* Lets consider the tiny jubjub curve *TJJ\_13* from example XXX. We know from XXX that the order of *TJJ\_13* is 20 and that the order therefore has two prime factors. A "large" prime factor 5 and a small prime factor 2.

We start by computing the ebedding degree of *TJJ\_13* with respect to the large prime factor 5. To find that embedding degree we have to find the smallest integer  $k$ , such that 5 divides  $q^k - 1 = 13^k - 1$ . We try and increment until we find a proper  $k$ .

$k = 1: 13^1 - 1 = 12$	not divisible by 5
$k = 2: 13^2 - 1 = 168$	not divisible by 5
$k = 3: 13^3 - 1 = 2196$	not divisible by 5
$k = 4: 13^4 - 1 = 28560$	divisible by 5

So we know that the embedding degree of *TJJ\_13* is 4 relative to the the prime factor 5.

In real world applications, like on pairing friendly elliptic curves as for example BLS\_12-381, usually only the embedding degree of the large prime factor are relevant, which in case of out tiny-jubjub curve, is represented by 5. It should however, be noted that every prime factor of a curves order has its own notation of embedding degree despite the fact that this is mostly irrelevant in applications.

To find the embedding degree of the small prime factor 2 we have to find the smallest integer  $k$ , such that 2 divides  $q^k - 1 = 13^k - 1$ . We try and increment until we find a proper  $k$ .

$k = 1: 13^1 - 1 = 12$	divisible by 2
------------------------	----------------

So we know that the embedding degree of *TJJ\_13* is 1 relative to the the prime factor 2. So as we have seen, different prime factors can have different embedding degrees in general.

```

Sage: p = 13
Sage: # large prime factor
Sage: n = 5
Sage: for k in range(1,5): # Fermat's little theorem
.....:     if (p^k-1)%n == 0:
.....:         break

```

```

906 Sage: k
907 Sage: # small prime factor
908 Sage: n = 2
909 Sage: for k in range(1,2): # Fermat's little theorem
910 .....:     if (p^k-1)%n == 0:
911 .....:         break
912 Sage: k

```

913 *Example 83.* To give an example of a cryptographically secure real-world elliptic curve that  
 914 does not have a small embedding degree let's look at curve secp256k1 again. We know from  
 915 XXX that the order of this curve is a prime number, so we only have a single embedding degree.

add refer-  
ence

916 To test potential embedding degrees  $k$ , say in the range  $1 \dots 1000$ , we can invoke Sage and  
 917 compute:

```

918 Sage: p = 1157920892373161954235709850086879078532699846656%
919 40564039457584007908834671663
920 Sage: n = 1157920892373161954235709850086879078528375642790%
921 74904382605163141518161494337
922 Sage: for k in range(1,1000):
923 .....:     if (p^k-1)%n == 0:
924 .....:         break
925 Sage: k

```

926 So we see that secp256k1 has at least no embedding degree  $k < 1000$ , which renders secp256k1  
 927 as a curve that has no small embedding degree. A property that is of importance later on.

928 **Elliptic Curves over extension fields** Suppose that  $p$  is a prime number and  $\mathbb{F}_p$  its associated  
 929 prime field. We know from XXX, that the fields  $\mathbb{F}_{p^m}$  are extensions of  $\mathbb{F}_p$  in the sense that  $\mathbb{F}_p$   
 930 is a subfield of  $\mathbb{F}_{p^m}$ . This implies that we can extend the affine plane an elliptic curve is defined  
 931 on, by changing the base field to any extension field. To be more precise let  $E(\mathbb{F}) = \{(x, y) \in$   
 932  $\mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + a \cdot x + b\}$  be an affine short Weierstraß curve, with parameters  $a$  and  $b$  taken  
 933 from  $\mathbb{F}$ . If  $\mathbb{F}'$  is any extension field of  $\mathbb{F}$ , then we extend the domain of the curve by defining

add refer-  
ence

$$E(\mathbb{F}') = \{(x, y) \in \mathbb{F}' \times \mathbb{F}' \mid y^2 = x^3 + a \cdot x + b\} \quad (5.13)$$

934 So while we did not change the defining parameters, we consider curve points from the affine  
 935 plane over the extension field now. Since  $\mathbb{F} \subset \mathbb{F}'$  it can be shown that the original elliptic curve  
 936  $E(\mathbb{F})$  is a sub curve of the extension curve  $E(\mathbb{F}')$ .

*Example 84.* Consider the prime field  $\mathbb{F}_5$  from example XXX and the elliptic curve  $E_1(\mathbb{F}_5)$  from  
 example XXX. Since we know from XXX that  $\mathbb{F}_{5^2}$  is an extension field of  $\mathbb{F}_5$ , we can extend  
 the definition of  $E_1(\mathbb{F}_5)$  to define a curve over  $\mathbb{F}_{5^2}$ :

add refer-  
ence

$$E_1(\mathbb{F}_{5^2}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + x + 1\}$$

add refer-  
ence

937 Since  $\mathbb{F}_{5^2}$  contains 25 points, in order to compute the set  $E_1(\mathbb{F}_{5^2})$ , we have to try  $25 \cdot 25 = 625$   
 938 pairs, which is probably a bit to much for the average motivated reader. Instead, we involve  
 939 Sage to compute the curve for us. To do so choose the representation of  $\mathbb{F}_{5^2}$  from XXX. We get:

add refer-  
ence

```

940 Sage: F5= GF(5)
941 Sage: F5t.<t> = F5[]

```

add refer-  
ence

```

942 Sage: P = F5t (t^2+2)
943 Sage: P.is_irreducible()
944 Sage: F5_2.<t> = GF(5^2, name='t', modulus=P)
945 Sage: E1F5_2 = EllipticCurve(F5_2, [1,1])
946 Sage: E1F5_2.order()

```

So curve  $E_1(\mathbb{F}_{5^2})$  consist of 27 points, in contrast to curve  $E_1(\mathbb{F}_5)$ , which consists of 9 points. Printing the points gives

$$\begin{aligned}
 E_1(\mathbb{F}_{5^2}) = \{ & \mathcal{O}, (0, 4), (0, 1), (3, 4), (3, 1), (4, 3), (4, 2), (2, 4), (2, 1), \\
 & (4t + 3, 3t + 4), (4t + 3, 2t + 1), (3t + 2, t), (3t + 2, 4t), \\
 & (2t + 2, t), (2t + 2, 4t), (2t + 1, 4t + 4), (2t + 1, t + 1), \\
 & (2t + 3, 3), (2t + 3, 2), (t + 3, 2t + 4), (t + 3, 3t + 1), \\
 & (3t + 1, t + 4), (3t + 1, 4t + 1), (3t + 3, 3), (3t + 3, 2), (1, 4t) \}
 \end{aligned}$$

As we can see, curve  $E_1(\mathbb{F}_5)$  sits inside curve  $E(\mathbb{F}_{5^2})$ , which is implied from  $\mathbb{F}_5$  being a subfield of  $\mathbb{F}_{5^2}$ .

**Full Torsion groups** The fundamental theorem of finite cyclic groups XXX implies, that every prime factor  $n$  of a cyclic groups order defines a subgroup of the size of the prime factor. We called such a subgroup an  $n$ -torsion group. We have seen many of those subgroups in the examples XXX and XXX.

Now when we consider elliptic curve extensions as defined in XXX, we could ask, what happens to the  $n$ -torsion groups in the extension. One might intuitively think that their extension just parallels the extension of the curve. For example when  $E(\mathbb{F}_p)$  is a curve over prime field  $\mathbb{F}_p$ , with some  $n$ -torsion group  $\mathbb{G}$  and when we extend the curve to  $E(\mathbb{F}_{p^m})$ , then there is a bigger  $n$ -torsion group, such that  $\mathbb{G}$  is a subgroup. Naively this would make sense, as  $E(\mathbb{F}_p)$  is a sub-curve of  $E(\mathbb{F}_{p^m})$ .

However, the real situation is a bit more surprising than that. To see that, let  $\mathbb{F}_p$  be a prime field and  $E(\mathbb{F}_p)$  an elliptic curve of order  $r$ , with embedding degree  $k$  and  $n$ -torsion group  $E(\mathbb{F}_p)[n]$  for same prime factor  $n$  of  $r$ . Then it can be shown that the  $n$ -torsion group  $E(\mathbb{F}_{p^m})[n]$  of a curve extension is equal to  $E(\mathbb{F}_p)[n]$ , as long as the power  $m$  is less then the embedding degree  $k$  of  $E(\mathbb{F}_p)$ .

However, for the prime power  $p^m$ , for any  $m \geq k$ ,  $E(\mathbb{F}_{p^m})[n]$  is strictly larger then  $E(\mathbb{F}_p)[n]$  and contains  $E(\mathbb{F}_p)[n]$  as a subgroup. We call the  $n$ -torsion group  $E(\mathbb{F}_{p^k})[n]$  of the extension of  $E$  over  $\mathbb{F}_{p^k}$  the **full  $n$ -torsion group** of that elliptic curve. It can be shown that it contains  $n^2$  many elements and consists of  $n + 1$  subgroups, one of which is  $E(\mathbb{F}_p)[n]$ .

So roughly speaking, when we consider towers of curve extensions  $E(\mathbb{F}_{p^m})$ , ordered by the prime power  $m$ , then the  $n$ -torsion group stays constant for every level  $m$  small then the embedding degree, while it suddenly blossoms into a larger group on level  $k$ , with  $n + 1$  subgroups and it then stays like that for any level  $m$  larger then  $k$ . In other words, once the extension field is big enough to find one more point of order  $n$  (that is not defined over the base field), then we actually find all of the points in the full torsion group.

*Example 85.* Consider curve  $E_1(\mathbb{F}_5)$  again. We know that it contains a 3-torsion group and that the embedding degree of 3 is 2. From this we can deduce that we can find the full 3-torsion group  $E_1[3]$  in the curve extension  $E_1(\mathbb{F}_{5^2})$ , the latter of which we computed in XXX.

Since that curve is small, in order to find the full 3-torsion, we can loop through all elements of  $E_1(\mathbb{F}_{5^2})$  and check check the defining equation  $[3]P = \mathcal{O}$ . Invoking Sage, we compute

add refer-  
ence

add refer-  
ence

add refer-  
ence

add refer-  
ence

add refer-  
ence

```

979 Sage: INF = E1F5_2(0) # Point at infinity
980 Sage: L_E1_3 = []
981 Sage: for p in E1F5_2:
982     ....:     if 3*p == INF:
983     ....:         L_E1_3.append(p)
984 Sage: E1_3 = Set(L_E1_3) # Full 3-torsion set

```

we get

$$E_1[3] = \{\mathcal{O}, (1, t), (1, 4t), (2, 1), (2, 4), (2t+1, t+1), (2t+1, 4t+4), (3t+1, t+4), (3t+1, 4t+1)\}$$

985 *Example 86.* Consider the tiny jubjub curve from example XXX. we know from XXX that it  
 986 contains a 5-torsion group and that the embedding degree of 5 is 4. This implies that we can  
 987 find the full 5-torsion group  $TJJ\_13[5]$  in the curve extension  $TJJ\_13(\mathbb{F}_{13^4})$ .

add refer-  
ence

988 To compute the full torsion, first observe that since  $\mathbb{F}_{13^4}$  contains 28561 element, computing  
 989  $TJJ\_13(\mathbb{F}_{13^4})$  means checking  $28561^2 = 815730721$  elements. From each of these curve points  
 990  $P$ , we then have to check the equation  $[5]P = \mathcal{O}$ . Doing this for 815730721 is a bit to slow even  
 991 on a computer.

add refer-  
ence

992 Fortunate Sage has a way to loop through points of given order efficiently. The following  
 993 Sage code then gives a way to compute the full torsion group:

```

994 Sage: # define the extension field
995 Sage: F13= GF(13) # prime field
996 Sage: F13t.<t> = F13[] # polynomials over t
997 Sage: P = F13t(t^4+2) # irreducible polynomial of degree 4
998 Sage: P.is_irreducible()
999 Sage: F13_4.<t> = GF(13^4, name='t', modulus=P) # F_{13^4}
1000 Sage: TJJF13_4 = EllipticCurve(F13_4,[8,8]) # tiny jubjub extension
1001 Sage: # compute the full 5-torsion
1002 Sage: L_TJJF13_4_5 = []
1003 Sage: INF = TJJF13_4(0)
1004 Sage: for P in INF.division_points(5): # [5]P == INF
1005     ....:     L_TJJF13_4_5.append(P)
1006 Sage: len(L_TJJF13_4_5)
1007 Sage: TJJF13_4_5 = Set(L_TJJF13_4_5)

```

1008 So as expected we get a group that contains  $5^2 = 25$  elements. As it's rather tedious to write  
 1009 this group down and as we don't need it in what follows, we skip writing it. To see that the  
 1010 embedding degree 4 is actually the smallest prime power to find the full 5-torsion group, let's  
 1011 compute the 5-torsion group over of the tiny-jubjub curve the extension field  $\mathbb{F}_{13^3}$ . We get

```

1012 Sage: # define the extension field
1013 Sage: P = F13t(t^3+2) # irreducible polynomial of degree 3
1014 Sage: P.is_irreducible()
1015 Sage: F13_3.<t> = GF(13^3, name='t', modulus=P) # F_{13^3}
1016 Sage: TJJF13_3 = EllipticCurve(F13_3,[8,8]) # tiny jubjub extension
1017 Sage: # compute the 5-torsion
1018 Sage: L_TJJF13_3_5 = []
1019 Sage: INF = TJJF13_3(0)
1020 Sage: for P in INF.division_points(5): # [5]P == INF

```



```

1021 . . . . :      L_TJJF13_3_5.append(P)
1022 Sage: len(L_TJJF13_3_5)
1023 Sage: TJJF13_3_5 = Set(L_TJJF13_3_5) # full $5$-torsion

```

1024 So as we can see the 5-torsion group of tiny-jubjub over  $\mathbb{F}_{13^3}$  is equal to the 5-torsion group of  
 1025 tiny-jubjub over  $\mathbb{F}_{13}$  itself.

1026 *Example 87.* Lets look at curve Secp256k1. We know from XXX that the curve is of some  
 1027 prime order  $r$  and hence the only  $n$ -torsion group to consider is the curve itself. So the curve  
 1028 group is the  $r$ -torsion.

add refer-  
ence

However, in order to find the full  $r$ -torsion of Secp256k1, we need to compute the embed-  
 ding degree  $k$  and as we have seen in XXX it is at least not small. We know from Fermat's little  
 theorem that a finite embedding degree must exist, though. It can be shown that it is given by

add refer-  
ence

$$k = 192986815395526992372618308347813175472927379845817397100860523586360249056$$

1029 which is a 256bit number. So the embedding degree is huge, which implies that the fiel ex-  
 1030 tension  $\mathbb{F}_{p^k}$  is huge too. To understand how big  $\mathbb{F}_{p^k}$  is, recall that an element of  $\mathbb{F}_{p^m}$  can be  
 1031 represented as a string  $[x_0, \dots, x_m]$  of  $m$  elements, each containing a number from the prime  
 1032 field  $\mathbb{F}_p$ . Now in the case of Secp256k1, such a representation has  $k$ -many entries, each of 256  
 1033 bits in size. So without any optimizations, representing such an element would need  $k \cdot 256$  bits,  
 1034 which is too much to be represented in the observable universe.

1035 **Torsion-Subgroups** As we have stated above, any full  $n$ -torsion group contains  $n + 1$  cyclic  
 1036 subgroups, two of which are of particular interest in pairing based elliptic curve cryptography.  
 1037 To characterize these groups we need to consider the so called **Frobenius** endomorphism

$$\pi : E(\mathbb{F}) \rightarrow E(\mathbb{F}) : \begin{array}{ccc} (x, y) & \mapsto & (x^p, y^p) \\ \mathcal{O} & \mapsto & \mathcal{O} \end{array} \quad (5.14)$$

1038 of an alliptic curve  $E(\mathbb{F})$  over some finite field  $\mathbb{F}$  of characteristic  $p$ . It can be shown that  $\pi$  maps  
 1039 curve points to curve points. The first thing to note is that in case that  $\mathbb{F}$  is a prime field, the  
 1040 Frobenius endomorphism acts trivially, since  $(x^p, y^p) = (x, y)$  on prime fields, due to Fermat's  
 1041 little theorem XX. So the Frobenius map is more interesting over prime field extensions.

1042 With the Frobenius map at hand, we can now characterize two important subgroups of the  
 1043 full  $n$ -torsion. The first subgroup is the  $n$ -torsion group that already exists in the curve over the  
 1044 base field. In pairing based cryptography this group is usually written as  $\mathbb{G}_1$ , assuming that the  
 1045 prime factor ' $n$ ' in the definition is implicitly given. Since we know that the Frobenius map,  
 1046 acts trivially on curve over the prime field we can define  $\mathbb{G}_1$  as:

$$\mathbb{G}_1[n] := \{(x, y) \in E[n] \mid \pi(x, y) = (x, y)\} \quad (5.15)$$

1047 In more mathematical terms this definition means, that  $\mathbb{G}_1$  is the **Eigenspace** of the Frobenius  
 1048 map with respect to the **Eigenvalue** 1.

1049 Now it can be shown, that there is another subgroup of the full  $n$ -torsion group that can be  
 1050 characterized by the Frobenius map. In the context of so called type 3 pairing based cryptogra-  
 1051 phy this subgroup is usually called  $\mathbb{G}_2$  and it defined as

$$\mathbb{G}_2[n] := \{(x, y) \in E[n] \mid \pi(x, y) = [p](x, y)\} \quad (5.16)$$

1052 So in mathematical terms  $\mathbb{G}_2$  is the **Eigenspace** of the Frobenius map with respect to the **Eigen-**  
 1053 **value**  $p$ .

*Notation and Symbols* 9. If the prime factor  $n$  of the curves order is clear from the context, we sometimes simply write  $\mathbb{G}_1$  and  $\mathbb{G}_2$  to mean  $\mathbb{G}_1[n]$  and  $\mathbb{G}_2[n]$ , respectively.

It should be noted however, that sometimes other definitions of  $\mathbb{G}_2$  appear in the literature, however, in the context of pairing based cryptography, this is the most common one. It is particularly useful, as we can define hash functions that map into  $\mathbb{G}_2$ , which is not possible for all subgroups of the full  $n$ -torsion.

*Example 88.* Consider the curve  $E_1(\mathbb{F}_5)$  from example XXX again. As we have seen this curve has embedding degree  $k = 2$  and a full 3-torsion group is given by

add reference

$$E_1[3] = \{\mathcal{O}, (2, 1), (2, 4), (1, t), (1, 4t), (2t + 1, t + 1), (2t + 1, 4t + 4), (3t + 1, t + 4), (3t + 1, 4t + 1)\}$$

According to the general theory,  $E_1[3]$  contains 4 subgroups and we can characterize the subgroups  $\mathbb{G}_1$  and  $\mathbb{G}_2$  using the Frobenius endomorphism. Unfortunately at the time of this writing Sage did not have a predefined Frobenius endomorphism for elliptic curves, so we have to use the Frobenius endomorphism of the underlying field as a temporary workaround. We compute

```
Sage: L_G1 = []
Sage: for P in E1_3:
.....:     PiP = E1F5_2([a.frobenius() for a in P]) # pi(P)
.....:     if P == PiP:
.....:         L_G1.append(P)
Sage: G1 = Set(L_G1)
```

So as expected the group  $\mathbb{G}_1 = \{\mathcal{O}, (2, 4), (2, 1)\}$  is identical to the 3-torsion group of the (un-extended) curve over the prime field  $E_1(\mathbb{F}_5)$ . We can use almost the same algorithm to compute the group  $\mathbb{G}_2$  and get

```
Sage: L_G2 = []
Sage: for P in E1_3:
.....:     PiP = E1F5_2([a.frobenius() for a in P]) # pi(P)
.....:     pP = 5*P # [5]P
.....:     if pP == PiP:
.....:         L_G2.append(P)
Sage: G2 = Set(L_G2)
```

so we compute the the second subgroup of the full 3-torsion group of curve  $E_1$  as the set  $\mathbb{G}_2 = \{\mathcal{O}, (1, t), (1, 4t)\}$ .

*Example 89.* Considering the tiny-jubjub curve *TJJ\_13* from example XXX. In example XXX we computed its full 5 torsion, which is a group that has 6 subgroups. We compute  $G1$  using Sage as:

add reference

```
Sage: L_TJJ_G1 = []
Sage: for P in TJJF13_4_5:
.....:     PiP = TJJF13_4([a.frobenius() for a in P]) # pi(P)
.....:     if P == PiP:
.....:         L_TJJ_G1.append(P)
Sage: TJJ_G1 = Set(L_TJJ_G1)
```

add reference

We get  $\mathbb{G}_1 = \{\mathcal{O}, (7, 2), (8, 8), (8, 5), (7, 11)\}$



```

1092 Sage: L_TJJ_G1 = []
1093 Sage: for P in TJJF13_4_5:
1094     ....:     PiP = TJJF13_4([a.frobenius() for a in P]) # pi(P)
1095     ....:     pP = 13*P # [5]P
1096     ....:     if pP == PiP:
1097         ....:         L_TJJ_G1.append(P)
1098 Sage: TJJ_G1 = Set(L_TJJ_G1)

```

```

1099  $\mathbb{G}_2 = \{\mathcal{O}, (9t^2 + 7, t^3 + 11t), (9t^2 + 7, 12t^3 + 2t), (4t^2 + 7, 5t^3 + 10t), (4t^2 + 7, 8t^3 + 3t)\}$ 

```

1100 *Example 90.* Consider Bitcoin's curve Secp256k1 again. Since the group  $\mathbb{G}_1$  is identical to the  
1101 torsion group of the unextended curve and since Secp256k1 has prime order, we know, that,  
1102 in this case,  $\mathbb{G}_1$  is identical to Secp256k1. It is however, infeasible not just to compute  $\mathbb{G}_2$   
1103 itself, but to even compute an average element of  $\mathbb{G}_2$  as elements need too much storage to be  
1104 representable in this universe.

1105 **The Weil Pairing** In this part we consider a pairing function defined on the subgroups  $\mathbb{G}_1[r]$   
1106 and  $\mathbb{G}_2[r]$  of the full  $r$ -torsion  $E[r]$  of a short Weierstraß elliptic curve. To be more precise let  
1107  $E(\mathbb{F}_p)$  be an elliptic curve of embedding degree  $k$ , such that  $r$  is a prime factor of its order. Then  
1108 the **Weil pairing** is a bilinear, non-degenerate map

$$e(\cdot, \cdot) : \mathbb{G}_1[r] \times \mathbb{G}_2[r] \rightarrow \mathbb{F}_{p^k} ; (P, Q) \mapsto (-1)^r \cdot \frac{f_{r,P}(Q)}{f_{r,Q}(P)} \quad (5.17)$$

1109 where the extension field elements  $f_{r,P}(Q), f_{r,Q}(P) \in \mathbb{F}_{p^k}$  are computed by **Miller's algorithm**:  
1110 Understanding in detail how the algorithm works requires the concept of **divisors**, which we  
1111 don't really need in this book. The interested reader might look at [REFERENCES]

1112 In real world application of pairing friendly elliptic curves, the embedding degree is usually  
1113 a small number like 2, 4, 6 or 12 and the number  $r$  is the largest prime factor of the curves order.

1114 *Example 91.* Consider curve  $E_1(\mathbb{F}_5)$  from example XXX. Since the only prime factor of the  
1115 groups order is 3 we can not compute the Weil pairing on this group using our definition of  
1116 Miller's algorithm. In fact since  $\mathbb{G}_1$  is of order 3, executing the if statement on line XXX will  
1117 lead to a division by zero error in the computation of the slope  $m$ .

*Example 92.* Consider the tiny-jubjub curve  $TJJ\_13(\mathbb{F}_{13})$  from example XXX again. We want  
to instantiate the general definition of the Weil pairing for this example. To do so, recall that  
we have seen in example XXX, its embedding degree is 4 and that we have the following type-3  
pairing groups:

$$\mathbb{G}_1 = \{\mathcal{O}, (7, 2), (8, 8), (8, 5), (7, 11)\}$$

$$\mathbb{G}_2 = \{\mathcal{O}, (9t^2 + 7, t^3 + 11t), (9t^2 + 7, 12t^3 + 2t), (4t^2 + 7, 5t^3 + 10t), (4t^2 + 7, 8t^3 + 3t)\}$$

1118 where  $\mathbb{G}_1$  and  $\mathbb{G}_2$  are subgroups of the full 5-torsion found in the curve  $TJJ\_13(\mathbb{F}_{13^4})$ . The  
1119 type-3 Weil pairing is a map  $e(\cdot, \cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{F}_{13^4}$ . From the first if-statement in Miller's  
1120 algorithm, we can deduce that  $e(\mathcal{O}, Q) = 1$  as well as  $e(P, \mathcal{O}) = 1$  for all arguments  $P \in \mathbb{G}_1$  and  
1121  $Q \in \mathbb{G}_2$ . So in order to compute a non-trivial Weil pairing we choose the arguments  $P = (7, 2)$   
1122 and  $Q = (9t^2 + 7, 12t^3 + 2t)$ .

1123 In order to compute the pairing  $e((7, 2), (9t^2 + 7, 12t^3 + 2t))$  we have to compute the exten-  
1124 sion field elements  $f_{5,P}(Q)$  and  $f_{5,Q}(P)$  applying Miller's algorithm. Do do so first observe that  
1125 we have  $5 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2$ , so we get  $t = 2$  as well as  $b_0 = 1, b_1 = 0$  and  $b_2 = 1$ . The  
1126 loop therefore needs to be executed two times.

add refer-  
ence

add refer-  
ence

add refer-  
ence

add refer-  
ence

---

**Algorithm 7** Miller's algorithm for short Weierstraß curves  $y^2 = x^3 + ax + b$

---

**Require:**  $r > 3$ ,  $P \in E[r]$ ,  $Q \in E[r]$  and

$b_0, \dots, b_t \in \{0, 1\}$  with  $r = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \dots + b_t \cdot 2^t$  and  $b_t = 1$

**procedure** MILLER'S ALGORITHM( $P, Q$ )

**if**  $P = \mathcal{O}$  or  $Q = \mathcal{O}$  or  $P = Q$  **then**

**return**  $f_{r,P}(Q) \leftarrow (-1)^r$

**end if**

$(x_T, y_T) \leftarrow (x_P, y_P)$

$f_1 \leftarrow 1$

$f_2 \leftarrow 1$

**for**  $j \leftarrow t - 1, \dots, 0$  **do**

$m \leftarrow \frac{3 \cdot x_T^2 + a}{2 \cdot y_T}$

$f_1 \leftarrow f_1^2 \cdot (y_Q - y_T - m \cdot (x_Q - x_T))$

$f_2 \leftarrow f_2^2 \cdot (x_Q + 2x_T - m^2)$

$x_{2T} \leftarrow m^2 - 2x_T$

$y_{2T} \leftarrow -y_T - m \cdot (x_{2T} - x_T)$

$(x_T, y_T) \leftarrow (x_{2T}, y_{2T})$

**if**  $b_j = 1$  **then**

$m \leftarrow \frac{y_T - y_P}{x_T - x_P}$

$f_1 \leftarrow f_1 \cdot (y_Q - y_T - m \cdot (x_Q - x_T))$

$f_2 \leftarrow f_2 \cdot (x_Q + (x_P + x_T) - m^2)$

$x_{T+P} \leftarrow m^2 - x_T - x_P$

$y_{T+P} \leftarrow -y_T - m \cdot (x_{T+P} - x_T)$

$(x_T, y_T) \leftarrow (x_{T+P}, y_{T+P})$

**end if**

**end for**

$f_1 \leftarrow f_1 \cdot (x_Q - x_T)$

**return**  $f_{r,P}(Q) \leftarrow \frac{f_1}{f_2}$

**end procedure**

---

Computing  $f_{5,P}(Q)$ , we initiate  $(x_T, y_T) = (7, 2)$  as well as  $f_1 = 1$  and  $f_2 = 1$ . Then

$j$	$b_j$	$m$	$f_1$	$f_2$	$x_{2T}$	$y_{2T}$	$x_{T+P}$	$y_{T+P}$
1	.							

$$\begin{aligned} m &= \frac{3 \cdot x_T^2 + a}{2 \cdot y_T} \\ &= \frac{3 \cdot 2^2 + 1}{2 \cdot 4} = \frac{3}{3} \\ &= 1 \end{aligned}$$

$$\begin{aligned} f_1 &= f_1^2 \cdot (y_Q - y_T - m \cdot (x_Q - x_T)) \\ &= 1^2 \cdot (t - 4 - 1 \cdot (1 - 2)) = t - 4 + 1 \\ &= t + 2 \end{aligned}$$

$$\begin{aligned} f_2 &= f_2^2 \cdot (x_Q + 2x_T - m^2) \\ &= 1^2 \cdot (1 + 2 \cdot 2 - 1^2) = (1 + 4 - 1) \\ &= 4 \end{aligned}$$

$$\begin{aligned} x_{2T} &= m^2 - 2x_T \\ &= 1^2 - 2 \cdot 2 = -3 \\ &= 2 \end{aligned}$$

$$\begin{aligned} y_{2T} &= -y_T - m \cdot (x_{2T} - x_T) \\ &= -4 - 1 \cdot (2 - 2) = -4 \\ &= 1 \end{aligned}$$

So we update  $(x_T, y_T) = (2, 1)$  and since  $b_0 = 1$  we have to execute the if statement on line XXX in the for loop. However, since we only loop a single time, we don't need to compute  $y_{T+P}$ , since we only need the updated  $x_T$  in the final step. We get:

add reference

$$\begin{aligned} m &= \frac{y_T - y_P}{x_T - x_P} \\ &= \frac{1 - 4}{2 - x_P} \end{aligned}$$

$$f_1 = f_1 \cdot (y_Q - y_T - m \cdot (x_Q - x_T))$$

$$f_2 = f_2 \cdot (x_Q + (x_P + x_T) - m^2)$$

$$x_{T+P} = m^2 - x_T - x_P$$

## 5.3 Hashing to Curves

Elliptic curve cryptography frequently requires the ability to hash data onto elliptic curves. If the order of the curve is not a prime number hashing to prime number subgroups is also of importance. In the context of pairing friendly curves it is also sometimes necessary to hash specifically onto the group  $\mathbb{G}_1$  or  $\mathbb{G}_2$ .

As we have seen in XXX, many general methods are known to hash into groups in general and finite cyclic groups in particular. As elliptic groups are cyclic those method can be utilized in this case, too. However, in what follows we want to describe some methods special to elliptic curves, that are frequently used in applications.

add reference

**Try and increment hash functions** One of the most straight forward ways to hash a bitstring onto an elliptic curve point, in a secure way, is to use a cryptographic hash function together with one of the methods we described in XXX to hash to the modular arithmetics base field of the curve. Ideally the hash function generates an image that is at least one bit longer than the bit representation of the base field modulus.

add reference

The image in the base field can then be interpreted as the  $x$ -coordinate of the curve point and the two possible  $y$ -coordinates are then derived from the curve equation, while one of the bits that exceeded the modulus determines which of the two  $y$ -coordinates to choose.

Such an approach would be easy to implement and deterministic and it will conserve the cryptographic properties of the original hash function. However, not all  $x$ -coordinates generated in such a way, will result in quadratic residues, when inserted into the defining equation. It follows that not all field elements give rise to actual curve points. In fact on a prime field, only half of the field elements are quadratic residues and hence assuming an even distribution of the hash values in the field, this method would fail to generate a curve point in about half of the attempts.

One way to account for this problem is the so called **try and increment** method. Its basic assumption is, that hashing different values, the result will eventually lead to a valid curve point.

Therefore instead of simply hashing a string  $s$  to the field the concatenation of  $s$  with additional bytes is hashed to the field instead, that is a try and increment hash as described in XXX is used. If the first **try** of hashing to the field does not result in a valid curve point, the counter is **incremented** and the hashing is repeated again. This is done until a valid curve point is found eventually.

add reference

This method has the advantageous that is relatively easy to implement in code and that it preserves the cryptographic properties of the original hash function. However, it is not guaranteed to find a valid curve point, as there is a chance that all possible values in the chosen size of the counter will fail to generate a quadratic residue. Fortunately it is possible to make the probability for this arbitrarily small by choosing large enough counters and relying on the (approximate) uniformity of the hash-to-field function.

If the curve is not of prime order, the result will be a general curve point that might not be in the "large" prime order subgroup. A so called **cofactor clearing** step is then necessary to project the curve point onto the subgroup. This is done by scalar multiplication with the cofactor of prime order with respect to the curves order.

*Example 93.* Consider the tiny jubjub curve from example XXX. We want to construct a try and increment hash function, that hashes a binary string  $s$  of arbitrary length onto the large prime order subgroup of size 5.

add reference

Since the curve as well as our targeted subgroup are defined over the field  $\mathbb{F}_{13}$  and the binary representation of 13 is  $13.bits() = 1101$ , we apply SHA256 from Sage's crypto library on the

**Algorithm 8** Hash-to- $E(\mathbb{F}_r)$ **Require:**  $r \in \mathbb{Z}$  with  $r.\text{nbits}() = k$  and  $s \in \{0, 1\}^*$ **Require:** Curve equation  $y^2 = x^3 + ax + b$  over  $\mathbb{F}_r$ **procedure** TRY-AND-INCREMENT( $r, k, s$ ) $c \leftarrow 0$ **repeat** $s' \leftarrow s || c\_bits()$  $z \leftarrow H(s')_0 \cdot 2^0 + H(s')_1 \cdot 2^1 + \dots + H(s')_k \cdot 2^k$  $x \leftarrow z^3 + a \cdot z + b$  $c \leftarrow c + 1$ **until**  $z < r$  and  $x^{\frac{r-1}{2}} \bmod r = 1$ **if**  $H(s')_{k+1} == 0$  **then** $y \leftarrow \sqrt{x} \#(\text{root in } \mathbb{F}_r)$ **else** $y \leftarrow r - \sqrt{x} \#(\text{root in } \mathbb{F}_r)$ **end if****return**  $(x, y)$ **end procedure****Ensure:**  $(x, y) \in E(\mathbb{F}_r)$ 

concatenation  $s || c$  for some binary counter string and use the first 4 bits of the image to try to hash into  $\mathbb{F}_{13}$ . In case we are able to hash to a value  $z$ , such that  $z^3 + 8 \cdot z + 8$  is a quadratic residue in  $\mathbb{F}_{13}$ , we use the 5-th bit to decide which of the two possible roots of  $z^3 + 8 \cdot z + 8$  we will choose as the  $y$ -coordinate. The result is then a curve point different from the point at infinity. To project it to a point of  $\mathbb{G}_1$ , we multiply it with the cofactor 4. If the result is still not the point at infinity, it is the result of the hash.

To make this concrete let  $s = '10011001111010110100000111'$  be our binary string that we want to hash onto  $\mathbb{G}_1$ . We use a 4-bit binary counter, starting at zero, i.e we choose  $c = 0000$ . Invoking Sage we define the try-hash function as

```

Sage: import Crypto
Sage: from Crypto.Hash import SHA256
Sage: def try_hash(s, c):
.....:     s_1 = s+c
.....:     h = SHA256.new(s_1)
.....:     d = h.hexdigest()
.....:     d = Integer(d, base=16)
.....:     sign = d.str(2)[-5:-4]
.....:     d = d.str(2)[-4:]
.....:     z = Integer(d, base=2)
.....:     return (z, sign)
Sage: try_hash('10011001111010110100000111', '0000')
```

As we can see, our first attempt to hash into  $\mathbb{F}_{13}$  was not successful as 15 is not a number in  $\mathbb{F}_{13}$ , so we increment the binary counter by 1 and try again:

```

Sage: try_hash('10011001111010110100000111', '0001')
```

And we find a hash into  $\mathbb{F}_{13}$ . However, this point is not guaranteed to define a curve point. To see that we insert  $z = 3$  into the right side of the Weierstraß equation of the tinyjubjub curve and

1199 compute  $3^3 + 8 \cdot 3 + 8 = 7$ , but 7 is not a quadratic residue in  $\mathbb{F}_{13}$  since  $7^{\frac{13-1}{2}} = 7^6 = 12 = -1$ .  
 1200 So 3 is not a suitable point and we have to increment the counter two more times:

```
1201 Sage: try_hash('10011001111010110100000111', '0010')
1202 Sage: try_hash('10011001111010110100000111', '0011')
```

Since  $6^3 + 8 \cdot 6 + 8 = 12$  and we have  $\sqrt{12} \in \{5, 8\}$ , we finally found the valid  $x$  coordinate  $x = 6$  for the curve point hash. Now since the sign bit of this hash is 1, we choose the larger root  $y = 8$  as the  $y$ -coordinate and get the hash

$$H('10011001111010110100000111') = (6, 8)$$

which is a valid curve point on the tiny jubjub curve. Now in order to project it onto the "large" prime order subgroup we have to do cofactor clearing, that is we have to multiply the point with the cofactor 4. We get

$$[4](6, 8) = \mathcal{O}$$

1203 so the hash value is still not right. We therefore have to increment the counter two times again,  
 1204 until we finally find a correct hash to  $\mathbb{G}_1$

```
1205 Sage: try_hash('10011001111010110100000111', '0100')
1206 Sage: try_hash('10011001111010110100000111', '0101')
```

Since  $12^3 + 8 \cdot 12 + 8 = 12$  and we have  $\sqrt{12} \in \{5, 8\}$ , we found another valid  $x$  coordinate  $x = 12$  for the curve point hash. Now since the sign bit of this hash is 0, we choose the smaller root  $y = 5$  as the  $y$ -coordinate and get the hash

$$H('10011001111010110100000111') = (12, 5)$$

which is a valid curve point on the tiny jubjub curve and in order to project it onto the "large" prime order subgroup we have to do cofactor clearing, that is we have to multiply the point with the cofactor 4. We get

$$[4](12, 5) = (8, 5)$$

1207 So hashing the binary string '10011001111010110100000111' onto  $\mathbb{G}_1$  gives the hash value  
 1208  $(8, 5)$  as a result.

## 1209 5.4 Constructing elliptic curves

1210 Cryptographically secure elliptic curves like Secp256k1 from example XXX are known for  
 1211 quite some time. In the latest advancements of cryptography, it is however, often necessary to  
 1212 design and instantiate elliptic curves from scratch, that satisfy certain very specific properties.

1213 For example, in the context of SNARK development it was necessary to design a curve that  
 1214 can be efficiently implemented inside of a so called circuit, in order to enable primitives like  
 1215 elliptic curve signature schemes in a zero knowledge proof. Such a curve is given by the Baby-  
 1216 jubjub curve [XXX] and we have paralleled its definition by introducing the tiny-jubjub curve  
 1217 from example XX. As we have seen those curves are instances of so-called twisted Edwards  
 1218 curves and as such have easy to implement addition laws that work without branching. However,  
 1219 we introduced the tiny-jubjub curve out of thin air, as we just gave the curve parameters without  
 1220 explaining how we came up with them.

add refer-  
ence

add refer-  
ence

Another requirement in the context of many so called pairing based zero knowledge proofing systems is the existing of a suitable, pairing friendly curve with a specified security level and a low embedding degree as defined in XXX. Famous examples are the BLS\_12 or the NMT curves.

add reference

The major goal of this section is to explain the most important method to design elliptic curves with predefined properties from scratch, called the **complex multiplication method**. We will apply this method in section to synthesize a particular BLS\_6 curve, the most insecure BLS\_6 curve, which will serve as the main curve to build our pen-and-paper SNARKs on. As we will see, this curve has a "large" prime factor subgroup of order 13, which implies, that we can use our tiny-jubjub curve to implement certain elliptic curve cryptographic primitives in circuits over that BLS\_6 curve.

Before we introduce the complex multiplication method, we have to explain a few properties of elliptic curves that are of key importance in understanding the complex multiplication method.

**The Trace of Frobenius** To understand the complex multiplication method of elliptic curve, we have to define the so called **trace** of an elliptic curve first.

We know from XXX that elliptic curves over finite fields are cyclic groups of finite order. An interesting question therefore is, if it is possible to estimate the number of elements that curve contains. Since an affine short Weierstraß curve consists of pairs  $(x, y)$  of elements from a finite field  $\mathbb{F}_q$  plus the point at infinity and the field  $\mathbb{F}_q$  contains  $q$  elements, the number of curve points can not be arbitrarily large, since it can contain at most  $q^2 + 1$  many elements.

add reference

There is however, a more precise estimation, usually called the **Hasse bound**. To understand it, let  $E(\mathbb{F}_q)$  be an affine short Weierstraß curve over a finite field  $\mathbb{F}_w$  of order  $q$  and let  $|E(\mathbb{F}_q)|$  be the order of the curve. Then there is an integer  $t \in \mathbb{Z}$  called the **trace of Frobenius** of the curve, such that  $|t| \leq 2\sqrt{q}$  and

$$|E(\mathbb{F})| = q + 1 - t \quad (5.18)$$

A positive trace therefore implies, that the curve contains less points than the underlying field and a negative trace means that the curve contains more point. However, the estimation  $|t| \leq 2\sqrt{q}$  implies that the difference is not very large in either direction and the number of elements in an elliptic curve is always approximately in the same order of magnitude as the size of the curve's basefield.

*Example 94.* Consider the elliptic curve  $E_1(\mathbb{F}_5)$  from example XXX. We know that it contains 9 curve points. Since the order of  $\mathbb{F}_5$  is 5 we compute the trace of  $E_1(\mathbb{F})$  to be  $t = -3$ , since the Hasse bound is given by

$$9 = 5 + 1 - (-3)$$

add reference

And indeed we have  $|t| \leq 2\sqrt{q}$ , since  $\sqrt{5} > 2.23$  and  $|-3| = 3 \leq 4.46 = 2 \cdot 2.23 < 2 \cdot \sqrt{5}$ .

*Example 95.* To compute the trace of the tiny-jubjub curve, oberse from example XXX, that the order of  $PJJ_{13}$  is 20. Since the order of  $\mathbb{F}_{13}$  is 13, we can therefore use the Hasse bound and compute the trace as  $t = -6$ , since

oberse

add reference

$$20 = 13 + 1 - (-6)$$

Again we have  $|t| \leq 2\sqrt{q}$ , since  $\sqrt{13} > 3.60$  and  $|-6| = 6 \leq 7.20 = 2 \cdot 3.60 < 2 \cdot \sqrt{13}$ .

*Example 96.* To compute the trace of Secp256k1, recall from example XXX, that this curve is

add reference

defined over a prime field with  $p$  elements and that the order of that group is given by  $r$ , with

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$

$$r = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

Using the Hesse bound  $r = p + 1 - t$ , we therefore compute  $t = p + 1 - r$ , which gives the trace of curve Secp256k1 as

$$t = 4324203865659656852420866390673177327$$

1253 So as we can see Secp256k1 contains less elements than its underlying field. However, the  
1254 difference is tiny, since the order of Secp256k1 is in the same order of magnitude as the order  
1255 of the underlying field. Compared to  $p$  and  $r$ ,  $t$  is tiny.

```
1256 Sage: p = 115792089237316195423570985008687907853269984665640564039457584
1257 Sage: r = 115792089237316195423570985008687907852837564279074904382605163
1258 Sage: t = p + 1 - r
1259 Sage: t.nbits()
1260 Sage: abs(RR(t)) <= 2*sqrt(RR(p))
```

1261 **The  $j$ -invariant** As we have seen in XXX two elliptic curve  $E_1(\mathbb{F})$  defined by  $y^2 = x^3 + ax + b$   
1262 and  $E_2(\mathbb{F})$  defined by  $y^2 + a'x + b'$  are strictly isomorphic, if and only if there is a quadratic  
1263 residue  $d \in \mathbb{F}$ , such that  $a' = ad^2$  and  $b' = bd^3$ .

add refer-  
ence

1264 There is however, a more general way to classify elliptic curves over finite fields  $\mathbb{F}_q$ , based  
1265 on the so called  $j$ -invariant of an elliptic curve:

$$j(E(\mathbb{F}_q)) = (1728 \bmod q) \frac{4 \cdot a^3}{4 \cdot a^3 + (27 \bmod q) \cdot b^2} \quad (5.19)$$

1266 with  $j(E(\mathbb{F}_q)) \in \mathbb{F}_q$ . We will not go into the details of the  $j$ -invariant, but state only, that two  
1267 elliptic curves  $E_1(\mathbb{F})$  and  $E_2(\mathbb{F}')$  are isomorphic over the algebraic closures of  $\mathbb{F}$  and  $\mathbb{F}'$ , if and  
1268 only if  $\overline{\mathbb{F}} = \overline{\mathbb{F}'}$  and  $j(E_1) = j(E_2)$ .

1269 So the  $j$ -invariant is an important tool to classify elliptic curves and it is needed in the com-  
1270 plex multiplication method to decide on an actual curve instantiation, that implements abstractly  
1271 chosen properties.

*Example 97.* Consider the elliptic curve  $E_1(\mathbb{F}_5)$  from example XXX. We compute its  $j$ -invariant  
as

add refer-  
ence

$$\begin{aligned} j(E_1(\mathbb{F}_5)) &= (1728 \bmod 5) \frac{4 \cdot 1^3}{4 \cdot 1^3 + (27 \bmod 5) \cdot 1^2} \\ &= 3 \frac{4}{4 + 2} \\ &= 3 \cdot 4 = 2 \end{aligned}$$

*Example 98.* Consider the elliptic curve  $PJJ\_13$  from example XXX. We compute its  $j$ -invariant

add refer-  
ence



as

$$\begin{aligned}
 j(E_1(\mathbb{F}_5)) &= (1728 \bmod 13) \frac{4 \cdot 8^3}{4 \cdot 8^3 + (27 \bmod 13) \cdot 8^2} \\
 &= 12 \cdot \frac{4 \cdot 5}{4 \cdot 5 + 1 \cdot 12} \\
 &= 12 \cdot \frac{7}{7 + 12} \\
 &= 12 \cdot 7 \cdot 6^{-1} \\
 &= 12 \cdot 7 \cdot 11 \\
 &01
 \end{aligned}$$

1272 *Example 99.* Consider Secp256k1 from example XXX. We compute its  $j$ -invariant using Sage:

add reference

```

1273 Sage: p = 115792089237316195423570985008687907853269984665640564039457584
1274 Sage: F = GF(p)
1275 Sage: j = F(1728) * (F(4) * F(0)^3) / (F(4) * F(0)^3 + F(27) * F(7)^2)
1276 Sage: j == F(0)

```

1277 **The Complex Multiplication Method** As we have seen in the previous sections, elliptic  
1278 curves have various defining properties, like their order and their prime factors, the embedding  
1279 degree, or the cardinality of the base field. The so called **complex multiplication** (CM) gives a  
1280 practical method for constructing elliptic curves with pre-defined restrictions on the order and  
1281 the base field.

1282 The method usually starts by choosing a base field  $\mathbb{F}_q$  of the curve  $E(\mathbb{F}_q)$  we want to con-  
1283 struct, such that  $q = p^m$  for some prime number  $p$  and “ $m \in \mathbb{N}$  with  $m \geq 1$ . We assume  $p > 3$   
1284 to simplify things in what follows.

1285 Next the trace of Frobenius  $t \in \mathbb{Z}$  of the curve is chosen, such that  $p$  and  $t$  are coprime,  
1286 i.e. such that  $\gcd(p, t) = 1$  holds true. The choice of  $t$  also defines the curves order  $r$ , since  
1287  $r = p + 1 - t$  by the Hasse bound XXX, so choosing  $t$ , will define the large order subgroup as  
1288 well as all small cofactors.  $r$  has to be defined in such a way, that the elliptic curve meets the  
1289 security requirements of the application it is designed for.

add reference

1290 Note that the choice of  $p$  and  $t$  also determines the embedding degree  $k$  of any prime order  
1291 subgroup of the curve, since  $k$  is defined as the smallest number, such that the prime order  $n$   
1292 divides the number  $q^k - 1$ .

1293 In order for the complex multiplication method to work, both  $q$  and  $t$  can not be arbitrary,  
1294 but must be chosen in such a way that two additional integers  $D \in \mathbb{Z}$  and  $v \in \mathbb{Z}$  exist, such that  
1295  $D < 0$  as well as  $D \bmod 4 = 0$  or  $D \bmod 4 = 1$  and the equation

$$4q = t^2 + |D|v^2 \quad (5.20)$$

1296 holds. If those numbers exist, we call  $D$  the **CM-discriminant** and we know that we can  
1297 construct a curve  $E(\mathbb{F}_q)$  over a finite field  $\mathbb{F}_q$ , such that the order of the curve is  $|E(\mathbb{F}_q)| =$   
1298  $q + 1 - t$ .

1299 It is the content of the complex multiplication method to actually construct such a curve,  
1300 that is finding the parameters  $a$  and  $b$  from  $\mathbb{F}_q$  in the defining Weierstraß equation, such that the  
1301 curve has the desired order  $r$ .

1302 Finding solutions to equation XXX, can be achieved in different ways, which we will not  
1303 look much into. In general it can be said, that there are well known constraints for elliptic curve

add reference

families like the BLS (ECT) families, that provides families of solutions. In what follows we will look at one type curves the BLS-family, which gives an entire range of solutions.

Assuming that proper parameters  $q, t, D$  and  $v$  are found, we have to compute the so called **Hilbert class polynomial**  $H_D \in \mathbb{Z}[x]$  of the CM-discriminant  $D$ , which is a polynomial with integer coefficients. To do so, we first have to compute the following set:

$$ICG(D) = \{(A, B, C) \mid A, B, C \in \mathbb{Z}, D = B^2 - 4AC, \gcd(A, B, C) = 1, \\ |B| \leq A \leq \sqrt{\frac{|D|}{3}}, A \leq C, \text{ if } B < 0 \text{ then } |B| < A < C\}$$

One way to compute this set, is to first compute the integer  $A_{max} = \text{Floor}(\sqrt{\frac{|D|}{3}})$ , then loop through all the integers  $A$  in the range  $[0, \dots, A_{max}]$  as well as through all the integers  $B$  in the range  $[-A_{max}, \dots, A_{max}]$  and to see if there is an integer  $C$ , that satisfies  $D = B^2 - 4AC$  and the rest of the requirements in XXX.

To compute the Hilbert class polynomial, the so called  **$j$ -function** (or  $j$ -invariant) is needed, which is a complex function defined on the upper half  $\mathbb{H}$  of the complex plane  $\mathbb{C}$ , usually written as

$$j: \mathbb{H} \rightarrow \mathbb{C} \quad (5.21)$$

Roughly speaking what this means is that the  $j$ -functions takes complex numbers  $(x + i \cdot y)$  with positive imaginary part  $y > 0$  as inputs and returns a complex number  $j(x + i \cdot y)$  as result.

For the sake of this book, it is not important to actually understand the  $j$ -function, and we can use Sage to compute it in a similar way as we would use Sage to computer any other well known function. It should be noted however, that the computation of the  $j$ -function in Sage is sometimes prone to precision errors. For example, the  $j$ -function has a root in  $\frac{-1+i\sqrt{3}}{2}$ , which Sage only approximates. Therefore using Sage to compute the  $j$ -function, we need to take precision loss into account and eventually round to the nearest integer.

```
Sage: z = ComplexField(100)(0, 1)
Sage: z # (0+1i)
Sage: elliptic_j(z)
Sage: # j-function only defined for positive imaginary arguments
Sage: z = ComplexField(100)(1, -1)
Sage: try:
.....:     elliptic_j(z)
.....: except PariError:
.....:     pass
Sage: # root at (-1+i sqrt(3))/2
Sage: z = ComplexField(100)(-1, sqrt(3))/2
Sage: elliptic_j(z)
Sage: elliptic_j(z).imag().round()
Sage: elliptic_j(z).real().round()
```

With a way to compute the  $j$ -function and the precomputed set  $ICG(D)$  at hand, we can now compute the Hilbert class polynomial as

$$H_D(x) = \prod_{(A,B,C) \in ICG(D)} \left( x - j\left(\frac{-B + \sqrt{D}}{2A}\right) \right) \quad (5.22)$$

add refer-  
ence

So what we do is we loop over all elements  $(A, B, C)$  from the set  $ICG(D)$  and compute the  $j$ -function at the point  $\frac{-B+\sqrt{D}}{2A}$ , where  $D$  is the CM-discriminant that we decided in a previous step. The result then defines a factor of the Hilbert class polynomial and all factors are multiplied together.

It can be shown, that the Hilbert class polynomial is an integer polynomial, but actual computations need high precision arithmetics to avoid approximation errors, that usually occur in computer approximations of the  $j$ -function as shown above. So in case the calculated Hilbert class polynomial does not have integer coefficients, we need to round the result to the nearest integer. Given that the precision we used was high enough, the result will be correct.

In the next step we use the Hilbert class polynomial  $H_D \in \mathbb{Z}[x]$  and project it to a polynomial  $H_{D,q} \in \mathbb{F}_q[x]$  with coefficients in the base field  $\mathbb{F}_q$  as chosen in the first step. We do this by simply computing the new coefficients as the old coefficients modulus  $p$ , that is if  $H_D(x) = a_mx^m + a_{m-1}x^{m-1} + \dots + a_1x + a_0$  we compute the  $q$ -modulus of each coefficient  $\tilde{a}_j = a_j \bmod p$  which defines the **projected Hilbert class polynomial** as

$$H_{D,p}(x) = \tilde{a}_m x^m + \tilde{a}_{m-1} x^{m-1} + \dots + \tilde{a}_1 x + \tilde{a}_0$$

We then search for roots of  $H_{D,p}$ , since every root  $j_0$  of  $H_{D,p}$  defines a family of elliptic curves over  $\mathbb{F}_q$ , which all have a  $j$ -invariant XXX equal to  $j_0$ . We can pick any root and all of them will lead to proper curves eventually.

add reference

However, some of the curves with the correct  $j$ -invariant might have an order different from the one we initially decided on. So we need a way to decide on a curve with the correct order.

To compute such a curve, we have to distinguish a few different cases, based on our choice of the root  $j_0$  and of the CM-discriminant  $D$ . If  $j_0 \neq 0$  or  $j_0 \neq 1728 \bmod q$  we compute  $c_1 = \frac{j_0}{(1728 \bmod q) - j_0}$  and then we chose some arbitrary quadratic non-residue  $c_2 \in \mathbb{F}_q$  and some arbitrary cubic non residue  $c_3 \in \mathbb{F}_q$ .

The following table is guaranteed to define a curve with the correct order  $r = q + 1 - t$ , for the trace of Frobenius  $t$  we initially decided on:

- Case  $j_0 \neq 0$  and  $j_0 \neq 1728 \bmod q$ . A curve with the correct order is defined by one of the following equations

$$y^2 = x^3 + 3c_1x + 2c_1 \quad \text{or} \quad y^2 = x^3 + 3c_1c_2^2x + 2c_1c_2^3 \quad (5.23)$$

- Case  $j_0 = 0$  and  $D \neq -3$ . A curve with the correct order is defined by one of the following equations

$$y^2 = x^3 + 1 \quad \text{or} \quad y^2 = x^3 + c_2^3 \quad (5.24)$$

- Case  $j_0 = 0$  and  $D = -3$ . A curve with the correct order is defined by one of the following equations

$$\begin{aligned} y^2 &= x^3 + 1 \quad \text{or} \quad y^2 = x^3 + c_2^3 \quad \text{or} \\ y^2 &= x^3 + c_2^2 \quad \text{or} \quad y^2 = c_2^3c_3^2 \quad \text{or} \\ y^2 &= x^3 + c_3^{-2} \quad \text{or} \quad y^2 = x^3 + c_3^{-2}c_2^3 \end{aligned}$$

- Case  $j_0 = 1728 \bmod q$  and  $D \neq -4$ . A curve with the correct order is defined by one of the following equations

$$y^2 = x^3 + x \quad \text{or} \quad y^2 = x^3 + c_2^2x \quad (5.25)$$

- Case  $j_0 = 1728 \bmod q$  and  $D = -4$ . A curve with the correct order is defined by one of the following equations

$$\begin{aligned} y^2 &= x^3 + x \quad \text{or} \quad y^2 = x^3 + c_2x \quad \text{or} \\ y^2 &= x^3 + c_2^2x \quad \text{or} \quad y^2 = x^3 + c_2^3x \end{aligned}$$

To decide the proper defining Weierstraß equation, we therefore have to compute the order of any of the potential curves above and then choose the one that fits out initial requirements. Since it can be shown that the Hilbert class polynomials for the CM-discriminants  $D = -3$  and  $D = -4$  are given by  $H_{-3,q}(x) = x$  and  $H_{-4,q}(x) = x - (1728 \bmod q)$  (EXERCISE) the previous cases are exhaustive.

To summarize, using the complex multiplication method, it is possible to synthesize elliptic curve with predefined order over predefined base fields from scratch. However, the curves that are constructed this way are just some representatives of a larger class of curves, all of which have the same order. In applications it is therefore sometimes more advantageous to choose a different representative from that class. To do so recall from XXX, that any curve defined by the Weierstraß equation  $y^2 = x^3 + ax + b$  is isomorphic to a curve of the form  $y^2 = x^3 + ad^2x + bd^3$  for some quadratic residue  $d \in \mathbb{F}_q$ .

add reference

So in order to find a nice representative (e.g. with small parameters  $a$  and  $b$ ) in a last step, the designer might choose a quadratic residue  $d$  such that the transformed curve looks the way they wanted it.

*Example 100.* Consider curve  $E_1(\mathbb{F}_5)$  from example XXX. We want to use the complex multiplication method to derive that curve from scratch. Since  $E_1(\mathbb{F}_5)$  is a curve of order  $r = 9$  over the prime field of order  $q = 5$ , we know from example XX that its trace of Frobenius is  $t = -3$ , which also implies that  $q$  and  $|t|$  are coprime.

add reference

We then have to find parameters  $D, v \in \mathbb{Z}$  with  $D < 0$  and  $D \bmod 4 = 0$  or  $D \bmod 4 = 1$ , such that  $4q = t^2 + |D|v^2$  holds. We get

$$\begin{aligned} 4q &= t^2 + |D|v^2 && \Rightarrow \\ 20 &= (-3)^2 + |D|v^2 && \Leftrightarrow \\ 11 &= |D|v^2 \end{aligned}$$

Now since 11 is a prime number, the only solution is  $|D| = 11$  and  $v = 1$  here. So  $D = -11$  and since the Euclidean division of  $-11$  by 4 is  $-11 = -3 \cdot 4 + 1$  we have  $-11 \bmod 4 = 1$ , which shows that  $D = -11$  is a proper choice.

In the next step, we have to compute the Hilbert class polynomial  $H_{-11}$  and to do so, we first have to find the set  $ICG(D)$ . To compute that set, observe, that since  $\sqrt{\frac{|D|}{3}} \approx 1.915 < 2$ , we know from  $A \leq \sqrt{\frac{|D|}{3}}$  and  $A \in \mathbb{Z}$  that  $A$  must be either 0 or 1.

For  $A = 0$ , we know  $B = 0$  from the constraint  $|B| \leq A$ , but in this case there can be no  $C$  satisfying  $-11 = B^2 - 4AC$ . So we try  $A = 1$  and deduce  $B \in \{-1, 0, 1\}$  from the constraint  $|B| \leq A$ . The case  $B = -1$  can be excluded since then  $B < 0$  has to imply  $|B| < A$ . In addition, the case  $B = 0$  can be excluded as there can be integer  $C$  with  $-11 = -4C$  since 11 is a prime number.

This leaves the case  $B = 1$  and we compute  $C = 3$  from the equation  $-11 = 1^2 - 4C$ , which gives the solution  $(A, B, C) = (1, 1, 3)$  and we get

$$ICG(D) = \{(1, 1, 3)\}$$

With the set  $ICG(D)$  at hand we can compute the Hilbert class polynomial of  $D = -11$ . To do so, we have to insert the term  $\frac{-1+\sqrt{-11}}{2}$  into the  $j$ -function. To do so first observe that  $\sqrt{-11} = i\sqrt{11}$ , where  $i$  is the imaginary unit, defined by  $i^2 = -1$ . Using this, we can invoke SageMath to compute the  $j$ -invariant and get

$$H_{-11}(x) = x - j \left( \frac{-1 + i\sqrt{11}}{2} \right) = x + 32768$$

So as we can see, in this particular case, the Hilbert class polynomial is a linear function with a single integer coefficient. In the next step we have to project it onto a polynomial from  $\mathbb{F}_5[x]$ , by computing the modular 5 remainder of the coefficients 1 and 32768. We get  $32768 \bmod 5 = 3$  from which follows that the projected Hilbert class polynomial is

$$H_{-11,5}(x) = x + 3$$

considered as a polynomial from  $\mathbb{F}_5[x]$ . As we can see the only root of this polynomial is  $j = 2$ , since  $H_{-11,5}(2) = 2 + 3 = 0$ . We therefore have a situation with  $j \neq 0$  and  $j \neq 1728$ , which tells us that we have to compute the parameter

$$c_1 = \frac{2}{1728 - 2}$$

1393 in modular 5 arithmetics. Since  $1728 \bmod 5 = 3$ , we get  $c_1 = 2$ . Then we have to check if the  
 1394 curve  $E(\mathbb{F}_5)$  defined by the Weierstraß  $y^2 = x^3 + 3 \cdot 2x + 2 \cdot 2$  has the correct order. We invoke  
 1395 Sage and find that the order is indeed 9, so it is a curve with the required parameters and we are  
 1396 done.

Note however, that in real world applications, it might be useful to choose parameters  $a$  and  $b$  that have certain properties, e.g. to be as small as possible. As we know from XXX, choosing any quadratic residue  $d \in \mathbb{F}_5$  gives a curve of the same order defined by  $y^2 = x^3 + ak^2x + bk^3$ . Since 4 is a quadratic residue in  $\mathbb{F}_5$ , we can transform the curve defined by  $y^2 = x^3 + x + 4$  into the curve  $y^2 = x^3 + 4^2 + 4 \cdot 4^3$  which gives

$$y^2 = x^3 + x + 1$$

1397 which is the curve  $E_1(\mathbb{F}_5)$ , that we used extensively throughout this book. So using the complex  
 1398 multiplication method, we were able to derive a curve with specific properties from scratch.

1399 *Example 101.* Consider the tiny jubjub curve  $TJJ\_13$  from example XXX. We want to use the  
 1400 complex multiplication method to derive that curve from scratch. Since  $TJJ\_13$  is a curve of  
 1401 order  $r = 20$  over the prime field of order  $q = 13$ , we know from example XX that its trace of  
 1402 Frobenius is  $t = -6$ , which also implies that  $q$  and  $|t|$  are coprime.

We then have to find parameters  $D, v \in \mathbb{Z}$  with  $D < 0$  and  $D \bmod 4 = 0$  or  $D \bmod 4 = 1$ , such that  $4q = t^2 + |D|v^2$  holds. We get

$$\begin{aligned} 4q &= t^2 + |D|v^2 && \Rightarrow \\ 4 \cdot 13 &= (-6)^2 + |D|v^2 && \Rightarrow \\ 52 &= 36 + |D|v^2 && \Leftrightarrow \\ 16 &= |D|v^2 \end{aligned}$$

1403 This equation has two solutions for  $(D, v)$ , given by  $(-4, \pm 2)$  and  $(-16, \pm 1)$ . Now looking at  
 1404 the first solution, we know that the case  $D = -4$  implies  $j = 1728$  and the constructed curve is

add refer-  
ence

add refer-  
ence

defined by a Weierstraß equation XXX that has a vanishing parameter  $b = 0$ . We can therefore conclude that choosing  $D = -4$  will not help us reconstructing *TJJ\_13*. It will produce curves with order 20, just not the one we are looking for.

add reference

So we choose the second solution  $D = -16$  and in the next step, we have to compute the Hilbert class polynomial  $H_{-16}$ . To do so, we first have to find the set  $ICG(D)$ . To compute that set, observe, that since  $\sqrt{\frac{|-16|}{3}} \approx 2.31 < 3$ , we know from  $A \leq \sqrt{\frac{|-16|}{3}}$  and  $A \in \mathbb{Z}$  that  $A$  must be in the range  $0..2$ . So we loop through all possible values of  $A$  and through all possible values of  $B$  under the constraints  $|B| \leq A$  and if  $B < 0$  then  $|B| < A$  and the compute potential  $C$ 's from  $-16 = B^2 - 4AC$ . We get the following two solution  $(1, 0, 4)$  and  $(2, 0, 2)$ , giving we get

$$ICG(D) = \{(1, 0, 4), (2, 0, 2)\}$$

With the set  $ICG(D)$  at hand we can compute the Hilbert class polynomial of  $D = -16$ . We can invoke Sagemath to compute the  $j$ -invariant and get

$$\begin{aligned} H_{-16}(x) &= \left( x - j \left( \frac{i\sqrt{16}}{2} \right) \right) \left( x - j \left( \frac{i\sqrt{16}}{4} \right) \right) \\ &= (x - 287496)(x - 1728) \end{aligned}$$

So as we can see, in this particular case, the Hilbert class polynomial is a quadratic function with two integer coefficient. In the next step we have to project it onto a polynomial from  $\mathbb{F}_5[x]$ , by computing the modular 5 remainder of the coefficients 1, 287496 and 1728. We get  $287496 \bmod 13 = 1$  and  $1728 \bmod 13 = 2$  from which follows that the projected Hilbert class polynomial is

$$H_{-11,5}(x) = (x - 1)(x - 12) = (x + 12)(x + 1)$$

considered as a polynomial from  $\mathbb{F}_5[x]$ . So we have two roots given by  $j = 1$  and  $j = 12$ . We already know that  $j = 12$  is the wrong root to construct the tiny jubjub curve, since  $1728 \bmod 13 = 2$  and that case can not construct a curve with  $b \neq 0$ . So we choose  $j = 1$ .

Another way to decide the proper root, is to compute the  $j$ -invariant of the tiny-jubjub curve. We get

$$\begin{aligned} j(TJJ_{13}) &= 12 \frac{4 \cdot 8^3}{4 \cdot 8^3 + 1 \cdot 8^2} \\ &= 12 \frac{4 \cdot 5}{4 \cdot 5 + 12} \\ &= 12 \frac{7}{7 + 12} \\ &= 12 \frac{7}{7 + 12} \\ &= 1 \end{aligned}$$

which is equal to the root  $j = 1$  of the Hilbert class polynomial  $H_{-16,13}$  as expected. We therefore have a situation with  $j \neq 0$  and  $j \neq 1728$ , which tells us that we have to compute the parameter

$$c_1 = \frac{1}{12 - 1} = 6$$

in modular 5 arithmetics. Since  $1728 \bmod 13 = 12$ , we get  $c_1 = 6$ . Then we have to check if the curve  $E(\mathbb{F}_5)$  defined by the Weierstraß  $y^2 = x^3 + 3 \cdot 6x + 2 \cdot 6$  which is equivalent to

$$y^2 = x^3 + 5x + 12$$



has the correct order. We invoke Sage and find that the order is 8, which implies that the trace of this curve is 6 not  $-6$  as required. So we have to consider the second possibility and choose some quadratic non-residue  $c_2 \in \mathbb{F}_{13}$ . We choose  $c_2 = 5$  and compute the Weierstraß equation  $y^2 = x^3 + 5c_2^2 + 12c_2^3$  as

$$y^2 = x^3 + 8x + 5$$

We invoke Sage and find that the order is 20, which is indeed the correct one. As we know from XXX, choosing any quadratic residue  $d \in \mathbb{F}_5$  gives a curve of the same order defined by  $y^2 = x^2 + ad^2x + bd^3$ . Since 12 is a quadratic residue in  $\mathbb{F}_{13}$ , we can transform the curve defined by  $y^2 = x^3 + 8x + 5$  into the curve  $y^2 = x^3 + 12^2 \cdot 8 + 5 \cdot 12^3$  which gives

$$y^2 = x^3 + 8x + 8$$

1411 which is the tiny jubjub curve, that we used extensively throughout this book. So using the  
1412 complex multiplication method, we were able to derive a curve with specific properties from  
1413 scratch.

*Example 102.* To consider a real world example, we want to use the complex multiplication method in combination with Sage to compute Secp256k1 from scratch. So by example XXX, we decided to compute an elliptic curve over a prime field  $\mathbb{F}_p$  of order  $r$  for the security parameters

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$

$$r = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

1414 which, according to example XXX gives the trace of Frobeniois  $t = 43242038656565965685242086639067$   
1415 We also decided that we want a curve of the form  $y^2 = x^3 + b$ , that is we want the parameter  $a$   
1416 to be zero. This implies, the  $j$ -invariant of our curve must be zero.

In a first step we have to find a CM-discriminant  $D$  and some integer  $v$ , such that the equation

$$4p = t^2 + |D|v^2$$

1417 is satisfied. Since we aim for a vanishing  $j$ -invariant, the first thing to try is  $D = -3$ . In this  
1418 case we can compute  $v^2 = (4p - t^2)$  and if  $v^2$  happens to be an integers that has a square root  $v$ ,  
1419 we are done. Invoking Sage we compute

```
1420 Sage: D = -3
1421 Sage: p = 1157920892373161954235709850086879078532699846656%
1422 40564039457584007908834671663
1423 Sage: r = 11579208923731619542357098500868790785283756427907%
1424 4904382605163141518161494337
1425 Sage: t = p+1-r
1426 Sage: v_sqr = (4*p - t^2)/abs(D)
1427 Sage: v_sqr.is_integer()
1428 Sage: v = sqrt(v_sqr)
1429 Sage: v.is_integer()
1430 Sage: 4*p == t^2 + abs(D)*v^2
1431 Sage: v
```

1432 So indeed the pair  $(D, v) = (-3, 303414439467246543595250775667605759171)$  solves the  
1433 equation, which tells us that there is a curve of order  $r$  over a prime field of order  $p$ , defined by  
1434 a Weierstraß equation  $y^2 = x^3 + b$  for some  $b \in \mathbb{F}_p$ . So we need to compute  $b$ .

Now for  $D = -3$  we already know that the associated Hilbert class polynomial is given by  $H_{-3}(x) = x$ , which gives the projected Hilbert class polynomial as  $H_{-3,p} = x$  and the  $j$ -invariant of our curve is guaranteed to be  $j = 0$ . Now looking into table XXX, we see that there are 6 possible cases to construct a curve with the correct order  $r$ . In order to construct the curves of those case we have to choose some arbitrary quadratic and cubic non residue. So we loop through  $\mathbb{F}_p$  to find them, invoking Sage:

add reference

```

1441 Sage: F = GF(p)
1442 Sage: for c2 in F:
1443     ....:     try: # quadratic residue
1444     ....:         _ = c2.nth_root(2)
1445     ....:     except ValueError: # quadratic non residue
1446     ....:         break
1447 Sage: c2
1448 Sage: for c3 in F:
1449     ....:     try:
1450     ....:         _ = c3.nth_root(3)
1451     ....:     except ValueError:
1452     ....:         break
1453 Sage: c3

```

So we found the quadratic non residue  $c_2 = 3$  and the cubic non residue  $c_3 = 2$ . Using those numbers we check the six cases against the the expected order  $r$  of the curve we want to synthesize:

```

1457 Sage: C1 = EllipticCurve(F, [0, 1])
1458 Sage: C1.order() == r
1459 Sage: C2 = EllipticCurve(F, [0, c2^3])
1460 Sage: C2.order() == r
1461 Sage: C3 = EllipticCurve(F, [0, c3^2])
1462 Sage: C3.order() == r
1463 Sage: C4 = EllipticCurve(F, [0, c3^2*c2^3])
1464 Sage: C4.order() == r
1465 Sage: C5 = EllipticCurve(F, [0, c3^(-2)])
1466 Sage: C5.order() == r
1467 Sage: C6 = EllipticCurve(F, [0, c3^(-2)*c2^3])
1468 Sage: C6.order() == r

```

So as expected we found an elliptic curve of the correct order  $r$  over a prime field of size  $p$ . So in principal we are done, as we have found a curve with the same basic properties as Secp256k1. However, the curve is defined by the equation

$$y^2 = x^3 + 86844066927987146567678238756515930889952488499230423029593188005931626003754$$

that use a very large parameter  $b_1$ , which might perform slow in certain algorithms. It is also not very elegant to be written down by hand. It might therefore be advantageous to find an isomorphic curve with the smallest possible parameter  $b_2$ . So in order to find such a  $b_2$ , we have to choose a quadratic residue  $d$ , such that  $b_2 = b_1 \cdot d^3$  is as small as possible. To do so we rewrite the last equation into

$$d = \sqrt[3]{\frac{b_2}{b_1}}$$



1469 and then invoke Sage to loop through values  $b_2 \in \mathbb{F}_p$  until it finds some number such that the  
 1470 quotient  $\frac{b_2}{b_1}$  has a cube root  $d$  and this cube root itself is a quadratic residue.

```

1471 Sage: b1=8684406692798714656767823875651593088995248849923042%
1472 3029593188005931626003754
1473 Sage: for b2 in F:
1474     ....:     try:
1475     ....:         d = (b2/b1).nth_root(3)
1476     ....:         try:
1477     ....:             _ = d.nth_root(2)
1478     ....:             if d != 0:
1479     ....:                 break
1480     ....:         except ValueError:
1481     ....:             pass
1482     ....:     except ValueError:
1483     ....:         pass
1484 Sage: b2

```

So indeed the smallest possible value is  $b_2 = 7$  and the defining Weierstraß equation of a curve over  $\mathbb{F}_p$  with prime order  $r$  is

$$y^2 = x^3 + 7$$

1485 which we might call secp256k1. As we have seen the complex multiplication method is power-  
 1486 ful enough to derive cryptographically secure curves like Secp256k1 from scratch.

1487 **The BLS6\_6 pen& paper curve** In this paragraph we to summarize our understanding of  
 1488 elliptic curves to derive our main pen & paper example for the rest of the book. To do so, we  
 1489 want to use the complex multiplication method, to derive a pairing friendly elliptic curve that  
 1490 has similar properties to curves that are used in actual cryptographic protocols. However, we  
 1491 design the curve specifically to be useful in pen&paper examples, which mostly means that the  
 1492 curve should contain only a few points, such that we are able to derive exhaustive addition and  
 1493 pairing tables.

1494 A well understood family of pairing friendly curves are the BLS curves (STUFF ABOUT  
 1495 THE HISTORY AND THE NAMING CONVENTION), which are derived in [XXX]. BLS  
 1496 curves are particular useful in our case if the embedding degree  $k$  satisfies  $k \equiv 6 \pmod{0}$ . Of  
 1497 course the smallest embedding degree  $k$  that satisfies this congruency, is  $k = 6$  and we therefore  
 1498 aim for a BLS6 curve as our main pen&paper example.

1499 To apply the complex multiplication method from XXX, recall that this method starts with  
 1500 a definition of the base field  $\mathbb{F}_{p^m}$  as well as the trace of Frobenius  $t$  and the order of the curve.  
 1501 If the order  $p^m + 1 - t$  is not a prime number, then what is necessary to control is the order  $r$  of  
 1502 the largest prime factor group.

In the case of BLS\_6 curves, the parameter  $m$  is chosen to be 1, which means that the curves are defined over prime fields. All relevant parameters  $p$ ,  $t$  and  $r$  are then themselves parameterized by the following three polynomials

$$\begin{aligned}
 r(x) &= \Phi_6(x) \\
 t(x) &= x + 1 \\
 q(x) &= \frac{1}{3}(x-1)^2(x^2 - x + 1) + x
 \end{aligned}$$

add refer-  
ence

add refer-  
ence

where  $\Phi_6$  is the 6-th cyclotomic polynomial and  $x \in \mathbb{N}$  is a parameter that the designer has to choose in such a way that the evaluation of  $p$ ,  $t$  and  $r$  at the point  $x$  gives integers that have the proper size to meet the security requirements of the curve that they want to design. It is then guaranteed that the complex multiplication method can be used in combination with those parameters to define an elliptic curve with CM-discriminant  $D = -3$  and embedding degree  $k = 6$  and curve equation  $y^2 = x^3 + b$  for some  $b \in \mathbb{F}_p$ .

For example if the curve should target the 128-bit security level, due to the Pholaard-rho attack (TODO) the parameter  $r$  should be prime number of at least 256 bits.

In order to design the smallest, most insecure BLS\_6 curve, we therefore have to find a parameter  $x$ , such that  $r(x)$ ,  $t(x)$  and  $q(x)$  are the smallest natural numbers, that satisfy  $q(x) > 3$  and  $r(x) > 3$ .

We therefore initiate the design process of our BLS6 curve by looking-up the 6-th cyclotomic polynomial which is  $\Phi_6 = x^2 - x + 1$  and then insert small values for  $x$  into the defining polynomials  $r, t, q$ . We get the following results:

$$\begin{array}{lll} x = 1 & (r(x), t(x), q(x)) & (1, 2, 1) \\ x = 2 & (r(x), t(x), q(x)) & (3, 3, 3) \\ x = 3 & (r(x), t(x), q(x)) & (7, 4, \frac{37}{3}) \\ x = 4 & (r(x), t(x), q(x)) & (13, 5, 43) \end{array}$$

Since  $q(1) = 1$  is not a prime number, the first  $x$  that gives a proper curve is  $x = 2$ . However, such a curve would be defined over a base field of characteristic 3 and we would rather like to avoid that. We therefore find  $x = 4$ , which defines a curve over the prime field of characteristic 43, that has a trace of Frobenius  $t = 5$  and a larger order prime group of size  $r = 13$ .

Since the prime field  $\mathbb{F}_{43}$  has 43 elements and 43's binary representation is  $43_2 = 101011$ , which are 6 digits, the name of our pen&paper curve should be BLS6\_6, since its is common behaviour to name a BLS curve by its embedding degree and the bit-length of the modulus in the base field. We call BLS6\_6 the **moon-math-curve**.

Recalling from XXX, we know that the Hasse bound implies that BLS6\_6 will contain exactly 39 elements. Since the prime factorization of 39 is  $39 = 3 \cdot 13$ , we have a "large" prime factor group of size 13 as expected and a small cofactor group of size 3. Fortunately a subgroup of order 13 is well suited for our purposes as 13 elements can be easily handled in the associated addition, scalar multiplication and pairing tables in a pen-and-paper style.

We can check that the embedding degree is indeed 6 as expected, since  $k = 6$  is the smallest number  $k$  such that  $r = 13$  divides  $43^k - 1$ .

```
Sage: for k in range(1,42): # Fermat's little theorem
.....:     if (43^k-1)%13 == 0:
.....:         break
Sage: k
```

In order to compute the defining equation  $y^2 = x^3 + ax + b$  of BLS6-6, we use the complex multiplication method as described in XXX. The goal is to find  $a, b \in \mathbb{F}_{43}$  representations, that are particularly nice to work with. The authors of XXX showed that the CM-discriminant of every BLS curve is  $D = -3$  and indeed the equation

$$\begin{aligned} 4p &= t^2 + |D|v^2 && \Rightarrow \\ 4 \cdot 43 &= 5^2 + |D|v^2 && \Rightarrow \\ 172 &= 25 + |D|v^2 && \Leftrightarrow \\ 49 &= |D|v^2 \end{aligned}$$

add reference

add reference

add reference

has the four solutions  $(D, v) \in \{(-3, -7), (-3, 7), (-49, -1), (-49, 1)\}$  if  $D$  is required to be negative, as expected. So  $D = -3$  is indeed a proper CM-discriminant and we can deduce that the parameter  $a$  has to be 0 and that the Hilbert class polynomial is given by

$$H_{-3,43}(x) = x$$

1533 This implies that the  $j$ -invariant of *BLS6\_6* is given by  $j(\text{BLS6\_6}) = 0$ . We therefore have to  
 1534 look at case XXX in table XXX to derive a parameter  $b$ . To decide the proper case for  $j_0 = 0$   
 1535 and  $D = -3$ , we therefore have to choose some arbitrary quadratic non residue  $c_2$  and cubic  
 1536 non residue  $c_3$  in  $\mathbb{F}_{43}$ . We choose  $c_2 = 5$  and  $c_3 = 36$ . We check

add refer-  
enceadd refer-  
ence

```
1537 Sage: F43 = GF(43)
1538 Sage: c2 = F43(5)
1539 .....: try: # quadratic residue
1540 .....:     c2.nth_root(2)
1541 .....: except ValueError: # quadratic non residue
1542 .....:     c2
1543 Sage: c3 = F43(36)
1544 .....: try:
1545 .....:     c3.nth_root(3)
1546 .....: except ValueError:
1547 .....:     c3
```

1548 Using those numbers we check the six possible cases from XXX against the the expected  
 1549 order 39 of the curve we want to synthesize:

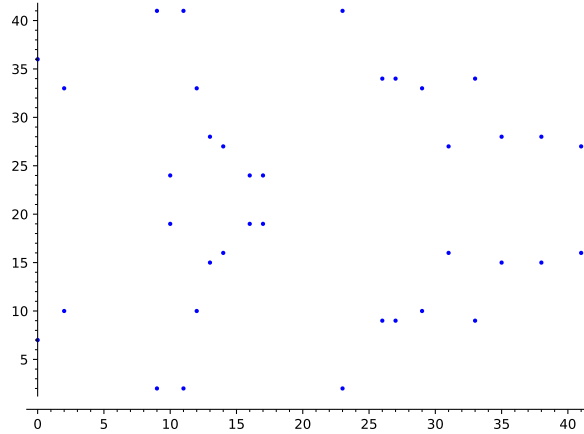
add refer-  
ence

```
1550 Sage: BLS61 = EllipticCurve(F43, [0, 1])
1551 Sage: BLS61.order() == 39
1552 Sage: BLS62 = EllipticCurve(F43, [0, c2^3])
1553 Sage: BLS62.order() == 39
1554 Sage: BLS63 = EllipticCurve(F43, [0, c3^2])
1555 Sage: BLS63.order() == 39
1556 Sage: BLS64 = EllipticCurve(F43, [0, c3^2*c2^3])
1557 Sage: BLS64.order() == 39
1558 Sage: BLS65 = EllipticCurve(F43, [0, c3^(-2)])
1559 Sage: BLS65.order() == 39
1560 Sage: BLS66 = EllipticCurve(F43, [0, c3^(-2)*c2^3])
1561 Sage: BLS66.order() == 39
1562 Sage: BLS6 = BLS63 # our BLS6 curve in the book
```

1563 So, as expected, we found an elliptic curve of the correct order 39 over a prime field of size 43,  
 1564 defined by the equation

$$\text{BLS6\_6} := \{(x, y) \mid y^2 = x^3 + 6 \text{ for all } x, y \in \mathbb{F}_{43}\} \quad (5.26)$$

1565 There are other choice for  $b$  like  $b = 10$  or  $b = 23$ , but all these curves are isomorphic and  
 1566 hence represent the same curve really but in a different way only. Since BLS6-6 only contains  
 1567 39 points it is possible to give a visual impression of the curve:



1568

1569 As we can see our curve is somewhat nice, as it does not contain self inverse points that is points  
 1570 with  $y = 0$ . It follows that the addition law can be optimized, since the branch for those cases  
 1571 can be eliminated.

1572 Summarizing the previous procedure, we have used the method of Barreto, Lynn and Scott  
 1573 to construct a pairing friendly elliptic curve of embedding degree 6. However, in order to do  
 1574 elliptic curve cryptography on this curve note that since the order of  $BLS6\_6$  is 39 its group of  
 1575 rational points is not a finite cyclic group of prime order. We therefore have to find a suitable  
 1576 subgroup as our main target and since  $39 = 13 \cdot 3$ , we know that the curve must contain a "large"  
 1577 prime order group of size 13 and a small cofactor group of order 3.

1578 It is the content of the following step to construct this group. One way to do so is to find  
 1579 a generator. We can achieve this by choosing an arbitrary element of the group that is not the  
 1580 point at infinity and then multiply that point with the cofactor of the groups order. If the result  
 1581 is not the point at infinity, the result will be a generator and if it is the point at infinity we have  
 1582 to choose a different element.

So in order to find a generator for the large order subgroup of size 13, we first notice that the cofactor of 13 is 3, since  $39 = 3 \cdot 13$ . We then need to construct an arbitrary element from  $BLS6\_6$ . To do so in a pen-and-paper style, we can choose some arbitrary  $x \in \mathbb{F}_{43}$  and see if there is some solution  $y \in \mathbb{F}_{43}$  that satisfies the defining Weierstraß equation  $y^2 = x^3 + 6$ . We choose  $x = 9$ . Then  $y = 2$  is a proper solution, since

$$\begin{aligned} y^2 &= x^3 + 6 && \Rightarrow \\ 2^2 &= 9^3 + 6 && \Leftrightarrow \\ 4 &= 4 \end{aligned}$$

1583 and this implies that  $P = (9, 2)$  is therefore a point on  $BLS6\_6$ . To see if we can project this  
 1584 point onto a generator of the large order prim group  $BLS6\_6[13]$ , we have to multiply  $P$  with  
 1585 the cofactor, that is we have to compute  $[3](9, 2)$ . After some computation (EXERCISE) we get  
 1586  $[3](9, 2) = (13, 15)$  and since this is not the point at infinity we know that  $(13, 15)$  must be a  
 1587 generator of  $BLS6\_6[13]$ . We write

$$g_{BLS6\_6[13]} = (13, 15) \tag{5.27}$$

1588 as we will need this generator in pairing computations all over the book. Since  $g_{BLS6\_6[13]}$   
 1589 is a generator, we can use it to construct the subgroup  $BLS6\_6[13]$ , by repeatedly adding the  
 1590 generator to itself. We use Sage and get

1591 Sage: `P = BLS6(9, 2)`

```

1592 Sage: Q = 3*P
1593 Sage: Q.xy()
1594 Sage: BLS6_13 = []
1595 Sage: for x in range(0,13): # cyclic of order 13
1596 .....:     P = x*Q
1597 .....:     BLS6_13.append(P)

```

Repeatedly adding a generator to itself as we just did, will generate small groups in logarithmic order with respect to the generator as explained in XXX. We therefore get the following description of the large prime order subgroup of  $BLS6\_6$ :

add reference

$$BLS6\_6[13] = \{(13,15) \rightarrow (33,34) \rightarrow (38,15) \rightarrow (35,28) \rightarrow (26,34) \rightarrow (27,34) \rightarrow (27,9) \rightarrow (26,9) \rightarrow (35,15) \rightarrow (38,28) \rightarrow (33,9) \rightarrow (13,28) \rightarrow \mathcal{O}\} \quad (5.28)$$

Having a logarithmic description of this group is tremendously helpful in pen-and-paper computations. To see that, observe that we know from XXX that there is an exponential map from the scalar field  $\mathbb{F}_{13}$  to  $BLS6\_6[13]$  with respect to our generator

add reference

$$[\cdot]_{(13,15)} : \mathbb{F}_{13} \rightarrow BLS6\_6[13] ; x \mapsto [x](13,15)$$

which generates the group in logarithmic order. So for example we have  $[1]_{(13,15)} = (13,15)$ ,  $[7]_{(13,15)} = (27,9)$  and  $[0]_{(13,15)} = \mathcal{O}$  and so on. The point for our purposes is, that we can use this representation to do computations in  $BLS6\_6[13]$  efficiently in our head using XXX. For example

add reference

$$\begin{aligned}
 (27,34) \oplus (33,9) &= [6](13,15) \oplus [11](13,15) \\
 &= [6+11](13,15) \\
 &= [4](13,15) \\
 &= (35,28)
 \end{aligned}$$

So XXX is really all we need to do computations in  $BLS6\_6[13]$  in this book efficiently. However, out of convenience, the following picture lists the entire addition table of that group. It might be useful in pen-and-paper computations:

add reference

$\oplus$	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)
$\mathcal{O}$	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)
(13,15)	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$
(33,34)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)
(38,15)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)
(35,28)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)
(26,34)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)
(27,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)
(27,9)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)
(26,9)	(26,9)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)
(35,15)	(35,15)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)
(38,28)	(38,28)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)
(33,9)	(33,9)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)
(13,28)	(13,28)	$\mathcal{O}$	(13,15)	(33,34)	(38,15)	(35,28)	(26,34)	(27,34)	(27,9)	(26,9)	(35,15)	(38,28)	(33,9)

Now that we have constructed a "large" cyclic prime order subgroup of  $BLS6\_6$  suitable for many pen-and-paper computations in elliptic curve cryptography, we have to look at how to do pairings in this context. We know that  $BLS6\_6$  is a pairing friendly curve by design, since it has

a small embedding degree  $k = 6$ . It is therefore possible to compute Weil pairings efficiently. However, in order to do so, we have to decide the groups  $\mathbb{G}_1$  and  $\mathbb{G}_2$  as explained in XXX.

add reference

Since  $BLS6\_6$  has two non-trivial subgroups, it would be possible to use any of them as the  $n$ -torsion group. However, in cryptography, the only secure choice is to use the large prime order subgroup, which in our case is  $BLS6\_6[13]$ . we therefore decide to consider the 13-torsion and define

$$\mathbb{G}_1[13] = \{(13, 15) \rightarrow (33, 34) \rightarrow (38, 15) \rightarrow (35, 28) \rightarrow (26, 34) \rightarrow (27, 34) \rightarrow (27, 9) \rightarrow (26, 9) \rightarrow (35, 15) \rightarrow (38, 28) \rightarrow (33, 9) \rightarrow (13, 28) \rightarrow \mathcal{O}\}$$

as the first argument for the Weil pairing function.

In order to construct the domain for the second argument, we need to construct  $\mathbb{G}_2[13]$ , which, according to the general theory should be defined by those elements  $P$  of the full 13-torsion group  $BLS6\_6[13]$ , that are mapped to  $43 \cdot P$  under the Frobenius endomorphism XXX.

add reference

To compute  $\mathbb{G}_2[13]$  we therefore have to find the full 13-torsion group first. To do so, we use the technique from XXX, which tells us, that the full 13-torsion can be found in the curve extension

add reference

$$BLS6\_6 := \{(x, y) \mid y^2 = x^3 + 6 \text{ for all } x, y \in \mathbb{F}_{43^6}\} \quad (5.29)$$

over the extension field  $\mathbb{F}_{43^6}$ , since the embedding degree of  $BLS6\_6$  is 6. So we have to construct  $\mathbb{F}_{43^6}$ , a field that contains 6321363049 many elements. In order to do so we use the procedure of XXX and start by choosing a non-reducible polynomial of degree 6 from the ring of polynomials  $\mathbb{F}_{43}[t]$ . We choose  $p(t) = t^6 + 6$ . Using Sage we get

add reference

```
Sage: F43 = GF(43)
Sage: F43t.<t> = F43[]
Sage: p = F43t(t^6+6)
Sage: p.is_irreducible()
Sage: F43_6.<v> = GF(43^6, name='v', modulus=p)
```

Recall from XXX that elements  $x \in \mathbb{F}_{43^6}$  can be seen as polynomials  $a_0 + a_1v + a_2v^2 + \dots + a_5v^5$  with the usual addition of polynomials and multiplication modulo  $t^6 + 6$ .

add reference

In order to compute  $\mathbb{G}_2[13]$  we first have to extend  $BLS6\_6$  to  $\mathbb{F}_{43^6}$ , that is we keep the defining equation but extend the domain from  $\mathbb{F}_{43}$  to  $\mathbb{F}_{43^6}$ . After that we have to find at least one element  $P$  from that curve, that is not the point at infinity, that is in the full 13-torsion and that satisfies the identity  $\pi(P) = [43]P$ . We can then use this element as our generator of  $\mathbb{G}_2[13]$  and construct all other elements by repeated addition to itself.

Since  $BLS6(\mathbb{F}_{43^6})$  contains 6321251664 elements, its not a good strategy to simply loop through all elements. Fortunately Sage has a way to loop through elements from the torsion group directly. We get

```
Sage: BLS6 = EllipticCurve(F43_6, [0, 6]) # curve extension
Sage: INF = BLS6(0) # point at infinity
Sage: for P in INF.division_points(13): # full 13-torsion
.....: # PI(P) == [q]P
.....:     if P.order() == 13: # exclude point at infinity
.....:         PiP = BLS6([a.frobenius() for a in P])
.....:         qP = 43*P
.....:         if PiP == qP:
.....:             break
Sage: P.xy()
```

1639 So we found an element from the full 13-torsion, that is in the Eigenspace of the Eigenvalue 43,  
 1640 which implies that it is an element of  $\mathbb{G}_2[13]$ . As  $\mathbb{G}_2[13]$  is cyclic of prime order this element  
 1641 must be a generator and we write

$$g_{\mathbb{G}_2[13]} = (7v^2, 16v^3) \quad (5.30)$$

1642 We can use this generator to compute  $\mathbb{G}_2$  is logarithmic order with respect to  $g_{\mathbb{G}_2[13]}$ . Using  
 1643 Sage we get

```
1644 Sage: Q = BLS6(7*v^2, 16*v^3)
1645 Sage: BLS6_13_2 = []
1646 Sage: for x in range(0, 13):
1647     ....:     P = x*Q
1648     ....:     BLS6_13_2.append(P)
```

$$\begin{aligned} \mathbb{G}_2 = \{ & (7v^2, 16v^3) \rightarrow (10v^2, 28v^3) \rightarrow (42v^2, 16v^3) \rightarrow (37v^2, 27v^3) \rightarrow \\ & (16v^2, 28v^3) \rightarrow (17v^2, 28v^3) \rightarrow (17v^2, 15v^3) \rightarrow (16v^2, 15v^3) \rightarrow \\ & (37v^2, 16v^3) \rightarrow (42v^2, 27v^3) \rightarrow (10v^2, 15v^3) \rightarrow (7v^2, 27v^3) \rightarrow \mathcal{O} \} \end{aligned}$$

Again, having a logarithmic description of  $\mathbb{G}_2[13]$  is tremendously helpful in pen-and-paper computations, as it reduces complicated computation in the extended curve to modular 13 arithmetics. For example

$$\begin{aligned} (17v^2, 28v^3) \oplus (10v^2, 15v^2) &= [6](7v^2, 16v^3) \oplus [11](7v^2, 16v^3) \\ &= [6 + 11](7v^2, 16v^3) \\ &= [4](7v^2, 16v^3) \\ &= (37v^2, 27v^3) \end{aligned}$$

1649 So XXX is really all we need to do computations in  $\mathbb{G}_2[13]$  in this book efficiently.

To summarize the previous steps, we have found two subgroups  $\mathbb{G}_1[13]$  as well as  $\mathbb{G}_2[13]$  suitable to do Weil pairings on *BLS6\_6* as explained in XXX. Using the logarithmic order XXX of  $\mathbb{G}_1[13]$ , the logarithmic order XXX of  $\mathbb{G}_2[13]$  and the bilinearity

$$e([k_1]g_{BLS6\_6[13]}, [k_2]g_{\mathbb{G}_2[13]}) = e(g_{BLS6\_6[13]}, g_{\mathbb{G}_2[13]})^{k_1 \cdot k_2}$$

we can do Weil pairings on *BLS6\_6* in a pen-and-paper style, observing that the Weil pairing between our two generators is given by the identity

$$e(g_{BLS6\_6[13]}, g_{\mathbb{G}_2[13]}) = 5v^5 + 16v^4 + 16v^3 + 15v^2 + 3v + 41$$

```
1650 Sage: g1 = BLS6([13, 15])
1651 Sage: g2 = BLS6([7*v^2, 16*v^3])
1652 Sage: g1.weil_pairing(g2, 13)
```

add refer-  
ence

add refer-  
ence

add refer-  
ence

add refer-  
ence



1653 **Hashing to the pairing groups** We give various constructions to hash into  $\mathbb{G}_1$  and  $\mathbb{G}_2$ .  
 1654 We start with hashing to the scalar field... TO APPEAR  
 1655 Non of these techniques work for hashing into  $\mathbb{G}_2$ . We therefore implement Pederson's  
 1656 Hash for BLS6.  
 We start with  $\mathbb{G}_1$ . Our goal is to define an 12-bit bounded hash function

$$H_1 : \{0, 1\}^{12} \rightarrow \mathbb{G}_1$$

Since  $12 = 3 \cdot 4$  we "randomly" select 4 uniformly distributed generators  $\{(38, 15), (35, 28), (27, 34), (38, 28)\}$  from  $\mathbb{G}_1$  and use the pseudo-random function from XXX. For every generator we therefore have to choose a set of 4 randomly generated invertible elements from  $\mathbb{F}_{13}$ . We choose

add reference

$$\begin{aligned} (38, 15) & : \{2, 7, 5, 9\} \\ (35, 28) & : \{11, 4, 7, 7\} \\ (27, 34) & : \{5, 3, 7, 12\} \\ (38, 28) & : \{6, 5, 1, 8\} \end{aligned}$$

So our hash function is computed like this:

$$H_1(x_{11}, x_1, \dots, x_0) = [2 \cdot 7^{x_{11}} \cdot 5^{x_{10}} \cdot 9^{x_9}](38, 15) + [11 \cdot 4^{x_8} \cdot 7^{x_7} \cdot 7^{x_6}](35, 28) + [5 \cdot 3^{x_5} \cdot 7^{x_4} \cdot 12^{x_3}](27, 34) + [6 \cdot 5^{x_2} \cdot 1^{x_1} \cdot 8^{x_0}](38, 28)$$

Note that  $a^x = 1$  whe  $x = 0$  and hence those terms can be omitted in the computation. In particular the hash of the 12-bit zero string is given by

$$\begin{aligned} \text{WRONG-ORDERING-REDO} H_1(0) &= [2](38, 15) + [11](35, 28) + [5](27, 34) + [6](38, 28) = \\ &= (27, 34) + (26, 34) + (35, 28) + (26, 9) = (33, 9) + (13, 28) = (38, 28) \end{aligned}$$

The hash of 011010101100 is given by

$$\begin{aligned} H_1(011010101100) &= \text{WRONG-ORDERING-REDO} \\ &= [2 \cdot 7^0 \cdot 5^1 \cdot 9^1](38, 15) + [11 \cdot 4^0 \cdot 7^1 \cdot 7^0](35, 28) + [5 \cdot 3^1 \cdot 7^0 \cdot 12^1](27, 34) + [6 \cdot 5^1 \cdot 1^0 \cdot 8^0](38, 28) = \\ &= [2 \cdot 5 \cdot 9](38, 15) + [11 \cdot 7](35, 28) + [5 \cdot 3 \cdot 12](27, 34) + [6 \cdot 5](38, 28) = \\ &= [12](38, 15) + [12](35, 28) + [11](27, 34) + [4](38, 28) = \end{aligned}$$

TO APPEAR

We can use the same technique to define a 12-bit bounded hash function in  $\mathbb{G}_2$ :

$$H_2 : \{0, 1\}^{12} \rightarrow \mathbb{G}_2$$

Again we "randomly" select 4 uniformly distributed generators  $\{(7v^2, 16v^3), (42v^2, 16v^3), (17v^2, 15v^3), (10v^2, 15v^3)\}$  from  $\mathbb{G}_2$  and use the pseudo-random function from XXX. For every genrator we therefore have to choose a set of 4 randomly generated invertible elements from  $\mathbb{F}_{13}$ . We choose

add reference

$$\begin{aligned} (7v^2, 16v^3) & : \{8, 4, 5, 7\} \\ (42v^2, 16v^3) & : \{12, 1, 3, 8\} \\ (17v^2, 15v^3) & : \{2, 3, 9, 11\} \\ (10v^2, 15v^3) & : \{3, 6, 9, 10\} \end{aligned}$$



So our hash function is computed like this:

$$H_1(x_{11}, x_{10}, \dots, x_0) = [8 \cdot 4^{x_{11}} \cdot 5^{x_{10}} \cdot 7^{x_9}](7v^2, 16v^3) + [12 \cdot 1^{x_8} \cdot 3^{x_7} \cdot 8^{x_6}](42v^2, 16v^3) + \\ [2 \cdot 3^{x_5} \cdot 9^{x_4} \cdot 11^{x_3}](17v^2, 15v^3) + [3 \cdot 6^{x_2} \cdot 9^{x_1} \cdot 10^{x_0}](10v^2, 15v^3)$$

We extend this to a hash function that maps unbounded bitstring to  $\mathbb{G}_2$  by precomposing with an actual hash function like *MD5* and feed the first 12 bits of its outcome into our previously defined hash function.

$$TinyMD5_{\mathbb{G}_2} : \{0, 1\}^* \rightarrow \mathbb{G}_2$$

1657 with  $TinyMD5_{\mathbb{G}_2}(s) = H_2(MD5(s)_0, \dots, MD5(s)_{11})$ . For example, since  $MD5("") =$   
 1658  $0xd41d8cd98f00b204e9800998ecf8427e$  and the binary representation of the hexadecimal  
 1659 number  $0x27e$  is  $001001111110$  we compute  $TinyMD5_{\mathbb{G}_2}$  of the empty string as  $TinyMD5_{\mathbb{G}_2}("") =$   
 1660  $H_2(MD5(s)_{11}, \dots, MD5(s)_0) = H_2(001001111110) =$

# Bibliography

- Jens Groth. On the size of pairing-based non-interactive arguments. *IACR Cryptol. ePrint Arch.*, 2016:260, 2016. URL <http://eprint.iacr.org/2016/260>.
- David Fifield. The equivalence of the computational diffie–hellman and discrete logarithm problems in certain groups, 2012. URL <https://web.stanford.edu/class/cs259c/finalpapers/dlp-cdh.pdf>.
- Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-46766-3. URL <https://fmouhart.epheme.re/Crypto-1617/TD08.pdf>.
- Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Report 2016/492, 2016. <https://ia.cr/2016/492>.