TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Curves, codes, and cryptography

Citation for published version (APA):
Peters, C. P. (2011). *Curves, codes, and cryptography*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR711052

DOI:
10.6100/IR711052

Document status and date:
Published: 01/01/2011

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Apr. 2025

Curves, Codes, and Cryptography

Copyright © 2011 by Christiane Peters.

Printed by Printservice Technische Universiteit Eindhoven.

Cover: Starfish found in Bouldin Creek, Austin, Texas. Fossils originate from the late Cretaceous, about 85 million years ago. Picture taken by Laura Hitt O'Connor at the Texas Natural Science Center, Austin, Texas. Design in cooperation with Verspaget & Bruinink.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Peters, Christiane

Curves, Codes, and Cryptography / by Christiane Peters. – Eindhoven: Technische Universiteit Eindhoven, 2011.
Proefschrift. – ISBN 978-90-386-2476-1
NUR 919
Subject heading: Cryptology
2000 Mathematics Subject Classification: 94A60, 11Y16, 11T71

Curves, Codes, and Cryptography

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 10 mei 2011 om 16.00 uur

door

Christiane Pascale Peters

geboren te Paderborn, Duitsland

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. T. Lange
en
prof.dr. D.J. Bernstein

Für meine Eltern

Angelika und Jürgen Peters

Thanks

I would like to thank many people for their support and company during my Ph.D. studies. First of all, I am most grateful to my supervisors Tanja Lange and Daniel J. Bernstein. I thank Tanja for giving me the opportunity to come to Eindhoven to do research and to get to know the (crypto) world.

I thank Tanja and Dan for letting me participate in great projects such as working on Edwards curves in cryptographic and number-theoretic settings, and for helping me find my way into the world of code-based cryptography. I am grateful for their patience and never-ending support. I highly value their many comments and suggestions on papers, proposals, and, of course, on my thesis. I very much enjoyed our discussions, the sessions with Dan as our "marketing department", and especially their hospitality at both their homes in Utrecht and Chicago.

I want to express my gratitude to Arjeh Cohen, Nicolas Sendrier, Paul Zimmermann, Andries Brouwer, Henk van Tilborg, and Johannes Buchmann for agreeing to serve on my Ph.D. committee and for reading this manuscript and giving many valuable comments.

I also thank Relinde Jurrius, Peter Schwabe, and Henk van Tilborg for careful proofreading and many comments on this thesis. A big thanks also to Jan-Willem Knopper for his LaTex support.

I want to thank my co-authors Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, Ruben Niederhagen, Peter Schwabe, and Henk van Tilborg for their fruitful collaboration.

I am very grateful for getting the opportunity to go on several research visits. I very much enjoyed the discussions at INRIA Nancy with the (former) CACAO team and in particular with Paul Zimmermann who encouraged me to look into Suyama constructions for Edwards curves leading to the beautiful and highly efficient twisted Edwards curves. I thank David Lubicz and Emmanuel Mayer for inviting me to Rennes for lecturing on Edwards curves and a hiking tour to Mont St Michel. I want to thank Nicolas Sendrier for the invitation to INRIA Rocquencourt to discuss information-set-decoding bounds and the FSB hash function. I thank Johannes Buchmann and Pierre-Louis Cayrel for inviting me to Darmstadt for interesting and motivating discussions.

I am very glad that I got to know Laura Hitt O'Connor with whom I had a great time during her stay in Eindhoven in February 2008 and during the ECC conferences. I am very grateful that she found the time to take starfish pictures for me.

I had a great time in Eindhoven and I am going to miss many people here. The coding and cryptology group at TU/e was like a second family for me. After all the travelling I always looked forward to joining the next tea break to catch up with the wonderful people working here.
Henk van Tilborg is the heart of the group. I am almost glad that I will leave before he does because I cannot imagine being at TU/e without having Henk around for a chat, a heated discussion, or a bike ride along the Boschdijk after work. I would like to thank him for all his support and help.
Unfortunately, I had to see Anita Klooster leave our group. I am still touched by her trying to protect me from everything she thought could do any harm to me. It was always a pleasure dropping by her office. I thank Rianne van Lieshout for taking over from Anita. I very much enjoyed the many chats I had with her.
I would like to thank Peter Birkner, Gaëtan Bisson, Dion Boesten, Mayla Brusò, Elisa Costante, Yael Fleischmann, Çiçek Güven, Maxim Hendriks, Sebastiaan de Hoogh, Tanya Ignatenko, Ellen Jochemsz, Relinde Jurrius, Mehmet Kiraz, Jan-Willem Knopper, Mikkel Kroigaard, Peter van Liesdonk, Michael Naehrig, Ruben Niederhagen, Jan-Jaap Osterwijk, Jing Pan, Bruno Pontes Soares Rocha, Reza Rezaeian Farashahi, Peter Schwabe, Andrey Sidorenko, Antonino Simone, Daniel Trivellato, Meilof Veeningen, Rikko Verrijzer, José Villegas, and Shona Yu for their company. In particular, I want to thank my "Ph.D. siblings" Peter, Gaëtan, and Michael for a great time, especially when travelling together. When being "at home" I very much enjoyed the company of my island mates Peter and Sebastiaan. I also enjoyed the conversations with many other people on the 9th and 10th floor, especially Bram van Asch, Jolande Matthijsse, and Nicolas Zannone.
I am very happy having found great friends in Eindhoven. Let me mention Çiçek and Tanır, Carmen and Iman, Mayla and Antonino, Shona, Peter vL, Jeroen and Bianca, my "German" connection Daniela, Andreas, Matthias, Stephan, Trea and Christoph, and my "sportmaatjes" Siebe, Ruth, and Marrit.
I also want to thank my faithful friends from Paderborn and Bielefeld, Annika, Birthe, Imke, Jonas, Julia, Peter, and Tommy, who all found the way to Eindhoven.
Finally, I want to thank my parents and my brother Thomas for their love, encouragement, and endless support.

Contents

Introduction

Elliptic *curves* and error-correcting *codes* are the mathematical objects investigated in this thesis for *cryptographic* applications. The main focus lies on *public-key cryptography* but also a code-based hash function is investigated. Public-key cryptography was invented by Diffie and Hellman [DH76] in 1976 with the goal to remove the need for in-person meetings or trusted couriers to exchange secret keys. While symmetric cryptography uses the same key for encryption and decryption, public-key cryptography uses a key pair consisting of a *public key* used for encryption and a *private key* used for decryption. In order to generate lots of possible key pairs mathematical one-way functions are used —functions which are easy to compute but hard to invert. In practice a sender can efficiently compute a ciphertext given the public key, but only the holder of the private key can use the hidden information for decryption. Parameters for public-key cryptography need to be chosen in a way that encryption and decryption can be carried out very fast. Simultaneously, those parameters have to guarantee that it is computationally infeasible to retrieve the original message from the ciphertext, or even worse, the private key from the public key.
Parameters for cryptography are chosen to provide *b-bit security* against the best attack known. This means that given the public key and public system parameters it takes at least 2^b bit operations to retrieve the original message from a given ciphertext; or in the context of the hash function that it takes at least 2^b bit operations to find a collision. The encryption and decryption algorithms in this thesis are mostly text-book versions. Understanding the underlying mathematical problems and structures is a fundamental object of this thesis. This thesis does not investigate protocols trying to provide security against malicious attackers who exploit (partial) knowledge on e.g., ciphertexts or private keys. Those protocols can be added as another layer to strengthen the security of the schemes investigated here.

Elliptic-curve cryptography

Cryptography based on groups was introduced by Diffie and Hellman [DH76] in 1976. Diffie and Hellman invented a protocol that allows two users to compute a common key in a finite abelian group via an insecure channel. The key-exchange protocol is based on the *discrete-logarithm problem* (DLP) which is, given a finitely generated group G and two elements $g, h \in G$, to determine an integer x such that

$h = g^x$. Miller [Mil86] and Koblitz [Kob87] independently proposed to use for G the group $E(\mathbb{F}_q)$, the rational points of an elliptic curve E over $\mathbb{F}_q$.
The basic operation in *elliptic-curve cryptography* (ECC) is the multiplication-by-m map $[m] : E \to E$ sending a point P to $P+\cdots+P$, the sum containing m summands. The discrete-logarithm problem on an elliptic curve E is to determine an integer k for two given points P and Q on E such that $Q = [k]P$.
The motivation for using elliptic curves over a finite field rather than the multiplicative group $\mathbb{F}_q^*$ is that the DLP on elliptic curves is much harder. The best known algorithms to solve the DLP on an elliptic curve over $\mathbb{F}_q$ take time exponential in the field size $\log_2 q$ whereas index-calculus attacks solve the DLP on $\mathbb{F}_q^*$ in time sub-exponential in $\log_2 q$.
The *ECRYPT-II Yearly Report on Algorithms and Keysizes (2009–2010)* [ECR09] recommends the following field sizes to provide 128-bit security:

- If the group used is $\mathbb{F}_q^*$ then the field size q should be 3248 bits.
- If the group used is $E(\mathbb{F}_q)$ then the field size q should be 256 bits.

Elliptic curves are used for realizing key exchange, digital signatures and cryptography on small handheld devices such as PDAs, smart cards, etc. Current research investigates both finite-field arithmetic and arithmetic for efficient implementation of elliptic-curve cryptography on those devices. This thesis uses elliptic curves in Edwards form and twisted Edwards form and shows how these achieve new speed results; e.g., for cryptanalytic applications such as the Elliptic-Curve Method for integer factorization.

Code-based cryptography

The most prominent example of a public-key cryptosystem is the protocol by Rivest, Shamir and Adleman [RSA78]. The *RSA* protocol is used in e.g., the https protocol on the Internet and is based on the hardness of factoring integers. RSA has received a lot of attention and it is well understood how to choose parameters that are secure against attacks using current computer platforms.
Research in quantum computation showed that quantum computers would dramatically change the landscape: the problem of factoring integers could be solved in polynomial time on a quantum computer using Shor's algorithm [Sho94] while on conventional computers the running time of the best algorithm is subexponential and superpolynomial. Similarly the elliptic-curve discrete logarithm problem will also have polynomial-time solutions on quantum computers. This does not mean that quantum computers will bring an end to secure communication but it does mean that other public-key cryptosystems need to take the place of RSA and elliptic-curve cryptography.
The area of *post-quantum cryptography* studies cryptosystems that are secure against attacks by conventional computers and quantum computers. One promising candidate is *code-based cryptography*. The basic idea was published by Robert J. McEliece

[McE78] in 1978. Encryption in McEliece's system is remarkably fast. The sender simply multiplies the information vector with a matrix and adds some errors. The receiver, having generated the code by secretly transforming a Goppa code, can use standard Goppa-code decoders to correct the errors and recover the plaintext.

The security of the McEliece cryptosystem relies on the fact that the published code does not come with any known structure. An attacker is faced with the *classical decoding problem* which was proven NP-complete by Berlekamp, McEliece, and van Tilborg [BMvT78] for binary codes. The classical decoding problem is assumed to be hard on average.

An attacker does not know the secret code and thus has to decode a random-looking code without any obvious structure. There are currently no subexponential decoding methods known to attack the original McEliece cryptosystem. The best known *generic* attacks which do not exploit any code structure rely on information-set decoding, an approach introduced by Prange in [Pra62]. The idea is to find a set of coordinates of a garbled vector which are error-free and such that the restriction of the code's generator matrix to these positions is invertible. Then, the original message can be computed by multiplying those coordinates of the encrypted vector by the inverse of the submatrix.

The main drawback of the McEliece cryptosystem is the large public-key size. For 128-bit security the best known attacks force a key size of 1537536 bits which is around 192192 bytes. Of course, any off-the-shelf PC can store millions of such keys and CPUs can easily handle the McEliece cryptosystem. The problem lies in small devices. There are in fact implementations on embedded devices. Eisenbarth, Güneysu, Heyse and Paar implemented an 80-bit secure instance of the McEliece cryptosystem on a 8-bit AVR microcontroller and on a Xilinx Spartan-3AN FPGA [EGHP09]. However, this is still ongoing research and the McEliece cryptosystem currently cannot compete with RSA keys (3248 bits) and keys for elliptic-curve cryptography (256 bits) when aiming at 128-bit security in the pre-quantum world.

The main objective in code-based cryptography is to reduce key sizes and to further investigate the security of not only of the encryption scheme but also of other code-based cryptographic applications such as the code-based hash function FSB [AFS03, AFS05, AFG+09]. The main idea behind reducing key sizes is to find alternatives to McEliece's choice of classical Goppa codes. Many suggestions have been broken as the proposed codes revealed too much structure. Most recently, [GL10] and [FOPT10] broke many instances of [MB09]. This thesis discusses "wild Goppa codes" as an alternative choice.

On the quantum side Bernstein describes a quantum information-set decoding attack in [Ber10b]. The attack uses Grover's quantum root-finding algorithm [Gro96, Gro97] and forces the McEliece key size to quadruple in order to thwart this attack. This thesis concentrates on attacks for current computer platforms and assumes, as in recent articles on factorization and discrete logarithms, e.g., [JL06] and [JLSV06], that large quantum computers currently do not exist.

Overview

Part I of this thesis consists of Chapters 1–3 and deals with elliptic-curve cryptography. Part II consists of Chapters 4–9 and covers various topics from code-based cryptography.

In more detail, Chapter 1 gives the background on elliptic curves in Edwards and twisted Edwards form. Twisted Edwards curves were developed in joint work with Bernstein, Birkner, Joye, and Lange in [BBJ+08]. Moreover, fast formulas for computing the 3- and 5-fold on an Edwards curve which were published as joint work with Bernstein, Birkner, and Lange in [BBLP07] are presented in this chapter.

Chapter 2 discusses elliptic-curve single-scalar multiplication. In particular, the use of double bases using Edwards curves is investigated. It turns out that double bases are a useful tool for curves in, e.g., Jacobian coordinates but for inverted Edwards coordinates single-base chains are a better choice. The results in this chapter are joint work with Bernstein, Birkner, and Lange and appeared in [BBLP07].

Chapter 3 presents EECM, the Elliptic-Curve Method of Factorization using Edwards curves which is joint work with Bernstein, Birkner, and Lange and which was published as [BBLP08].

Chapter 4 provides the reader with background on error-correcting codes for cryptography. The McEliece cryptosystem and the Niederreiter cryptosystem are introduced and possible attacks are discussed.

Chapter 5 discusses information-set decoding, a generic attack against the McEliece cryptosystem to retrieve the original ciphertext when given a McEliece public key of a binary code. The chapter also presents the successful attack on the original McEliece parameters which was developed and carried out together with Bernstein and Lange. Implementations such as [EGHP09] used the parameter suggestions provided here. The results in this chapter appeared in [BLP08]. Moreover, this chapter carries out an asymptotic analysis of information-set decoding. It shows that Stern's algorithm is superior to plain information-set decoding, also with the improvement by Lee–Brickell. This is joint work with Bernstein, Lange, and van Tilborg and appeared in [BLPvT09].

Chapter 6 generalizes the methods discussed in Chapter 5 to arbitrary finite fields. It is shown that codes over $\mathbb{F}_{31}$ offer advantages in key sizes compared to codes over $\mathbb{F}_2$ while maintaining the same security level against all attacks known and in particular against the attack outlined in this chapter. The results in this chapter were published as [Pet10].

Chapter 7 pursues the topic of minimizing keys by using codes over non-binary fields. The "Wild McEliece cryptosystem" is proposed, a variant of McEliece's cryptosystem using "wild Goppa codes" which are analyzed in this chapter. An efficient decoding algorithm as well as parameters for 128-bit security are proposed in order to outline the advantages over the classical system or the system with codes as proposed in Chapter 6. The results in this chapter are joint work with Bernstein and Lange and appeared in [BLP11].

Chapter 8 presents "ball-collision decoding", a generic decoding attack against the

McEliece cryptosystem with binary codes, which asymptotically beats the attacks presented in Chapter 5. A detailed asymptotic analysis is presented. The results in this chapter build on joint work with Bernstein and Lange which was published as [BLP10].
Chapter 9 describes the code-based hash function FSB which was submitted to NIST's cryptographic hash function competition. This is the only chapter dealing with symmetric cryptography. The main focus lies on applying Wagner's generalized birthday attack to find a collision in the compression function of FSB_{48}, a training case submitted by the FSB designers in their proposal [AFG+09]. The results in this chapter are joint work with Bernstein, Lange, Niederhagen, and Schwabe and appeared in [BLN+09].

Part I

Elliptic-curve cryptography using Edwards curves

Chapter 1

Edwards curves and twisted Edwards curves

Elliptic curves have been studied for centuries. An *elliptic curve* over a field k is a non-singular absolutely irreducible curve of genus 1 with a k-rational point. Many books and articles introduce elliptic curves as non-singular curves which can be written in Weierstrass form $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ with $a_1, a_2, a_3, a_4, a_6 \in k$. The two definitions are equivalent due to the Riemann-Roch Theorem; see e.g., [FL05, Theorem 4.106], [Sil86, Theorem II.5.4], or [Sti09, Theorem I.5.15]. The elliptic-curve group law is explained using the chord-and-tangent method which is an explicit version of divisor-class arithmetic. This thesis deviates from this approach and directly starts with elliptic curves in Edwards form after a small detour to the unit circle.

For a reader looking for a broader background on elliptic-curve cryptography: the book by Silverman [Sil86] gives a general introduction to elliptic curves. The *Handbook of Elliptic and Hyperelliptic Curve Cryptography* [CFD05], in particular [FL05], gives an introduction to elliptic curves used for cryptography.

Note that this thesis only considers elliptic curves over non-binary fields. For readers interested in the binary case we refer to the article by Bernstein, Lange, and Rezaeian Farashahi who investigated binary Edwards curves in [BLF08].

First of all, the necessary background for elliptic curves in Edwards form is presented which mainly builds on the articles by Edwards [Edw07] and Bernstein and Lange [BL07b]. Second, this chapter introduces "twisted Edwards curves" which were developed in joint work with Bernstein, Birkner, Joye, and Lange and published as [BBJ+08]. This chapter also presents related work by Bernstein and Lange on inverted Edwards curves and completed Edwards curves [BL07c, BL10] as well as Hisil et al.'s extended formulas and dual addition law for twisted Edwards curves [HWCD08]. Moreover, this chapter provides the background for the following two chapters and are therefore taken in parts from the articles [BBLP07] and [BBLP08] which are both joint work with Bernstein, Birkner, and Lange.

This chapter is organized as follows:

- The preliminaries section 1.1 gives the definitions and addition laws of the clock group and of Edwards curves. This is a general survey on Edwards curves.

- Section 1.2 presents twisted Edwards curves. The definition is given in Section 1.2.1 and the two addition laws in Section 1.2.2. Section 1.2.3 discusses different coordinate systems. Sections 1.2.4 and 1.2.5 are essentially taken from [BBLP08] and describe the group structure of twisted Edwards curves and give a characterization of points of small order; Section 1.2.6 is taken from [BBJ+08] and relates twisted Edwards curves and Montgomery curves.
- Section 1.3 deals with arithmetic on elliptic curves in (twisted) Edwards form. Efficient formulas from [BL07b], [BL07c], [HWCD08], [BBJ+08], [BBLP07] are presented. Sections 1.3.3, 1.3.4, and 1.3.5 are taken in parts from [BBLP08, Section 2].
- Sections 1.3.6 and 1.3.7 present tripling and quintupling formulas for Edwards curves; the sections are essentially taken from [BBLP07] which is joint work with Bernstein, Birkner and Lange.

1.1 Preliminaries

This section introduces Edwards curves and twisted Edwards curves. First the *clock group* is studied in order to motivate the view of elliptic curves in Edwards form.

1.1.1 The clock

We study the clock group by considering the unit circle over the real numbers, i.e., all tuples $(x, y) \in \mathbb{R}^2$ with $x^2 + y^2 = 1$. Each line through the origin and a point (x_1, y_1) on the unit circle makes an angle α_1 from the positive y-axis in the clockwise direction. In particular, we can rewrite (x_1, y_1) as $(\sin\alpha_1, \cos\alpha_1)$. Two points (x_1, y_1) and (x_2, y_2) on the unit circle are added by adding their corresponding angles α_1 and α_2. The well-known addition formulas for sine and cosine,

$$\begin{aligned}\sin(\alpha_1 + \alpha_2) &= \sin\alpha_1\cos\alpha_2 + \cos\alpha_1\sin\alpha_2,\\ \cos(\alpha_1 + \alpha_2) &= \cos\alpha_1\cos\alpha_2 - \sin\alpha_1\sin\alpha_2,\end{aligned}$$

lead to the following addition law:

$$(x_1, y_1), (x_2, y_2) \mapsto (x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)\,. \tag{1.1}$$

More generally, consider a field k whose characteristic does not equal 2 and consider all tuples $(x, y) \in k^2$ satisfying $x^2 + y^2 = 1$. It is easily checked that (1.1) defines a group law on the unit circle, where $\mathcal{O} = (0, 1)$ is the neutral element and each point (x_1, y_1) has inverse $(-x_1, y_1)$. Hence the elements in $k \times k$ lying on the unit circle together with (1.1) form a commutative group C which is called the *clock group*.
For cryptographic applications consider the group C over $k = \mathbb{F}_q$. Geometrically the unit circle has genus 0 and corresponds to the projective line $\mathbb{P}^1(\mathbb{F}_q)$. Solving the discrete-logarithm problem for the clock group corresponds to solving the DLP

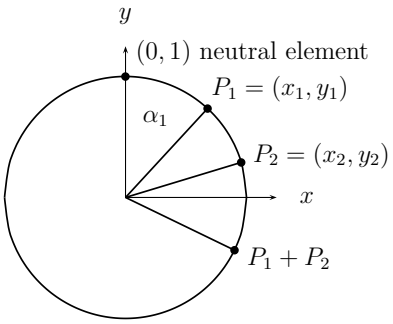

Figure 1.1: Clock addition.

in a subgroup of $\mathbb{F}_{q^2}^*$. Recall that the ECRYPT-II recommendations [ECR09] advise the underlying field to be of size around 2^{3248} to achieve 128-bit security against index-calculus attacks. In this case q should be a 1624-bit prime or prime power and -1 should be a nonsquare so that the group cannot be embedded into $\mathbb{F}_q^*$.
From now on this thesis deals with elliptic curves since in general there are no subexponential attacks for the DLP known for the genus-1 case. The addition law for the clock group will reappear in a slightly more complicated form when looking at elliptic curves in Edwards form.

1.1.2 Edwards curves

In 2007 Edwards [Edw07], generalizing an example from Euler and Gauss, introduced an addition law for the curves $x^2+y^2 = c^2(1+x^2y^2)$ over non-binary fields. Edwards showed that every elliptic curve can be expressed in this form over a small finite extension of the underlying field.

Definition 1.1. Let k be a field with char $\neq 2$. An *elliptic curve in Edwards form*, or simply *Edwards curve*, over k is given by an equation

$$x^2 + y^2 = 1 + dx^2y^2, \tag{1.2}$$

where $d \in k \setminus \{0, 1\}$.

Remark 1.2. Form (1.2) is due to Bernstein and Lange [BL07b] who generalized the addition law in [Edw07] to work for elliptic curves with equation $x^2 + y^2 = c^2(1 + dx^2y^2)$ and showed that they form a bigger class over a finite field than $x^2 + y^2 = c^2(1 + x^2y^2)$. If $d = \bar{d}\bar{c}^4$ then $(\bar{x}, \bar{y}) \mapsto (\bar{c}x, \bar{c}y)$ constitutes an isomorphism between $x^2 + y^2 = 1 + dx^2y^2$ and $\bar{x}^2 + \bar{y}^2 = \bar{c}^2(1 + \bar{d}\bar{x}^2\bar{y}^2)$.

Remark 1.3. Edwards curves are indeed elliptic curves. Though at a first glance there are two non-singular points at infinity; those can be resolved and the blowup over the extension field $k(\sqrt{d})$ is non-singular; see e.g., [BL10].

Remark 1.4. If d equals 0 then one gets the unit circle; see the previous subsection. If $d = 1$ then the equation $x^2 + y^2 = 1 + x^2y^2$ splits and describes four lines. In both cases equation (1.2) does not describe an elliptic curve.

Remark 1.5. The Edwards curve with coefficient d has j-invariant $16(1 + 14d + d^2)^3/d(1 - d)^4$.

Theorem 1.6 (Edwards addition law)**.** *Let $(x_1, y_1), (x_2, y_2)$ be points on the Edwards curve $E_d : x^2 + y^2 = 1 + dx^2y^2$. The sum of these points on E_d is*

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \right). \tag{1.3}$$

The neutral element is $(0, 1)$, and the negative of (x_1, y_1) is $(-x_1, y_1)$.

This addition law was proven correct in [BL07b, Section 3].

Remark 1.7. The addition law (1.3) is *strongly unified*: i.e., it can also be used to double a point.

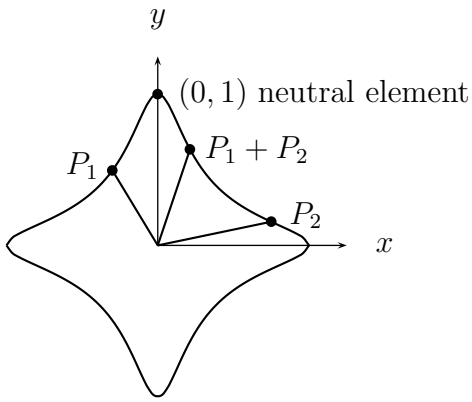

Figure 1.2: Adding two points on the Edwards curve E_{-30} over $\mathbb{R}$.

Remark 1.8. The point $(0, -1)$ has order 2. The points $(1, 0)$ and $(-1, 0)$ have order 4.

Remark 1.9. Let d be a nonsquare in k. Then by [BL07b, Theorem 3.3] the Edwards addition law is *complete*, i.e., there are no inputs (x_1, y_1), (x_2, y_2) for which the denominators $1 - dx_1x_2y_1y_2$, $1 + dx_1x_2y_1y_2$ are zero. In particular, if d is not a square all rational points of an Edwards curve are affine points and the addition is defined for any two rational points; see Remark 1.3. This means that the affine points form a group if d is not a square.

Remark 1.10 (Birational equivalence)**.** Theorem 2.1 in [BL07b] shows that any elliptic curve E having a point of order 4 over k can be written as an Edwards curve E_d with equation (1.2) using a few rational transformations. In this case the two curves E and E_d are *birationally equivalent*, i.e., there exist rational maps $\phi : E \longrightarrow E_d$ and $\psi : E_d \longrightarrow E$ with the property that $\psi \circ \phi$ is the identity map on E for all but finitely many points and $\phi \circ \psi$ is the identity map on E_d for all but finitely many points. Note that birationally equivalent should not be confused with "isomorphic" in the sense of isomorphisms between algebraic varieties. However, the rational maps ϕ and ψ establish an isomorphism between the function fields of E and E_d.

Section 1.2.6 proves a similar result for twisted Edwards curves

Remark 1.11. An elliptic curve which does not have a point of order 4 over k admits a point of order 4 over a finite extension K of k. Then there is a birational map defined over K to an Edwards curve defined over K.

1.2 Twisted Edwards curves

This section introduces twisted Edwards curves and relates them to Edwards curves as defined in the previous section. "Twisted Edwards curves" first appeared in [BBJ+08] which is joint work with Bernstein, Birkner, Joye, and Lange. This section contains the results presented in [BBJ+08]. Moreover, this section contains the characterization of small points on twisted Edwards curves which is part of [BBLP08] which is joint work with Bernstein, Birkner, and Lange.

1.2.1 Introduction of twisted Edwards curves

The existence of points of order 4 restricts the number of elliptic curves in Edwards form over k. The set of Edwards curves can be embedded into a larger set of elliptic curves of a similar shape by introducing twisted Edwards curves. Recall the definition of a quadratic twist of an elliptic curve:

Definition 1.12. Let E and E' be elliptic curves which are defined over a field k. The curve E' is called a *twist* of E if E and E' are isomorphic over the algebraic closure $\bar{k}$. The curve E' is called a *quadratic twist* of E if there is an isomorphism from E to E' which is defined over a quadratic extension of k .

This thesis only considers quadratic twists which most of the time simply will be called *twists*.

Definition 1.13 (Twisted Edwards curve)**.** Let a and d be two nonzero distinct elements in a non-binary field k. The *twisted Edwards curve with coefficients a and d* is the curve

$$E_{\mathrm{E},a,d} : ax^2 + y^2 = 1 + dx^2y^2.$$

An *Edwards curve* is a twisted Edwards curve with $a = 1$.

The subscript "E" in $E_{\mathrm{E},a,d}$ stands for Edwards. In Section 1.2.6 we will show that every twisted Edwards curve is birationally equivalent to an elliptic curve in Montgomery form, and vice versa; those curves in Montgomery form will be denoted by $E_{\mathrm{M},A,B}$.

Remark 1.14. The elliptic curve $E_{\mathrm{E},a,d}$ has j-invariant $16(a^2+14ad+d^2)^3/ad(a-d)^4$.

Remark 1.15 (Twisted Edwards curves as twists of Edwards curves)**.** The twisted Edwards curve $E_{\mathrm{E},a,d} : ax^2 + y^2 = 1 + dx^2y^2$ is a quadratic twist of the Edwards curve $E_{\mathrm{E},1,d/a} : \bar{x}^2 + \bar{y}^2 = 1 + (d/a)\bar{x}^2\bar{y}^2$. The map $(\bar{x}, \bar{y}) \mapsto (x, y) = (\bar{x}/\sqrt{a}, \bar{y})$ is an isomorphism from $E_{\mathrm{E},1,d/a}$ to $E_{\mathrm{E},a,d}$ over $k(\sqrt{a})$. If a is a square in k then $E_{\mathrm{E},a,d}$ is isomorphic to $E_{\mathrm{E},1,d/a}$ over k.

Remark 1.16. More generally, $E_{\mathrm{E},a,d}$ is a quadratic twist of $E_{\mathrm{E},\bar{a},\bar{d}}$ for any $\bar{a}, \bar{d}$ satisfying $\bar{d}/\bar{a} = d/a$. Conversely, every quadratic twist of a twisted Edwards curve is isomorphic to a twisted Edwards curve; i.e., the set of twisted Edwards curves is invariant under quadratic twists.

Remark 1.17. The twisted Edwards curve $E_{\mathrm{E},a,d} : ax^2 + y^2 = 1 + dx^2y^2$ is a quadratic twist of (actually is birationally equivalent to) the twisted Edwards curve $E_{\mathrm{E},d,a} : d\bar{x}^2 + \bar{y}^2 = 1 + a\bar{x}^2\bar{y}^2$. The maps $(\bar{x}, \bar{y}) \mapsto (\bar{x}, 1/\bar{y})$ and $(x, y) \mapsto= (x, 1/y)$ yield a birational equivalence from $E_{\mathrm{E},d,a}$ to $E_{\mathrm{E},a,d}$. More generally, $E_{\mathrm{E},a,d}$ is a quadratic twist of $E_{\mathrm{E},\bar{a},\bar{d}}$ for any $\bar{a}, \bar{d}$ satisfying $\bar{d}/\bar{a} = a/d$. This generalizes the known fact, used in [BL07b, proof of Theorem 2.1], that $E_{\mathrm{E},1,d}$ is a quadratic twist of $E_{\mathrm{E},1,1/d}$ having the same addition law.

1.2.2 The twisted Edwards addition law

The group law for (twisted) Edwards curves can be stated in two versions. The first one appeared in [BBJ+08] and the second in [HWCD08].

Theorem 1.18 (The twisted Edwards addition law)**.** *Let $(x_1, y_1), (x_2, y_2)$ be points on the twisted Edwards curve $E_{\mathrm{E},a,d} : ax^2 + y^2 = 1 + dx^2y^2$. The sum of these points on $E_{\mathrm{E},a,d}$ is*

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\right).$$

The neutral element is $(0, 1)$, and the negative of (x_1, y_1) is $(-x_1, y_1)$.

For the correctness of the addition law observe that it coincides with the Edwards addition law on $\bar{x}^2 + y^2 = 1 + (d/a)\bar{x}^2y^2$ with $\bar{x} = \sqrt{a}x$ which is proven correct in [BL07b, Section 3].

Remark 1.19. These formulas also work for doubling. These formulas are complete (i.e., have no exceptional cases) if a is a square in k and d is a nonsquare in k. We note that the isomorphism from $E_{\mathrm{E},a,d}$ to $E_{\mathrm{E},1,d/a}$ preserves the group law; if d/a is a nonsquare in k then the completeness follows from [BL07b, Theorem 3.1] which showed that the Edwards addition law is complete on $E_{\mathrm{E},1,d'}$ if d' is a nonsquare.

Hisil, Wong, Carter, and Dawson in [HWCD08] substituted the coefficients a and d in the twisted Edwards addition law by the curve equation and achieved degree-2 polynomials in the denominators.

Definition 1.20. The *dual addition law*

$$(x_1, y_1), (x_2, y_2) \mapsto \left(\frac{x_1y_1 + x_2y_2}{y_1y_2 + ax_1x_2}, \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2} \right)$$

computes the sum of two points $(x_1, y_1) \neq (x_2, y_2)$ on the twisted Edwards curve $E_{\mathrm{E},a,d}$.

Remark 1.21. This dual addition law produces the same output as the Edwards addition law when both are defined, but the exceptional cases are different. The dual addition law never works for doublings: if $(x_1, y_1) = (x_2, y_2)$ then the second output coordinate $(x_1y_1 - x_2y_2)/(x_1y_2 - y_1x_2)$ is $0/0$.

1.2.3 Projective twisted Edwards curves

A polynomial f in $k[x, y]$ is called *homogeneous* if all monomials have the same total degree. Given $f(x, y) \in k[x, y]$ of total degree δ one obtains a *homogenized* polynomial $f^{\mathrm{hom}} \in k[X, Y, Z]$ as $f^{\mathrm{hom}}(X, Y, Z) = Z^\delta f(X/Z, Y/Z)$.
Consider the homogenized twisted Edwards equation

$$E^{\mathrm{hom}}_{\mathrm{E},a,d} : (aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2.$$

A projective point $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ on $E^{\mathrm{hom}}_{\mathrm{E},a,d}$ corresponds to the affine point $(X_1/Z_1, Y_1/Z_1)$ on $E_{\mathrm{E},a,d} : ax^2+y^2 = 1+dx^2y^2$. The neutral element on $E^{\mathrm{hom}}_{\mathrm{E},a,d}$ is $\mathcal{O}^{\mathrm{hom}} = (0 : 1 : 1)$ and the inverse of a point $(X_1 : Y_1 : Z_1)$ is $(-X_1 : Y_1 : Z_1)$. Two points $(X_1 : Y_1 : Z_1)$, $(X_2 : Y_2 : Z_2)$ are equivalent if there is a nonzero scalar λ such that $(X_1 : Y_1 : Z_1) = (\lambda X_2 : \lambda Y_2 : \lambda Z_2)$. Hence one can remove denominators in the addition formulas by multiplying with the least common multiple of the denominators of the x and y coordinate.
Arithmetic in projective coordinates is more efficient than arithmetic in affine coordinates as will be discussed in Section 1.3.
In order to achieve faster addition formulas for Edwards curves Bernstein and Lange [BL07c] proposed an alternative way of representing Edwards curves in projective coordinates. The article [BBJ+08] generalized their "inverted coordinates" to twisted Edwards curves.

Definition 1.22 (Inverted twisted coordinates)**.** A point $(X_1 : Y_1 : Z_1)$ with $X_1Y_1Z_1 \neq 0$ on the projective curve

$$(X^2 + aY^2)Z^2 = X^2Y^2 + dZ^4$$

corresponds to $(Z_1/X_1, Z_1/Y_1)$ on the twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$. If $a = 1$ these coordinates are simply called *inverted coordinates*.

For contrast we refer to the former projective coordinates as *standard (twisted) Edwards coordinates*.

Remark 1.23. A point $(X_1 : Y_1 : Z_1)$ on the projective twisted Edwards curve $(aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2$ can be converted to inverted twisted Edwards coordinates by computing $(Y_1Z_1 : X_1Z_1 : X_1Y_1)$. Similarly, a point $(\bar{X}_1 : \bar{Y}_1 : \bar{Z}_1)$ on the inverted twisted Edwards curve $(\bar{X}^2 + a\bar{Y}^2)\bar{Z}^2 = \bar{X}^2\bar{Y}^2 + d\bar{Z}^4$ can be converted to projective twisted Edwards coordinates by computing $(\bar{Y}_1\bar{Z}_1 : \bar{X}_1\bar{Z}_1 : \bar{X}_1\bar{Y}_1)$.

Remark 1.24 (Restriction)**.** There are two extra points if d is a square, namely $(\pm\sqrt{d} : 0 : 1)$; two extra points if d/a is a square, namely $(0 : \pm\sqrt{d/a} : 1)$; and two singular points at infinity, namely $(0 : 1 : 0)$ and $(1 : 0 : 0)$. Following [BL07c] we exclude these points with the restriction $X_1Y_1Z_1 \neq 0$ in order to connect to twisted Edwards curves and maintain an almost complete addition law. With the restriction $X_1Y_1Z_1 \neq 0$ it is not possible to represent the affine points $(0, \pm 1)$ (and $(\pm 1, 0)$ for $a = 1$) in inverted twisted coordinates; in particular, the neutral element is excluded. These special points have to be handled differently as discussed in [BL07c].

For arithmetic in inverted twisted coordinates see Section 1.3.4.
Again motivated by faster arithmetic (see Section 1.3.5) Hisil et al. introduced a fourth variable for the projective twisted Edwards curve.

Definition 1.25. The *extended* twisted Edwards curve is given by

$$\big\{(X : Y : Z : T) \in \mathbb{P}^3 : aX^2 + Y^2 = Z^2 + dT^2 \text{ and } XY = ZT\big\}.$$

Remark 1.26. The extended points are the affine points (x_1, y_1), embedded into $\mathbb{P}^3$ by $(x_1, y_1) \mapsto (x_1 : y_1 : 1 : x_1y_1)$; two extra points at infinity if d is a square, namely $(0 : \pm\sqrt{d} : 0 : 1)$; and two extra points at infinity if a/d is a square, namely $(1 : 0 : 0 : \pm\sqrt{a/d})$.

Bernstein and Lange [BL10] solved the problems posed by points at infinity for the Edwards addition law by introducing *completed points*.

Definition 1.27. The *completed* twisted Edwards curve is given by

$$\overline{E}_{\mathrm{E},a,d} = \big\{((X : Z), (Y : T)) \in \mathbb{P}^1 \times \mathbb{P}^1 : aX^2T^2 + Y^2Z^2 = Z^2T^2 + dX^2Y^2\big\}.$$

The completed points are the affine points (x_1, y_1), embedded as usual into $\mathbb{P}^1 \times \mathbb{P}^1$ by $(x_1, y_1) \mapsto ((x_1 : 1), (y_1 : 1))$; two extra points at infinity if d is a square, namely $((1 : \pm\sqrt{d}), (1 : 0))$; and two extra points at infinity if d/a is a square, namely $((1 : 0), (\pm\sqrt{a/d} : 1))$.

Remark 1.28. The completed twisted Edwards curve does not have singularities at infinity. The points at infinity are $((1 : 0), (\pm\sqrt{a/d} : 1))$ and $((1 : \pm\sqrt{d}), (1 : 0))$. They are defined over $k(\sqrt{ad})$.

In order to map points on the completed curve to twisted Edwards curves one needs a detour via $\mathbb{P}^3$. We recall for this purpose that the *Segre embedding* is the embedding of the Cartesian product $\mathbb{P}^r \times \mathbb{P}^s$ into a subvariety of $\mathbb{P}^{rs+r+s}$ respecting the lexicographical order as $((a_0, \ldots, a_r), (b_0, \ldots, b_s)) \mapsto (\ldots, a_i b_j, \ldots)$ $(0 \le i \le r, 0 \le j \le s)$; see also [Har77, Exercise 2.14].
The completed curve maps isomorphically to the extended curve via the Segre embedding $((X : Z), (Y : T)) \mapsto (XT : YZ : ZT : XY)$ of $\mathbb{P}^1 \times \mathbb{P}^1$ into $\mathbb{P}^3$. It maps onto the projective curve $E^{\text{hom}}_{\text{E},a,d}$ via $((X : Z), (Y : T)) \mapsto (XT : YZ : ZT)$, but this map is not an isomorphism: it sends the two points $((1 : \pm\sqrt{d}), (1 : 0))$ to $(0 : 1 : 0)$ and sends the two points $((1 : 0), (\pm\sqrt{a/d} : 1))$ to $(1 : 0 : 0)$. The completed curve also maps onto the inverted curve via $((X : Z), (Y : T)) \mapsto (YZ : XT : XY)$, but this map sends the two points $((0 : 1), (\pm 1 : 1))$ to $(1 : 0 : 0)$, and sends the two points $((\pm 1 : 1), (0 : 1))$ to $(0 : 1 : 0)$.
Bernstein and Lange [BL10] developed a group law on the completed curve $\overline{E}_{\text{E},a,d}$ which is stated in the following section.

1.2.4 The Edwards group

As discussed in Remark 1.9, if $a = 1$ and d is not a square then the affine Edwards addition law is complete: the affine points (x_1, y_1) on the curve form a group.
However, if d is a square then the addition law is not necessarily a group law: there can be pairs (x_1, y_1) and (x_2, y_2) where $1 + dx_1x_2y_1y_2 = 0$ or $1 - dx_1x_2y_1y_2 = 0$. Even worse, there can be pairs (x_1, y_1) and (x_2, y_2) for which $1 + dx_1x_2y_1y_2 = 0 = x_1y_2 + y_1x_2$ or $1 - dx_1x_2y_1y_2 = 0 = y_1y_2 - ax_1x_2$. Switching from affine coordinates to projective or inverted or extended or completed coordinates does not allow the Edwards addition law to add such points.
There is nevertheless a standard group law for the completed curve $\overline{E}_{\text{E},a,d}$ in $\mathbb{P}^1 \times \mathbb{P}^1$. One way to define the group law is through a correspondence to the traditional chord-and-tangent group on an equivalent Weierstrass curve where one has to distinguish several cases; but it is simpler to directly define a group law $+ : \overline{E}_{\text{E},a,d} \times \overline{E}_{\text{E},a,d} \to \overline{E}_{\text{E},a,d}$. Bernstein and Lange showed in [BL10] that the Edwards addition law and the dual addition law form a complete system of addition laws for $\overline{E}_{\text{E},a,d}$: any pair of input points that cannot be added by the Edwards addition law can be added by the dual addition law.
The following theorem summarizes the results from [BL10]. The next section uses this group law to characterize points of small order in $\overline{E}_{\text{E},a,d}$.

Theorem 1.29. *Fix a field k with $\operatorname{char}(k) \neq 2$. Fix distinct nonzero elements $a, d \in k$. Fix $P_1, P_2 \in \overline{E}_{\text{E},a,d}(k)$. Write P_1 as $((X_1 : Z_1), (Y_1 : T_1))$ and write P_2 as $((X_2 : Z_2), (Y_2 : T_2))$. Define*

$$\begin{aligned} X_3 &= X_1Y_2Z_2T_1 + X_2Y_1Z_1T_2, \\ Z_3 &= Z_1Z_2T_1T_2 + dX_1X_2Y_1Y_2, \\ Y_3 &= Y_1Y_2Z_1Z_2 - aX_1X_2T_1T_2, \\ T_3 &= Z_1Z_2T_1T_2 - dX_1X_2Y_1Y_2; \end{aligned}$$

and

$$\begin{aligned} X_3' &= X_1Y_1Z_2T_2 + X_2Y_2Z_1T_1, \\ Z_3' &= aX_1X_2T_1T_2 + Y_1Y_2Z_1Z_2, \\ Y_3' &= X_1Y_1Z_2T_2 - X_2Y_2Z_1T_1, \\ T_3' &= X_1Y_2Z_2T_1 - X_2Y_1Z_1T_2. \end{aligned}$$

Then $X_3Z_3' = X_3'Z_3$ *and* $Y_3T_3' = Y_3'T_3$. *Furthermore, at least one of the following cases occurs:*

- $(X_3, Z_3) \neq (0,0)$ *and* $(Y_3, T_3) \neq (0,0)$. *Then* $P_1 + P_2 = ((X_3 : Z_3), (Y_3 : T_3))$.
- $(X_3', Z_3') \neq (0,0)$ *and* $(Y_3', T_3') \neq (0,0)$. *Then* $P_1 + P_2 = ((X_3' : Z_3'), (Y_3' : T_3'))$.

If $P_1 = P_2$ *then the first case occurs.*

1.2.5 Points of small order on $\overline{E}_{\mathrm{E},a,d}$

The complete set of addition laws from [BL10] (presented in the previous section) enables us to investigate the order of any point.
This section characterizes all points of order 2, 3, and 4, and states conditions on the parameters of the twisted Edwards curve for such points to exist. This section also characterizes points of order 8 relevant to later sections.
The following theorem gives a complete study of points of order 2 and 4 in $\overline{E}_{\mathrm{E},a,d}$.

Theorem 1.30. *Fix a field* k *with* $\mathrm{char}(k) \neq 2$. *Fix distinct nonzero elements* $a, d \in k$. *The following points are in* $\overline{E}_{\mathrm{E},a,d}(k)$ *and have the stated orders.*
Points of order 2**:**
The point $((0:1),(-1:1))$ *has order* 2.
If a/d *is a square in* k *then the points* $((1:0),(\pm\sqrt{a/d}:1))$ *have order* 2.
There are no other points of order 2.
Points of order 4 **doubling to** $((0:1),(-1:1))$**:**
If a *is a square in* k *then the points* $((1:\pm\sqrt{a}),(0:1))$ *have order* 4 *and double to* $((0:1),(-1:1))$.
If d *is a square in* k *then the points* $((1:\pm\sqrt{d}),(1:0))$ *have order* 4 *and double to* $((0:1),(-1:1))$.
There are no other points doubling to $((0:1),(-1:1))$.
Points of order 4 **doubling to** $((1:0),(\pm\sqrt{a/d}:1))$**:** *Assume that* $s \in k$ *satisfies* $s^2 = a/d$.
If s *and* $-s/a$ *are squares in* k *then the points* $((\pm\sqrt{-s/a}:1),(\pm\sqrt{s}:1))$, *where the signs may be chosen independently, have order* 4 *and double to* $((1:0),(s:1))$.
There are no other points doubling to $((1:0),(s:1))$.

Proof. Doublings can always be computed by X_3, Z_3, Y_3, T_3 from Theorem 1.29: in other words, all curve points $((X:Z),(Y:T))$ have $(2XYZT, Z^2T^2 + dX^2Y^2) \neq (0,0)$ and $(Y^2Z^2 - aX^2T^2, Z^2T^2 - dX^2Y^2) \neq (0,0)$, so

$$\begin{aligned} &[2]((X:Z),(Y:T)) \\ &\quad = ((2XYZT : Z^2T^2 + dX^2Y^2), (Y^2Z^2 - aX^2T^2 : Z^2T^2 - dX^2Y^2)). \end{aligned}$$

In particular:

- $[2]((0:1),(-1:1)) = ((0:1),(1:1))$.
- $[2]((1:0),(\pm\sqrt{a/d}:1)) = ((0:d(a/d)),(-a:-d(a/d))) = ((0:1),(1:1))$.
- $[2]((1:\pm\sqrt{a}),(0:1)) = ((0:a),(-a:a)) = ((0:1),(-1:1))$.
- $[2]((1:\pm\sqrt{d}),(1:0)) = ((0:d),(d:-d)) = ((0:1),(-1:1))$.
- $[2]((\pm\sqrt{-s/a}:1),(\pm\sqrt{s}:1)) = ((\ldots:1+d(-s/a)s),(s-a(-s/a):1-d(-s/a)s)) = ((1:0),(s:1))$ since $d(s/a)s = s^2d/a = 1$.

To see that there is no other point of order 2 or 4, observe first that every point $((X:Z),(Y:T))$ on $\overline{E}_{\mathrm{E},a,d}$ with $X=0$ or $Y=0$ or $Z=0$ or $T=0$ is either $((0:1),(1:1))$ or one of the points doubled above. The only remaining points are affine points $((x:1),(y:1))$ with $x \neq 0$ and $y \neq 0$. The double of $((x:1),(y:1))$ is $((2xy:1+dx^2y^2),(y^2-ax^2:1-dx^2y^2))$; but $2xy \neq 0$, so this double cannot be $((0:1),(1:1))$, so $((x:1),(y:1))$ cannot have order 2. For the same reason, the double cannot be $((0:1),(-1:1))$. The only remaining case is that the double is $((1:0),(s:1))$ where $s^2 = a/d$. Then $ax^2+y^2 = 1+dx^2y^2 = 0$ so $ax^2 = -y^2$; and $y^2-ax^2 = s(1-dx^2y^2)$, so $2y^2 = y^2-ax^2 = s(1-dx^2y^2) = 2s$, so $y = \pm\sqrt{s}$, and finally $ax^2 = -s$ so $x = \pm\sqrt{-s/a}$. □

Chapter 3 studies Edwards curves over the rationals $\mathbb{Q}$ for which $((1:\pm\sqrt{a}),(0:1))$ is on the curve. In this case the points of order 8 double to either these points or to $((1:\pm\sqrt{d}),(1:0))$.

Theorem 1.31. *Fix a field k with $\mathrm{char}(k) \neq 2$. Fix distinct nonzero elements $a, d \in k$. The following points are in $\overline{E}_{\mathrm{E},a,d}(k)$ and have the stated orders.*
Points of order 8 doubling to $((1:\pm\sqrt{a}),(0:1))$: *If $r \in k$ satisfies $r^2 = a$ then any element of $\overline{E}_{\mathrm{E},a,d}(k)$ doubling to $((1:r),(0:1))$ can be written as $((x_8:1),(rx_8:1))$ for some $x_8 \in k$ satisfying $adx_8^4 - 2ax_8^2 + 1 = 0$.*
Conversely, if $r, x_8 \in k$ satisfy $r^2 = a$ and $adx_8^4 - 2ax_8^2 + 1 = 0$ then the two points $((\pm x_8:1),(\pm rx_8:1))$, with matching signs, have order 8 and double to $((1:r),(0:1))$. If also d is a square in k then the two points $((1:\pm rx_8\sqrt{d}),(1:\pm x_8\sqrt{d}))$, with matching signs, have order 8, double to $((1:r),(0:1))$, and are different from $((\pm x_8:1),(\pm rx_8:1))$. There are no other points doubling to $((1:r),(0:1))$.
Points of order 8 doubling to $((1:\pm\sqrt{d}),(1:0))$: *If $s \in k$ satisfies $s^2 = d$ then any element of $\overline{E}_{\mathrm{E},a,d}(k)$ doubling to $((1:s),(1:0))$ can be written as $((\bar{x}_8:1),(1:s\bar{x}_8))$ for some $\bar{x}_8 \in k$ satisfying $ad\bar{x}_8^4 - 2d\bar{x}_8^2 + 1 = 0$.*
Conversely, if $s, \bar{x}_8 \in k$ satisfy $s^2 = d$ and $ad\bar{x}_8^4 - 2d\bar{x}_8^2 + 1 = 0$, then the two points $((\pm\bar{x}_8:1),(1:\pm s\bar{x}_8))$, with matching signs, have order 8 and double to $((1:s),(1:0))$. If also a is a square in k then the two points $((1:\pm sx_8\sqrt{a}),(\pm x_8\sqrt{a}:1))$, with matching signs, have order 8, double to $((1:s),(1:0))$, and are different from $((\pm\bar{x}_8:1),(1:\pm s\bar{x}_8))$. There are no other points doubling to $((1:s),(1:0))$.

Proof. Every point with a zero coordinate has order at most 4 by Theorem 1.30, so any point of order 8 has the form $((x_8 : 1), (y_8 : 1))$, with $x_8 \neq 0$ and $y_8 \neq 0$, and with double $((2x_8y_8 : 1 + dx_8^2y_8^2), (y_8^2 - ax_8^2 : 1 - dx_8^2y_8^2))$.

Part 1: If the double is $((1 : r), (0 : 1))$ then $y_8^2 - ax_8^2 = 0$ and $2x_8y_8r = 1 + dx_8^2y_8^2 = ax_8^2 + y_8^2 = 2ax_8^2 = 2r^2x_8^2$. Cancel $2x_8r$ to see that $y_8 = rx_8$. Hence $adx_8^4 - 2ax_8^2 + 1 = dx_8^2y_8^2 - (1 + dx_8^2y_8^2) + 1 = 0$ and the original point is $((x_8 : 1), (rx_8 : 1))$.

Conversely, if $r, x_8 \in k$ satisfy $r^2 = a$ and $adx_8^4 - 2ax_8^2 + 1 = 0$, then the point $((x_8 : 1), (rx_8 : 1))$ is on the curve since $ax_8^2 + (rx_8)^2 = 2ax_8^2 = adx_8^4 + 1 = 1 + dx_8^2(rx_8)^2$, and it doubles to $((2x_8rx_8 : 1 + dx_8^2r^2x_8^2), (r^2x_8^2 - ax_8^2 : \ldots)) = ((2x_8rx_8 : 2ax_8^2), (0 : \ldots)) = ((1 : r), (0 : 1))$.

The other points doubling to $((1 : r), (0 : 1))$ are $((x : 1), (rx : 1))$ for other $x \in k$ satisfying $adx^4 - 2ax^2 + 1 = 0$. If d is not a square in k then $adx^4 - 2ax^2 + 1 = adx^4 - (adx_8^2 + 1/x_8^2)x^2 + 1 = (x - x_8)(x + x_8)(adx^2 - 1/x_8^2)$, with $adx^2 - 1/x_8^2$ irreducible, so the only points doubling to $((1 : r), (0 : 1))$ are $((\pm x_8 : 1), (\pm rx_8 : 1))$. If d is a square in k then $adx^4 - 2ax^2 + 1 = (x - x_8)(x + x_8)(rx\sqrt{d} - 1/x_8)(rx\sqrt{d} + 1/x_8)$ so the only points doubling to $((1 : r), (0 : 1))$ are $((\pm x_8 : 1), (\pm rx_8 : 1))$ and $((1 : \pm rx_8\sqrt{d}), (1 : \pm x_8\sqrt{d}))$. These points are distinct: otherwise $\pm rx_8^2\sqrt{d} = 1$ so $adx_8^4 = 1$ so $2ax_8^2 = 2$ so $ax_8^2 = 1$ so $y_8 = 0$ from the curve equation; contradiction.

Part 2: If the double of $((\bar{x}_8 : 1), (\bar{y}_8 : 1))$ is $((1 : s), (1 : 0))$ then $1 - d\bar{x}_8^2\bar{y}_8^2 = 0$ and $2\bar{x}_8\bar{y}_8s = 1 + d\bar{x}_8^2\bar{y}_8^2 = 2$ so $\bar{y}_8 = 1/(s\bar{x}_8)$. Hence $ad\bar{x}_8^4 - 2d\bar{x}_8^2 + 1 = (a\bar{x}_8^2 - 2 + \bar{y}_8^2)d\bar{x}_8^2 = 0$ and the original point is $((\bar{x}_8 : 1), (1 : s\bar{x}_8))$.

Conversely, if $s, \bar{x}_8 \in k$ satisfy $s^2 = d$ and $ad\bar{x}_8^4 - 2d\bar{x}_8^2 + 1 = 0$, then the point $((\bar{x}_8 : 1), (1 : s\bar{x}_8))$ is on the curve since $d\bar{x}_8^2(a\bar{x}_8^2 + \bar{y}_8^2) = d\bar{x}_8^2(a\bar{x}_8^2 + 1/(s^2\bar{x}_8^2)) = ad\bar{x}_8^4 + 1 = 2d\bar{x}_8^2 = d\bar{x}_8^2 + d\bar{x}_8^4/\bar{x}_8^2 = d\bar{x}_8^2(1 + d\bar{x}_8^2/(s^2\bar{x}_8^2)) = d\bar{x}_8^2(1 + d\bar{x}_8^2\bar{y}_8^2))$. The point doubles to $((2s\bar{x}_8^2 : s^2\bar{x}_8^2 + d\bar{x}_8^2), (1 - as^2\bar{x}_8^4 : s^2\bar{x}_8^2 - d\bar{x}_8^2)) = ((1 : s), (1 - ad\bar{x}_8^4 : s^2\bar{x}_8^2 - s^2\bar{x}_8^2)) = ((1 : s), (1 : 0))$.

The other points doubling to $((1 : s), (1 : 0))$ are $((x : 1), (1 : sx))$ for other $x \in k$ satisfying $adx^4 - 2dx^2 + 1 = 0$. If a is not a square in k then $adx^4 - 2dx^2 + 1 = adx^4 - (ad\bar{x}_8^2 + 1/\bar{x}_8^2)x^2 + 1 = (x - \bar{x}_8)(x + \bar{x}_8)(adx^2 - 1/\bar{x}_8^2)$, with $adx^2 - 1/\bar{x}_8^2$ irreducible, so the only points doubling to $((1 : s), (1 : 0))$ are $((\pm\bar{x}_8 : 1), (1 : \pm s\bar{x}_8))$. If a is a square in k then $adx^4 - 2dx^2 + 1 = (x - \bar{x}_8)(x + \bar{x}_8)(sx\sqrt{a} - 1/\bar{x}_8)(sx\sqrt{a} + 1/\bar{x}_8)$ so the only points doubling to $((1 : s), (1 : 0))$ are $((\pm\bar{x}_8 : 1), (1 : \pm s\bar{x}_8))$ and $((1 : \pm s\bar{x}_8\sqrt{a})), (\pm\bar{x}_8\sqrt{a} : 1))$. These points are distinct: otherwise $\pm s\bar{x}_8^2\sqrt{a} = 1$ so $ad\bar{x}_8^4 = 1$ so $2d\bar{x}_8^2 = 2$ so $d\bar{x}_8^2 = 1$ so $a\bar{x}_8^2 = 1$ from the curve equation and in particular $a\bar{x}_8^2 = d\bar{x}_8^2$. So either $a = d$ or $\bar{x}_8 = 0$; contradiction. □

Theorem 1.32. *Fix a field k with $\mathrm{char}(k) \neq 2$. Fix distinct nonzero elements $a, d \in k$. If $x_3, y_3 \in k$ satisfy $ax_3^2 + y_3^2 = 1 + dx_3^2y_3^2 = -2y_3$ then $((x_3 : 1), (y_3 : 1))$ is a point of order 3 on $\overline{E}_{\mathrm{E},a,d}(k)$. Conversely, all points of order 3 on $\overline{E}_{\mathrm{E},a,d}(k)$ arise in this way.*

Proof. Doublings can always be computed by X_3, Z_3, Y_3, T_3 from Theorem 1.29, as in the proof of Theorem 1.30.

Observe that $((x_3 : 1), (y_3 : 1)) \in \overline{E}_{\mathrm{E},a,d}(k)$ since $ax_3^2 + y_3^2 = 1 + dx_3^2y_3^2$. Now

$$\begin{aligned}[2]((x_3 : 1), (y_3 : 1)) &= ((2x_3y_3 : 1 + dx_3^2y_3^2), (y_3^2 - ax_3^2 : 1 - dx_3^2y_3^2)) \\ &= ((2x_3y_3 : -2y_3), (2y_3^2 + 2y_3 : 2y_3 + 2)) \\ &= ((-x_3 : 1), (y_3 : 1))\end{aligned}$$

so $((x_3 : 1), (y_3 : 1))$ has order dividing 3. It cannot have order 1 (since otherwise $x_3 = 0$ so $y_3^2 = 1 = -2y_3$), so it has order 3.
Conversely, consider any point $P = ((X_1 : Z_1), (Y_1 : T_1))$ of order 3 in $\overline{E}_{\mathrm{E},a,d}(k)$. The equation $[2]P = -P$ then implies $(2X_1Y_1Z_1T_1 : Z_1^2T_1^2 + dX_1^2Y_1^2) = (-X_1 : Z_1)$. Every point in $\overline{E}_{\mathrm{E},a,d}$ with a zero coordinate has order 1, 2, or 4 by Theorem 1.30, so $X_1, Z_1, Y_1, T_1 \neq 0$. Define $x_3 = X_1/Z_1$ and $y_3 = Y_1/T_1$. Then $P = ((x_3 : 1), (y_3 : 1))$; furthermore $(2x_3y_3 : 1 + dx_3^2y_3^2) = (-x_3 : 1)$ and $x_3 \neq 0$ so $-2y_3 = 1 + dx_3^2y_3^2 = ax_3^2 + y_3^2$. □

1.2.6 Montgomery curves and twisted Edwards curves

Montgomery in [Mon87, Section 10.3.1] introduced what are now called "Montgomery curves" and "Montgomery coordinates" in order to gain more speed for the Elliptic-Curve Method which will be discussed in Chapter 3.
This section shows that the set of Montgomery curves over k is equivalent to the set of twisted Edwards curves over k. We also analyze the extent to which this is true without twists.

Definition 1.33 (Montgomery curve)**.** Fix a field k with $\mathrm{char}(k) \neq 2$. Fix $A \in k \setminus \{-2, 2\}$ and $B \in k \setminus \{0\}$. The *Montgomery curve with coefficients A and B* is the curve

$$E_{\mathrm{M},A,B} : Bv^2 = u^3 + Au^2 + u.$$

Standard algorithms for transforming a Weierstrass curve into a Montgomery curve if possible (see, e.g., [DL05, Section 13.2.3.c]) can be combined with the following explicit transformation from a Montgomery curve to a twisted Edwards curve.

Theorem 1.34. *Fix a field k with* $\mathrm{char}(k) \neq 2$.
(i) Every twisted Edwards curve over k is birationally equivalent over k to a Montgomery curve.
Specifically, fix distinct nonzero elements $a, d \in k$. The twisted Edwards curve $E_{\mathrm{E},a,d}$ is birationally equivalent to the Montgomery curve $E_{\mathrm{M},A,B}$, where $A = 2(a+d)/(a-d)$ and $B = 4/(a-d)$. The map $(x, y) \mapsto (u, v) = ((1+y)/(1-y), (1+y)/((1-y)x))$ is a birational equivalence from $E_{\mathrm{E},a,d}$ to $E_{\mathrm{M},A,B}$, with inverse $(u, v) \mapsto (x, y) = (u/v, (u-1)/(u+1))$.
(ii) Conversely, every Montgomery curve over k is birationally equivalent over k to a twisted Edwards curve.
Specifically, fix $A \in k \setminus \{-2, 2\}$ and $B \in k \setminus \{0\}$. The Montgomery curve $E_{\mathrm{M},A,B}$ is birationally equivalent to the twisted Edwards curve $E_{\mathrm{E},a,d}$, where $a = (A+2)/B$ and $d = (A-2)/B$.

Proof. (i) Note that A and B are defined, since $a \neq d$. Note further that $A \in k \setminus \{-2, 2\}$ and $B \in k \setminus \{0\}$: if $A = 2$ then $a + d = a - d$ so $d = 0$; contradiction; if $A = -2$ then $a + d = d - a$ so $a = 0$; contradiction. Thus $E_{\mathrm{M},A,B}$ is a Montgomery curve.
The following script for the Sage computer-algebra system [S+10] checks that the quantities $u = (1+y)/(1-y)$ and $v = (1+y)/((1-y)x)$ satisfy $Bv^2 = u^3 + Au^2 + u$ in the function field of the curve $E_{\mathrm{E},a,d} : ax^2 + y^2 = 1 + dx^2y^2$:

```
R.<a,d,x,y>=QQ[]
A=2*(a+d)/(a-d)
B=4/(a-d)
S=R.quotient(a*x^2+y^2-(1+d*x^2*y^2))
u=(1+y)/(1-y)
v=(1+y)/((1-y)*x)
0==S((B*v^2-u^3-A*u^2-u).numerator())
```

The exceptional cases $y = 1$ and $x = 0$ occur for only finitely many points (x, y) on $E_{\mathrm{E},a,d}$. Conversely, let $x = u/v$ and $y = (u-1)/(u+1)$; the exceptional cases $v = 0$ and $u = -1$ occur for only finitely many points (u, v) on $E_{\mathrm{M},A,B}$.
(ii) Note that a and d are defined, since $B \neq 0$. Note further that $a \neq 0$ since $A \neq -2$; $d \neq 0$ since $A \neq 2$; and $a \neq d$. Thus $E_{\mathrm{E},a,d}$ is a twisted Edwards curve. Furthermore

$$2\frac{a+d}{a-d} = 2\frac{\frac{A+2}{B} + \frac{A-2}{B}}{\frac{A+2}{B} - \frac{A-2}{B}} = A \quad \text{and} \quad \frac{4}{(a-d)} = \frac{4}{\frac{A+2}{B} - \frac{A-2}{B}} = B.$$

Hence $E_{\mathrm{E},a,d}$ is birationally equivalent to $E_{\mathrm{M},A,B}$ by (i). □

Remark 1.35 (Exceptional points for the birational equivalence)**.** The map $(u, v) \mapsto (u/v, (u-1)/(u+1))$ from $E_{\mathrm{M},A,B}$ to $E_{\mathrm{E},a,d}$ in Theorem 1.34 is undefined at the points of $E_{\mathrm{M},A,B} : Bv^2 = u^3 + Au^2 + u$ with $v = 0$ or $u + 1 = 0$. We investigate these points in more detail:

- The neutral element $(0, 1)$ on $E_{\mathrm{E},a,d}$ is mapped to the neutral element on $E_{\mathrm{M},A,B}$, which is the point at infinity. The point $(0, 0)$ on $E_{\mathrm{M},A,B}$ corresponds to the affine point of order 2 on $E_{\mathrm{E},a,d}$, namely $(0, -1)$. This point and $(0, 1)$ are the only exceptional points of the inverse map $(x, y) \mapsto ((1+y)/(1-y), (1+y)/((1-y)x))$.

- If $(A + 2)(A - 2)$ is a square (i.e., if ad is a square) then there are two more points with $v = 0$, namely $((-A \pm \sqrt{(A+2)(A-2)})/2, 0)$. These points have order 2. These points correspond to two points of order 2 at infinity on the desingularization of $E_{\mathrm{E},a,d}$.

- If $(A - 2)/B$ is a square (i.e., if d is a square) then there are two points with $u = -1$, namely $(-1, \pm\sqrt{(A-2)/B})$. These points have order 4. These points correspond to two points of order 4 at infinity on the desingularization of $E_{\mathrm{E},a,d}$.

Every Montgomery curve $E_{\mathrm{M},A,B}$ is birationally equivalent to a twisted Edwards curve by Theorem 1.34, and therefore to a quadratic twist of an Edwards curve. In other words, there is a quadratic twist of $E_{\mathrm{M},A,B}$ that is birationally equivalent to an Edwards curve.

There are two situations in which twisting is not necessary. These are summarized in two theorems: Theorem 1.36 states that every elliptic curve having a point of order 4 is birationally equivalent to an Edwards curve. Theorem 1.37 states that, over a finite field k with $\#k \equiv 3 \pmod 4$, every Montgomery curve is birationally equivalent to an Edwards curve.

Some special cases of these theorems were already known. Bernstein and Lange proved in [BL07b, Theorem 2.1(1)] that every elliptic curve having a point of order 4 is birationally equivalent to a twist of an Edwards curve, and in [BL07b, Theorem 2.1(3)] that, over a finite field, every elliptic curve having a point of order 4 and a unique point of order 2 is birationally equivalent to an Edwards curve. The following theorem proves that the twist in [BL07b, Theorem 2.1(1)] is unnecessary, and that the unique point of order 2 in [BL07b, Theorem 2.1(3)] is unnecessary.

Theorem 1.36. *Fix a field k with $\mathrm{char}(k) \neq 2$. Let E be an elliptic curve over k. The group $E(k)$ has an element of order 4 if and only if E is birationally equivalent over k to an Edwards curve.*

Proof. Assume that E is birationally equivalent over k to an Edwards curve $E_{\mathrm{E},1,d}$. The elliptic-curve addition law corresponds to the Edwards addition law; see [BL07b, Theorem 3.2]. The point $(1,0)$ on $E_{\mathrm{E},1,d}$ has order 4, so E must have a point of order 4.

Conversely, assume that E has a point (u_4, v_4) of order 4. As in [BL07b, Theorem 2.1, proof], observe that $u_4 \neq 0$ and $v_4 \neq 0$; assume without loss of generality that E has the form $v^2 = u^3 + (v_4^2/u_4^2 - 2u_4)u^2 + u_4^2 u$; define $d = 1 - 4u_4^3/v_4^2$; and observe that $d \notin \{0, 1\}$.

The following script for the Sage computer-algebra system checks that the quantities $x = v_4 u/(u_4 v)$ and $y = (u - u_4)/(u + u_4)$ satisfy $x^2 + y^2 = 1 + dx^2y^2$ in the function field of E:

```
R.<u,v,u4,v4>=QQ[]
d=1-4*u4^3/v4^2
S=R.quotient((v^2-u^3-(v4^2/u4^2-2*u4)*u^2-u4^2*u).numerator())
x=v4*u/(u4*v)
y=(u-u4)/(u+u4)
0==S((x^2+y^2-1-d*x^2*y^2).numerator())
```

The exceptional cases $u_4 v = 0$ and $u = -u_4$ occur for only finitely many points (u, v) on E. Conversely, let $u = u_4(1 + y)/(1 - y)$ and $v = v_4(1 + y)/((1 - y)x)$; the exceptional cases $y = 1$ and $x = 0$ occur for only finitely many points (x, y) on $E_{\mathrm{E},1,d}$.

Therefore the rational map $(u, v) \mapsto (x, y) = (v_4 u/(u_4 v), (u - u_4)/(u + u_4))$, with inverse $(x, y) \mapsto (u, v) = (u_4(1 + y)/(1 - y), v_4(1 + y)/((1 - y)x))$, is a birational equivalence from E to the Edwards curve $E_{\mathrm{E},1,d}$. □

Theorem 1.37. *If k is a finite field with $\#k \equiv 3 \pmod 4$ then every Montgomery curve over k is birationally equivalent over k to an Edwards curve.*

Proof. Fix $A \in k \setminus \{-2, 2\}$ and $B \in k \setminus \{0\}$. We will use an idea of Okeya, Kurumatani, and Sakurai [OKS00], building upon the observations credited to Suyama in [Mon87, page 262], to prove that the Montgomery curve $E_{\mathrm{M},A,B}$ has a point of order 4. This fact can be extracted from [OKS00, Theorem 1] when $\#k$ is prime, but to keep this thesis self-contained we include a direct proof.
Case 1: $(A + 2)/B$ is a square. Then (as mentioned before) $E_{\mathrm{M},A,B}$ has a point $(1, \sqrt{(A + 2)/B})$ of order 4.
Case 2: $(A + 2)/B$ is a nonsquare but $(A - 2)/B$ is a square. Then $E_{\mathrm{M},A,B}$ has a point $(-1, \sqrt{(A - 2)/B})$ of order 4.
Case 3: $(A + 2)/B$ and $(A - 2)/B$ are nonsquares. Then $(A + 2)(A - 2)$ must be square, since k is finite. The Montgomery curve $E_{\mathrm{M},A,A+2}$ has three points $(0, 0)$, $((-A \pm \sqrt{(A + 2)(A - 2)})/2, 0)$ of order 2, and a point $(1, 1)$ of order 4, and two points $(1, \pm 1)$ of order 4, so $\#E_{\mathrm{M},A,A+2}(k) \equiv 0 \pmod 8$. Furthermore, $E_{\mathrm{M},A,B}$ is a nontrivial quadratic twist of $E_{\mathrm{M},A,A+2}$, so $\#E_{\mathrm{M},A,B}(k) + \#E_{\mathrm{M},A,A+2}(k) = 2\#k + 2 \equiv 0 \pmod 8$. Therefore $\#E_{\mathrm{M},A,B}(k) \equiv 0 \pmod 8$. The curve $E_{\mathrm{M},A,B}$ cannot have more than three points of order 2, so it must have a point of order 4.
In every case $E_{\mathrm{M},A,B}$ has a point of order 4. By Theorem 1.36, $E_{\mathrm{M},A,B}$ is birationally equivalent to an Edwards curve. □

This theorem does not generalize to $\#k \equiv 1 \pmod 4$. For example, the Montgomery curve $\mathrm{E}_{\mathrm{M},9,1}$ over $\mathbb{F}_{17}$ has order 20 and group structure isomorphic to $\mathbb{Z}/2 \times \mathbb{Z}/10$. This curve is birationally equivalent to the twisted Edwards curve $\mathrm{E}_{\mathrm{E},11,7}$, but it does not have a point of order 4, so it is not birationally equivalent to an Edwards curve.

Theorem 1.38. *Let k be a finite field with $\#k \equiv 1 \pmod 4$. Let $E_{\mathrm{M},A,B}$ be a Montgomery curve so that $(A + 2)(A - 2)$ is a square and let δ be a nonsquare.*
Exactly one of $E_{\mathrm{M},A,B}$ and its nontrivial quadratic twist $E_{\mathrm{M},A,\delta B}$ is birationally equivalent to an Edwards curve.
In particular, $E_{\mathrm{M},A,A+2}$ is birationally equivalent to an Edwards curve.

Proof. Since $(A+2)(A-2)$ is a square both $E_{\mathrm{M},A,B}$ and $E_{\mathrm{M},A,\delta B}$ contain a subgroup isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. This subgroup accounts for a factor of 4 in the group order. Since $\#E_{\mathrm{M},A,B}(k) + \#E_{\mathrm{M},A,\delta B}(k) = 2\#k + 2 \equiv 4 \pmod 8$ exactly one of $\#E_{\mathrm{M},A,B}(k)$ and $\#E_{\mathrm{M},A,\delta B}(k)$ is divisible by 4 but not by 8. That curve cannot have a point of order 4 while the other one has a point of order 4. The first statement follows from Theorem 1.36.
The second statement also follows from Theorem 1.36, since the point $(1, 1)$ on $E_{\mathrm{M},A,A+2}$ has order 4. □

1.3 Arithmetic on (twisted) Edwards curves

This section shows how elliptic curves in Edwards form and twisted Edwards form speed up elliptic-curve cryptography. In order to measure the cost of elliptic-curve arithmetic we count how many field multiplications **M**, squarings **S**, additions **a**, multiplications by a small constant factor **D**, and how many inversions **I** an arithmetic operation on a given curve shape takes.[1]
The addition formulas for Edwards curves and twisted Edwards curves both involve divisions, i.e., inversions in the field which are much more costly than additions, multiplications or squarings. E.g., the "Explicit-Formulas Database" [BL07a] includes a cost table that assumes that one field inversion **I** costs as much as 100 field multiplications **M**. In general, the data base counts inversions and multiplications separately. In the following we discuss formulas which avoid inversions.
Second, this section deals with the problem of computing the m-fold of a point on an elliptic curve in Edwards form. The naive way of computing $[m]P$ is to repeatedly add the point to itself. This section considers fast formulas which given a point P on an Edwards curve yield the result of *doubling*, i.e., computing the 2-fold $[2]P$; *tripling*, i.e., computing the 3-fold $[3]P$; and *quintupling*, i.e., computing the 5-fold $[5]P$, using fewer operations.

1.3.1 Inversion-free formulas

A common way to speed up elliptic-curve arithmetic is to switch to projective coordinates in order to get rid of denominators. The formulas presented in [BBJ+08] are for Edwards curves and twisted Edwards curves. For Edwards curves multiplications by the twisted-Edwards-curve coefficient a need not to be considered.
The sum $(X_3 : Y_3 : Z_3)$ of two points $(X_1 : Y_1 : Z_1)$, $(X_2 : Y_2 : Z_2)$ on $E^{\text{hom}}_{\text{E},a,d}$ equals

$$A = Z_1 \cdot Z_2;\ B = A^2;\ C = X_1 \cdot X_2;\ D = Y_1 \cdot Y_2;\ E = dC \cdot D;$$
$$F = B - E;\ G = B + E;\ X_3 = A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D);$$
$$Y_3 = A \cdot G \cdot (D - aC);\ Z_3 = F \cdot G.$$

These formulas compute the sum in $10\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$, where the $2\mathbf{D}$ are one multiplication by a and one by d. If Z_2 is known to be 1 then the multiplication $A = Z_1 \cdot Z_2$ can be omitted. This is called *mixed addition* and takes $1\mathbf{M}$ less, namely $9\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$.

1.3.2 Doubling on twisted Edwards curves

Bernstein and Lange [BL07b] derived special formulas for doubling from the general Edwards addition law (1.3). The doubling formulas for twisted Edwards curves

[1] If the curve coefficient is small (e.g., d for Edwards curves) those multiplications can be implemented as a few additions and thus we count them separately using **D**.

in [BBJ+08] are obtained in the same way. Given a point (x_1, y_1) on $E_{\mathrm{E},a,d} : ax^2 + y^2 = 1 + dx^2y^2$ substitute $dx_1^2y_1^2$ by $ax_1^2 + y_1^2 - 1$ in the following formula:

$$(x_1, y_1), (x_1, y_1) \mapsto \left(\frac{2x_1y_1}{1 + dx_1^2y_1^2}, \frac{y_1^2 - ax_1^2}{1 - dx_1^2y_1^2}\right) = \left(\frac{2x_1y_1}{ax_1^2 + y_1^2}, \frac{y_1^2 - x_1^2}{2 - (ax_1^2 + y_1^2)}\right).$$

This substitution reduces the degree of the denominator from 4 to 2 which is reflected in faster doublings.
Doublings on twisted Edwards curves in standard projective coordinates can be computed as

$$B = (X_1 + Y_1)^2;\ C = X_1^2;\ D = Y_1^2;\ E = aC; F = E + D;\ H = Z_1^2;$$
$$J = F - 2H;\ X_3 = (B - C - D) \cdot J;\ Y_3 = F \cdot (E - D);\ Z_3 = F \cdot J$$

in $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$, where the $1\mathbf{D}$ is a multiplication by a and $2H$ is computed as $H + H$.

1.3.3 Clearing denominators in projective coordinates

Here is an alternative approach to arithmetic on the twisted Edwards curve $E_{\mathrm{E},a,d}$ when a is a square in k.
The curve $E_{\mathrm{E},a,d} : a\bar{x}^2 + \bar{y}^2 = 1 + d\bar{x}^2\bar{y}^2$ is isomorphic to the Edwards curve $E_{\mathrm{E},1,d/a} : x^2 + y^2 = 1 + (d/a)x^2y^2$ by $x = \sqrt{a}\bar{x}$ and $y = \bar{y}$; see Remark 1.15. The following formulas add on $E_{\mathrm{E},1,d/a}$ using $10\mathbf{M} + 1\mathbf{S} + 3\mathbf{D} + 7\mathbf{a}$, where the $3\mathbf{D}$ are two multiplications by a and one by d:

$$A = Z_1 \cdot Z_2;\ B = aA^2;\ H = aA;\ C = X_1 \cdot X_2;\ D = Y_1 \cdot Y_2;\ E = dC \cdot D;$$
$$F = B - E;\ G = B + E;\ X_3 = H \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D);$$
$$Y_3 = H \cdot G \cdot (D - C);\ Z_3 = F \cdot G.$$

One can double on $E_{\mathrm{E},1,d/a}$ with $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$, independent of the curve coefficient d/a, using the formulas from [BL07b, Section 4] which is the case $a = 1$ in the previous subsection.
These addition formulas for $E_{\mathrm{E},1,d/a}$ are slower (by 1 multiplication by a) than the addition formulas for $E_{\mathrm{E},a,d}$. On the other hand, doubling for $E_{\mathrm{E},1,d/a}$ is faster (by 1 multiplication by a) than doubling for $E_{\mathrm{E},a,d}$. Some applications (such as batch signature verification) have more additions than doublings, while other applications have more doublings than additions, so all of the formulas are of interest.

1.3.4 Inverted twisted Edwards coordinates

Arithmetic in inverted twisted Edwards coordinates (or *inverted Edwards coordinates* if $a = 1$) saves $1\mathbf{M}$ in addition compared to standard coordinates.
The following formulas from [BBJ+08] generalize the formulas in [BL07c] to twisted Edwards curves.

Remark 1.39 (Addition in inverted twisted coordinates)**.** The following formulas compute $(X_3 : Y_3 : Z_3) = (X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2)$ in $9\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$, where the $2\mathbf{D}$ are one multiplication by a and one by d:

$$A = Z_1 \cdot Z_2;\ B = dA^2;\ C = X_1 \cdot X_2;\ D = Y_1 \cdot Y_2;\ E = C \cdot D;$$
$$H = C - aD;\ I = (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D;$$
$$X_3 = (E + B) \cdot H;\ Y_3 = (E - B) \cdot I;\ Z_3 = A \cdot H \cdot I.$$

Remark 1.40 (Doubling in inverted twisted coordinates)**.** The following formulas compute $(X_3 : Y_3 : Z_3) = 2(X_1 : Y_1 : Z_1)$ in $3\mathbf{M} + 4\mathbf{S} + 2\mathbf{D} + 6\mathbf{a}$, where the $2\mathbf{D}$ are one multiplication by a and one by $2d$:

$$A = X_1^2;\ B = Y_1^2;\ U = aB;\ C = A + U;\ D = A - U;$$
$$E = (X_1 + Y_1)^2 - A - B;\ X_3 = C \cdot D;\ Y_3 = E \cdot (C - 2dZ_1^2);\ Z_3 = D \cdot E.$$

Compared to doubling in inverted Edwards coordinates the inverted twisted coordinates need $1\mathbf{D}$ extra. Again, this is worthwhile if both a and d are small.

Remark 1.41 (Clearing denominators in inverted coordinates)**.** The following formulas add in inverted coordinates on $E_{\mathrm{E},1,d/a}$ using $9\mathbf{M} + 1\mathbf{S} + 3\mathbf{D} + 7\mathbf{a}$, where the $3\mathbf{D}$ are two multiplications by a and one by d:

$$A = Z_1 \cdot Z_2;\ B = dA^2;\ C = X_1 \cdot X_2;\ D = Y_1 \cdot Y_2;\ E = aC \cdot D;$$
$$H = C - D;\ I = (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D;$$
$$X_3 = (E + B) \cdot H;\ Y_3 = (E - B) \cdot I;\ Z_3 = aA \cdot H \cdot I.$$

1.3.5 Extended points

Hisil, Wong, Carter, and Dawson in [HWCD08] introduced extended addition formulas costing only $9\mathbf{M} + 1\mathbf{D} + 6\mathbf{a}$ for the coordinates defined in Remark 1.26:

$$A = X_1 \cdot X_2;\ B = Y_1 \cdot Y_2;\ C = Z_1 \cdot T_2;\ D = T_1 \cdot Z_2;$$
$$E = D + C;\ F = (X_1 - Y_1) \cdot (X_2 + Y_2) + B - A;\ G = B + aA;$$
$$H = D - C;\ X_3 = E \cdot F;\ Y_3 = G \cdot H;\ Z_3 = F \cdot G;\ T_3 = E \cdot H.$$

These formulas save $1\mathbf{S}$ by switching from inverted coordinates to *extended coordinates*, and an extra $1\mathbf{D}$ by switching from the Edwards addition law to the dual addition law. Hisil et al. also introduced addition formulas costing only $8\mathbf{M}$ for the case $a = -1$.

A doubling in extended coordinates loses $1\mathbf{M}$ for computing the extended output coordinate T_3. However, the doubling formulas make no use of the extended input coordinate T_1, so if the input is not used for anything else then the operation *producing* that input can skip the computation of T_1, saving $1\mathbf{M}$.

Scalar multiplication can be carried out as a series of operations on an accumulator P: doublings replace P by $2P$, and double-and-add operations replace P by $2P+Q$.

If P is in projective coordinates and the precomputed points Q are in extended coordinates then doubling costs $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ and double-and-add costs $(3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}) + (9\mathbf{M} + 1\mathbf{D})$, with the $1\mathbf{M}$ loss in doubling cancelled by the $1\mathbf{M}$ savings in addition. This mixture of projective coordinates and extended coordinates was suggested in [HWCD08] and is used in EECM-MPFQ which will be described in Chapter 3.

1.3.6 Tripling on Edwards curves

The article [BBLP07] describes tripling formulas for Edwards curves which are faster than one doubling followed by an addition.
By applying the curve equation as for doubling one obtains

$$[3](x_1, y_1) = \left(\frac{((x_1^2 + y_1^2)^2 - (2y_1)^2)x_1}{4(x_1^2 - 1)x_1^2 - (x_1^2 - y_1^2)^2}, \frac{((x_1^2 + y_1^2)^2 - (2x_1)^2)y_1}{-4(y_1^2 - 1)y_1^2 + (x_1^2 - y_1^2)^2} \right). \tag{1.4}$$

For tripling we present two sets of formulas to do this operation in standard Edwards coordinates. The first one costs $9\mathbf{M}+4\mathbf{S}$ while the second needs $7\mathbf{M}+7\mathbf{S}$. If the $\mathbf{S}/\mathbf{M}$ ratio is small, specifically below $2/3$, then the second set is better while for larger ratios the first one is to be preferred. In general we assume that one squaring $\mathbf{S}$ costs as much as $0.8\mathbf{M}$. We omit counting $\mathbf{a}$ as multiplications and squaring dominate the cost.
Here are $9\mathbf{M} + 4\mathbf{S}$ formulas for tripling:

$$A = X_1^2;\ B = Y_1^2;\ C = (2Z_1)^2;\ D = A + B;\ E = D^2;\ F = 2D \cdot (A - B);$$
$$G = E - B \cdot C;\ H = E - A \cdot C;\ I = F + H;\ J = F - G;$$
$$X_3 = G \cdot J \cdot X_1;\ Y_3 = H \cdot I \cdot Y_1;\ Z_3 = I \cdot J \cdot Z_1.$$

Here are $7\mathbf{M} + 7\mathbf{S}$ formulas for tripling:

$$A = X_1^2;\ B = Y_1^2;\ C = Z_1^2;\ D = A + B;\ E = D^2;\ F = 2D \cdot (A - B);$$
$$K = 4C;\ L = E - B \cdot K;\ M = E - A \cdot K;\ N = F + M;\ O = N^2;\ P = F - L;$$
$$X_3 = 2L \cdot P \cdot X_1;\ Y_3 = M \cdot ((N + Y_1)^2 - O - B);\ Z_3 = P \cdot ((N + Z_1)^2 - O - C).$$

Hisil, Carter, and Dawson [HCD07] independently found tripling formulas for Edwards curves, needing $9\mathbf{M} + 4\mathbf{S}$.
We do not state a generalization of these tripling formulas to twisted Edwards curves. The twisted-Edwards-curve coefficient a spoils both numerator and denominator of (1.4) and does not lead to competitive formulas. We will see in Chapter 2 that scalar multiplication with large scalars on Edwards curves only relies on fast addition and doubling.

1.3.7 Quintupling on Edwards curves

This section shows how to efficiently compute the 5-fold of a point on an Edwards curve. In [BBLP07] we presented two different versions which lead to the same result

but have a different complexity. The first version needs $17\mathbf{M} + 7\mathbf{S}$ while version 2 needs $14\mathbf{M} + 11\mathbf{S}$. The second version is better if $\mathbf{S}/\mathbf{M}$ is small.
The following formulas for quintupling need $17\mathbf{M} + 7\mathbf{S}$:

$$\begin{aligned}
&A = X_1^2;\ B = Y_1^2;\ C = Z_1^2;\ D = A + B;\\
&E = 2C - D;\ F = D \cdot (B - A);\ G = E \cdot ((X_1 + Y_1)^2 - D);\\
&H = F^2;\ I = G^2;\ J = H + I;\ K = H - I;\ L = J \cdot K;\\
&M = D \cdot E;\ N = E \cdot F;\ O = 2M^2 - J;\ P = 4N \cdot O;\\
&Q = 4K \cdot N \cdot (D - C);\ R = O \cdot J;\ S = R + Q;\ T = R - Q;\\
&X_5 = X_1 \cdot (L + B \cdot P) \cdot T;\ Y_5 = Y_1 \cdot (L - A \cdot P) \cdot S;\ Z_5 = Z_1 \cdot S \cdot T.
\end{aligned}$$

These formulas do not have minimal degree. The $17\mathbf{M} + 7\mathbf{S}$ variables X_5, Y_5, Z_5 in the last line above have total degree 33 in the initial variables X_1, Y_1, Z_1, even though one would expect degree 5^2. In fact, X_5, Y_5, Z_5 are all divisible by the degree-8 polynomial $((X_1^2-Y_1^2)^2+4Y_1^2(Z_1^2-Y_1^2))((X_1^2-Y_1^2)^2+4X_1^2(Z_1^2-X_1^2))$. Minimizing the number of operations led to better results for our extended polynomials.
The second set of formulas needing $14\mathbf{M} + 11\mathbf{S}$ for quintupling have even bigger degree, namely total degree 37:

$$\begin{aligned}
&A = X_1^2;\ B = Y_1^2;\ C = Z_1^2;\ D = A + B;\ E = 2C - D;\\
&F = A^2;\ G = B^2;\ H = F + G;\ I = D^2 - H;\ J = E^2;\\
&K = G - F;\ L = K^2;\ M = 2I \cdot J;\ N = L + M;\ O = L - M;\\
&P = N \cdot O;\ Q = (E + K)^2 - J - L;\ R = ((D + E)^2 - J - H - I)^2 - 2N;\\
&S = Q \cdot R;\ T = 4Q \cdot O \cdot (D - C);\ U = R \cdot N;\ V = U + T;\ W = U - T;\\
&X_5 = 2X_1 \cdot (P + B \cdot S) \cdot W;\ Y_5 = 2Y_1 \cdot (P - A \cdot S) \cdot V;\ Z_5 = Z_1 \cdot V \cdot W.
\end{aligned}$$

Note that only variables $A, \ldots, E$ have the same values in the two versions.

Chapter 2

Analyzing double-base elliptic-curve single-scalar multiplication

---

The most important operation for elliptic-curve cryptography is single-scalar multiplication, i.e., sending a point P to its m-fold $[m]P$. We can influence how to efficiently compute $[m]P$ by choosing a curve shape (e.g., Edwards curves, Weierstrass form), a coordinate system (e.g., inverted Edwards coordinates, Jacobian coordinates), algorithms such as double-and-add, double-base chains, or sliding-window methods. We will introduce these techniques and shapes in the following and show how they go together.
The motivation for the comparison described in this chapter was the finding of fast tripling formulas for Edwards curves, introduced in Section 1.3.6. Tripling itself is not a very interesting operation for cryptographic purposes. This chapter investigates for which curve shapes it makes sense to carry out scalar multiplication using a chain of doublings and triplings.
The main differences between the content of this chapter and [BBLP07] are the following.

- The background on Edwards curves is omitted as it was handled in the previous section.
- The tripling and quintupling formulas for Edwards curves were already given in Sections 1.3.6 and 1.3.7.
- Section 2.2 gives background on single-base chains in the form of the double-and-add algorithm. This part is added here to give a broader overview. However, the main focus of this chapter is the analysis of double-base chains in combination with various other techniques for various curve shapes.

The current speed records for elliptic-curve arithmetic are contained in Hisil's Ph.D. thesis [His10] and the "Explicit-Formulas Database" (EFD) [BL07a] by Bernstein and Lange. Also Imbert and Philippe conducted more research on double-base chains in [IP10] since [BBLP07] was published.
We nevertheless present the findings of [BBLP07] as they give a complete comparison between curve shapes and fastest formulas where we have to restrict "fastest" to end

2007. Note that Edwards curves and in particular inverted Edwards coordinates still provide one of the fastest formulas for elliptic-curve cryptography.

2.1 Fast addition on elliptic curves

There is a vast literature on elliptic-curve arithmetic. See [DL05, BSS99, HMV04] for overviews of efficient group operations on elliptic curves, and [BL07b, Section 6] for an analysis of Edwards-curve-scalar-multiplication performance without triplings.

Those overviews are not a satisfactory starting point for our analysis, because they do not include recent improvements in curve shapes and in addition formulas. Fortunately, all of the latest improvements have been collected into the "Explicit-Formulas Database" (EFD) [BL07a] by Bernstein and Lange, with Sage scripts verifying the correctness of the formulas. For example, this database also includes the tripling formulas from Section 1.3.6, the tripling formulas from [BL07c] for inverted Edwards coordinates, and the formulas from [HCD07] for other systems.

2.1.1 Jacobian coordinates

Let k be a field of characteristic at least 5. Every elliptic curve over k can then be written in short Weierstrass form $E : y^2 = x^3 + a_4x + a_6,\ a_4, a_6 \in k$, where $f(x) = x^3 + a_4x + a_6$ is squarefree. The set $E(k)$ of k-rational points of E is the set of tuples (x_1, y_1) satisfying the equation together with a point P_∞ at infinity.

The most popular representation of an affine point $(x_1, y_1) \in E(k)$ is as *Jacobian coordinates* $(X_1 : Y_1 : Z_1)$ satisfying $Y_1^2 = X_1^3 + a_4X_1Z_1^2 + a_6Z_1^6$ and $(x_1, y_1) = (X_1/Z_1^2, Y_1/Z_1^3)$. An *addition* of generic points $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ in Jacobian coordinates costs $11\mathbf{M} + 5\mathbf{S}$. A *readdition*—i.e., an addition where $(X_2 : Y_2 : Z_2)$ has been added before—costs $10\mathbf{M} + 4\mathbf{S}$, because Z_2^2 and Z_2^3 can be cached and reused. A *mixed addition*—i.e., an addition where Z_2 is known to be 1—costs $7\mathbf{M} + 4\mathbf{S}$. A *doubling*—i.e., an addition where $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ are known to be equal—costs $1\mathbf{M} + 8\mathbf{S}$. A *tripling* costs $5\mathbf{M} + 10\mathbf{S}$.

If $a_4 = -3$ then the cost for doubling changes to $3\mathbf{M} + 5\mathbf{S}$ and that for tripling to $7\mathbf{M} + 7\mathbf{S}$. Not every curve can be transformed to allow $a_4 = -3$ but important examples such as the NIST curves [P1300] make this choice. Note that the NIST recommendations speak of "projective curves"; we comment that these NIST curves have weights 2 and 3 for x and y. Thus those NIST curves are from now on refered to as Jacobian-3.

Most of the literature presents slower formulas producing the same output, and correspondingly reports higher costs for arithmetic in Jacobian coordinates. See, for example, [P1300, Section A.10.4] and the aforementioned overviews. We include the slower formulas in our experiments to simplify the comparison of our results to previous results in [DI06] and [DIM05] and to emphasize the importance of using faster formulas. We refer to the slower formulas as Std-Jac and Std-Jac-3.

Curve shape	ADD	reADD	mADD	DBL	TRI
3DIK	$11\mathbf{M}+6\mathbf{S}$	$10\mathbf{M}+6\mathbf{S}$	$7\mathbf{M}+4\mathbf{S}$	$2\mathbf{M}+7\mathbf{S}$	$6\mathbf{M}+6\mathbf{S}$
Edwards	$10\mathbf{M}+1\mathbf{S}$	$10\mathbf{M}+1\mathbf{S}$	$9\mathbf{M}+1\mathbf{S}$	$3\mathbf{M}+4\mathbf{S}$	$9\mathbf{M}+4\mathbf{S}$
ExtJQuartic	$8\mathbf{M}+3\mathbf{S}$	$8\mathbf{M}+3\mathbf{S}$	$7\mathbf{M}+3\mathbf{S}$	$3\mathbf{M}+4\mathbf{S}$	$4\mathbf{M}+11\mathbf{S}$
Hessian	$12\mathbf{M}+0\mathbf{S}$	$12\mathbf{M}+0\mathbf{S}$	$10\mathbf{M}+0\mathbf{S}$	$7\mathbf{M}+1\mathbf{S}$	$8\mathbf{M}+6\mathbf{S}$
InvEdwards	$9\mathbf{M}+1\mathbf{S}$	$9\mathbf{M}+1\mathbf{S}$	$8\mathbf{M}+1\mathbf{S}$	$3\mathbf{M}+4\mathbf{S}$	$9\mathbf{M}+4\mathbf{S}$
JacIntersect	$13\mathbf{M}+2\mathbf{S}$	$13\mathbf{M}+2\mathbf{S}$	$11\mathbf{M}+2\mathbf{S}$	$3\mathbf{M}+4\mathbf{S}$	$4\mathbf{M}+10\mathbf{S}$
Jacobian	$11\mathbf{M}+5\mathbf{S}$	$10\mathbf{M}+4\mathbf{S}$	$7\mathbf{M}+4\mathbf{S}$	$1\mathbf{M}+8\mathbf{S}$	$5\mathbf{M}+10\mathbf{S}$
Jacobian-3	$11\mathbf{M}+5\mathbf{S}$	$10\mathbf{M}+4\mathbf{S}$	$7\mathbf{M}+4\mathbf{S}$	$3\mathbf{M}+5\mathbf{S}$	$7\mathbf{M}+7\mathbf{S}$
Std-Jac	$12\mathbf{M}+4\mathbf{S}$	$11\mathbf{M}+3\mathbf{S}$	$8\mathbf{M}+3\mathbf{S}$	$3\mathbf{M}+6\mathbf{S}$	$9\mathbf{M}+6\mathbf{S}$
Std-Jac-3	$12\mathbf{M}+4\mathbf{S}$	$11\mathbf{M}+3\mathbf{S}$	$8\mathbf{M}+3\mathbf{S}$	$4\mathbf{M}+4\mathbf{S}$	$9\mathbf{M}+6\mathbf{S}$

Table 2.1: Cost of *addition* (ADD), *readdition* (reADD), *mixed addition* (mADD), *doubling* (DBL), and *tripling* (TRI) for various curve shapes.

2.1.2 More coordinate systems

Several other representations of elliptic curves have attracted attention because they offer faster group operations or extra features such as unified addition formulas that also work for doublings. Some of these representations can be reached through isomorphic transformation for any curve in Weierstrass form while others require, for example, a point of order 4. Our analysis includes all of the curve shapes listed in Table 2.1. "ExtJQuartic" and "Hessian" and "JacIntersect" refer to the latest addition formulas for Jacobi quartics $Y^2 = X^4 + 2aX^2Z^2 + Z^4$, Hessian curves $X^3 + Y^3 + Z^3 = 3dXYZ$, and Jacobi intersections $S^2 + C^2 = T^2, aS^2 + D^2 = T^2$. The EFD [BL07a] takes the improvements in [Duq07] and [HCD07] into account.

"3DIK" is an abbreviation for "tripling-oriented Doche/Icart/Kohel curves," the curves $Y^2 = X^3 + a(X + Z^2)^2Z^2$ introduced in [DIK06]. (The same article also introduces doubling-oriented curves that do not have fast additions or triplings and that are omitted from our comparison.)

We note that [DIK06] states incorrect formulas for doubling. The corrected and faster formulas are:

$$\begin{aligned} B &= X_1^2;\ C = 2a \cdot Z_1^2 \cdot (X_1 + Z_1^2);\ D = 3(B + C);\ E = Y_1^2;\ F = E^2; \\ Z_3 &= (Y_1 + Z_1)^2 - E - Z_1^2;\ G = 2((X_1 + E)^2 - B - F); \\ X_3 &= D^2 - 3a \cdot Z_3^2 - 2G;\ Y_3 = D \cdot (G - X_3) - 8F; \end{aligned}$$

which are now also included in the EFD [BL07a].

2.2 Double-base chains for single-scalar multiplication

This section reviews the previous state of the art in double-base chains for computing $[m]P$ given P. Abusing notation we write mP instead of $[m]P$ for the sake of simplicity.
The following computation methods can be used to compute multiples of elements in arbitrary additive abelian groups. So, we do not rely on any curve shape or coordinate system in this section.

2.2.1 Single-base scalar multiplication

The classical double-and-add algorithm computes mP by carrying out doublings and additions depending on the binary representation of the scalar m.

Algorithm 2.1: Double-and-add algorithm

Input: A point P on an elliptic curve E and a scalar $m > 0$ in its binary representation $(m_{\ell-1} \dots m_0)_2$.
Output: The point $mP \in E$.

1: $Q \leftarrow \mathcal{O}$ and $i \leftarrow \ell - 1$
2: **While** $i \geq 0$
3: $\quad Q \leftarrow 2Q$
4: $\quad$ **If** $m_i = 1$ **then**
5: $\quad\quad Q \leftarrow P + Q$
6: $\quad i \leftarrow i - 1$
7: **Return** Q

Consider the scalar $314159 = 2^{18}+2^{15}+2^{14}+2^{11}+2^9+2^8+2^5+2^3+2^2+2^1+2^0$. The "base-2" chain

$$\begin{aligned} 314159P = 2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2P)) \\ +P)+P)))+P))+P)+P)))+P))+P)+P)+P)+P \end{aligned}$$

computes $314159P$ using the double-and-add method, starting from P, with a chain of 18 doublings and 10 additions of P.
The first step to save arithmetic operations is to allow subtractions in the computation chain. This saves two additions in our example (note that we count subtractions as additions):

$$\begin{aligned} 314159P = 2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2P) \\ +P))-P)))+P)+P))-P)))+P)+P))))-P. \end{aligned}$$

One can express this chain more concisely — with an implicit application of Horner's rule — as

$$314159P = 2^{18}P + 2^{16}P - 2^{14}P + 2^{11}P + 2^{10}P - 2^8P + 2^5P + 2^4P - 2^0P.$$

Any such representation with a single base point is called a *single-base chain*.
For a reader interested in speeding up elliptic-curve scalar multiplication using single-base chains we refer to [DL05]; a more recent analysis of the use of single-base chains which also includes Edwards curves we refer to [BL08]. The remainder of this chapter deals with double-base chains which will be introduced in the following.

2.2.2 Double-base scalar multiplication

The "double-base-2-and-3" equation

$$\begin{aligned}314159P &= 2^{15}3^2P + 2^{11}3^2P + 2^8 3^1 P + 2^4 3^1 P - 2^0 3^0 P \\ &= 3(2(2(2(2(2(2(3(2(2(2(2(2(2(P)))) + P)))) + P)))) + P))))) - P\end{aligned}$$

can be viewed as a better algorithm to compute $314159P$, starting from P, with a chain of 2 triplings, 15 doublings, and 4 additions of P. If 1 tripling has the same cost as 1 doubling and 1 addition then this chain has the same cost as 17 doublings and 6 additions which is fewer operations than the 18 doublings and 8 additions of P needed in the base-2 expansion.
One can object to this comparison by pointing out that adding mP for $m > 1$ is more expensive than adding P — typically P is provided in affine form, allowing a mixed addition of P, while mP requires a more expensive non-mixed addition — so a tripling is more expensive than a doubling and an addition of P. But this objection is amply answered by dedicated tripling formulas that are *less* expensive than a doubling and an addition. See Sections 1.3 and 2.1.
Double-base chains were introduced by Dimitrov, Imbert, and Mishra in an article [DIM05] at Asiacrypt 2005. There were several previous "double-base number system" articles expanding mP in various ways as $\sum c_i 2^{a_i} 3^{b_i} P$ with $c_i \in \{-1, 1\}$; the critical advance in [DIM05] was to require $a_1 \geq a_2 \geq a_3 \geq \cdots$ and $b_1 \geq b_2 \geq b_3 \geq \cdots$, allowing a straightforward chain of doublings and triplings without the expensive backtracking that plagued previous articles.
There are a few issues in comparing single base to double base. One can object that the benefit of fast double-base chains is outweighed by the cost of finding those chains. Perhaps this objection will be answered someday by an optimized algorithm that finds a double-base chain in less time than is saved by applying that chain. We rely on a simpler answer: we focus on cryptographic applications in which the same m is used many times (as in [DH76, Section 3]), allowing the chain for m to be constructed just once and then reused. The software used for the experiments in [BBLP07] has not been heavily optimized but takes under a millisecond to compute an expansion of a cryptographic-size integer m.
A more troubling objection is that the simple base-2 chains described above were obsolete before the advent of double-base chains. Typical speed-oriented elliptic-curve software instead uses "sliding window" base-2 chains that use marginally more temporary storage but considerably fewer additions — see below. Even if double-base chains are faster than obsolete base-2 chains, there is no reason to believe that they are faster than state-of-the-art sliding-window base-2 chains.

2.2.3 Sliding window

The "sliding-window base-2" equation

$$\begin{aligned}314159P &= 2^{16}5P - 2^{11}7P + 2^8 3P + 2^4 3P - 2^0 P \\ &= 2(2(2(2(2(2(2(2(2(2(2(2(2(5P))))) - 7P))) + 3P)))) + 3P)))) - P\end{aligned}$$

can be viewed as an algorithm to compute $314159P$ starting from $\{P, 3P, 5P, 7P\}$ with a chain of 16 doublings and 4 additions. It can therefore be viewed as an algorithm to compute $314159P$, starting from P, with 17 doublings and 7 additions; this operation count includes the obvious chain of 1 doubling and 3 additions to produce $2P, 3P, 5P, 7P$ from P.

The idea of starting with $\{P, 2P, 3P, 4P, \ldots, (2^w - 1)P\}$ ("fixed length-w window") was introduced by Brauer long ago in [Bra39]. By optimizing the choice of w as a function of the bit length ℓ, Brauer showed that one can compute mP for an ℓ-bit integer m using $\approx \ell$ doublings and at most $\approx \ell/\log_2 \ell$ additions (even without subtractions). The idea to start with $\{P, 2P, 3P, 5P, 7P, \ldots, (2^w - 1)P\}$ ("sliding length-w window") was introduced by Thurber in [Thu73], saving some additions. For comparison, the simple base-2 chains considered earlier use $\approx \ell$ doublings and $\approx \ell/3$ additions (on average; as many as $\ell/2$ in the worst case). The benefit of the sliding-window method increases slowly with ℓ.

Doche and Imbert, in their article [DI06] at Indocrypt 2006, introduced an analogous improvement to double-base chains. Example: The "sliding-window double-base-2-and-3" equation

$$\begin{aligned}314159P &= 2^{12}3^3 3P - 2^7 3^3 5P - 2^4 3^1 7P - 2^0 3^0 P \\ &= 3(2(2(2(2(3(3(2(2(2(2(2(2(2(3P))))) - 5P))))) - 7P))))) - P\end{aligned}$$

can be viewed as an algorithm to compute $314159P$ starting from $\{P, 3P, 5P, 7P\}$ with a chain of 3 triplings, 12 doublings, and 3 additions. It can therefore be viewed as an algorithm to compute $314159P$, starting from P, with 3 triplings, 13 doublings, and 6 additions. (The set $\{P, 3P, 5P, 7P\}$ was not considered in [DI06]; we use this example to emphasize the analogy between single-base chains and double-base chains.)

Doche and Imbert state an algorithm to compute double-base chains for arbitrary coefficient sets S containing 1. In their experiments they focus on sets of the form $\{1, 2, 3, 2^2, 3^2, \ldots, 2^k, 3^k\}$ or sets of odd integers co-prime to 3. In this chapter we study several coefficient sets including all sets considered in [DI06] and additional sets such as $\{P, 2P, 3P, 5P, 7P\}$.

2.2.4 Computing a chain

Finding the chain $314159 = 2^{18} + 2^{16} - 2^{14} + 2^{11} + 2^{10} - 2^8 + 2^5 + 2^4 - 2^0$ is a simple matter of finding the closest power of 2 to 314159, namely $2^{18} = 262144$; then finding the closest power of 2 to the difference $|314159 - 262144| = 52015$, namely $2^{16} = 65536$; and so on.

Similarly, by inspecting the first few bits of a nonzero integer m one can easily see which of the integers

$$\begin{array}{llllll} \pm 1, & \pm 2, & \pm 2^2, & \pm 2^3, & \pm 2^4, & \dots \\ \pm 3, & \pm 2 \cdot 3, & \pm 2^2 3, & \pm 2^3 3, & \pm 2^4 3, & \dots \\ \pm 5, & \pm 2 \cdot 5, & \pm 2^2 5, & \pm 2^3 5, & \pm 2^4 5, & \dots \\ \pm 7, & \pm 2 \cdot 7, & \pm 2^2 7, & \pm 2^3 7, & \pm 2^4 7, & \dots \end{array}$$

is closest to m. By subtracting that integer from m and repeating the same process one expands m into Thurber's base-2 sliding-window chain $\sum_i c_i 2^{a_i}$ with $\pm c_i \in \{1, 3, 5, 7\}$ and $a_1 > a_2 > a_3 > \cdots$. For example, $2^{16} \cdot 5 = 327680$ is closest to 314159; $-2^{11} \cdot 7 = -14336$ is closest to $314159 - 327680 = -13521$; continuing in the same way one finds the chain $314159 = 2^{16}5P - 2^{11}7P + 2^8 3P + 2^4 3P - 2^0 P$ shown above. Similar comments apply to sets other than $\{1, 3, 5, 7\}$.

Dimitrov, Imbert, and Mishra in [DIM05, Section 3] proposed a similar, although slower, algorithm to find double-base chains with $c_i \in \{-1, 1\}$; Doche and Imbert in [DI06, Section 3.2] generalized the algorithm to allow a wider range of c_i. For example, given m and the set $\{1, 3, 5, 7\}$, the Doche-Imbert algorithm finds the product $c_1 2^{a_1} 3^{b_1}$ closest to m, with $\pm c_1 \in \{1, 3, 5, 7\}$, subject to limits on a_1 and b_1; it then finds the product $c_2 2^{a_2} 3^{b_2}$ closest to $m - c_1 2^{a_1} 3^{b_1}$, with $\pm c_2 \in \{1, 3, 5, 7\}$, subject to the chain conditions $a_1 \geq a_2$ and $b_1 \geq b_2$; continuing in this way it expands m as $\sum_i c_i 2^{a_i} 3^{b_i}$ with $\pm c_i \in \{1, 3, 5, 7\}$, $a_1 \geq a_2 \geq \cdots$, and $b_1 \geq b_2 \geq \cdots$.

(The algorithm statements in [DIM05] and [DI06] are ambiguous on the occasions that m is equally close to two or more products $c2^a 3^b$. Which (c, a, b) is chosen? In our experiments, when several $c2^a 3^b$ are equally close to m, we choose the first (c, b, a) in lexicographic order: we prioritize a small c, then a small b, then a small a.)

The worst-case and average-case chain lengths produced by this double-base algorithm are difficult to analyze mathematically. However, the average chain length for all m's can be estimated with high confidence as the average chain length seen for a large number of m's. Dimitrov, Imbert, and Mishra used 10000 integers m for each of their data points; Doche and Imbert used 1000. The next section discusses the experiments carried out for [BBLP07]; those experiments use 10000 uniform random integers.

2.3 Optimizing double-base elliptic-curve single-scalar multiplication

This section describes the experiments carried out in 2007 for [BBLP07] and the achieved multiplication counts. The results of the experiments are presented as a table and a series of graphs.

2.3.1 Parameter space

Our experiments included several bit sizes ℓ, namely 160, 200, 256, 300, 400, and 500. The choices $200, 300, 400, 500$ were used in [DI06] and we include them to ease comparison. The choices 160 and 256 are common in cryptographic applications.
Our experiments included the eight curve shapes described in Section 2.1: 3DIK, Edwards, ExtJQuartic, Hessian, InvEdwards, JacIntersect, Jacobian, and Jacobian-3. For comparison with previous results, and to show the importance of optimized curve formulas, we also carried out experiments for Std-Jac and Std-Jac-3.
Our experiments included many choices of the parameter a_0 in [DI06, Algorithm 1]. The largest power of 2 allowed in the algorithm is 2^{a_0}, i.e., a_0 is an upper bound for a_1 in the chain representation. The largest power of 3 allowed in the algorithm is 3^{b_0} where $b_0 = \lceil (\ell - a_0)/\log_2 3 \rceil$; and $b_1 \leq b_0$. Specifically, we tried each $a_0 \in \{0, 10, 20, \ldots, 10\lfloor \ell/10 \rfloor\}$. This matches the experiments reported in [DI06] for $\ell = 200$. We also tried all integers a_0 between 0.95ℓ and 1.00ℓ.
Our experiments included several sets S, i.e., sets of coefficients c allowed in $c2^a3^b$: the set $\{1\}$ used in [DIM05]; the sets $\{1, 2, 3\}$, $\{1, 2, 3, 4, 8, 9, 16, 27, 81\}$, $\{1, 5, 7\}$, $\{1, 5, 7, 11, 13, 17, 19, 23, 25\}$ appearing in the graphs in [DI06, Appendix B] with labels "$(1, 1)$" and "$(4, 4)$" and "S_2" and "S_8"; and $\{1, 2, 3, 4, 9\}$, $\{1, 2, 3, 4, 8, 9, 27\}$, $\{1, 5\}$, $\{1, 5, 7, 11\}$, $\{1, 5, 7, 11, 13\}$, $\{1, 5, 7, 11, 13, 17, 19\}$ appearing in the tables in [DI06, Appendix B]. We also included the sets $\{1, 2, 3, 5\}$, $\{1, 2, 3, 5, 7\}$, and so on through $\{1, 2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25\}$; these sets are standard in the base-2 context but do not seem to have been included in previous double-base experiments. Additionally, we took the same sets as mentioned before, but excluded 2 as a window element; e.g., we chose $\{1, 3\}$, $\{1, 3, 5\}$, and so forth.
We used straightforward combinations of additions, doublings, and triplings for the initial computation of cP for each $c \in S$.
We follow the standard (although debatable) practice of counting $\mathbf{S} = 0.8\mathbf{M}$ and disregarding other field operations. We caution the reader that other weightings of field operations can easily change the order of two systems with similar levels of performance.

2.3.2 Experiments and results

There are 8236 combinations of ℓ, a_0, and S described above. For each combination, we

- generated 10000 uniform random integers $n \in \{0, 1, \ldots, 2^\ell - 1\}$,
- converted each integer into a chain as specified by a_0 and S,
- checked that the chain indeed computed n starting the chain from 1, and
- counted the number of triplings, doublings, additions, readditions, and mixed additions for those 10000 choices of n.

We converted the results into multiplication counts for the curve shapes 3DIK, Edwards, ExtJQuartic, Hessian, InvEdwards, JacIntersect, Jacobian, Jacobian-3, Std-Jac, and Std-Jac-3, obtaining a cost for each of the 82360 combinations of ℓ, curve shape, a_0, and S.

Figure 2.1 shows, for each ℓ (horizontal axis) and each curve shape, the minimum cost per bit obtained when a_0 and S are chosen optimally. The implementor can easily read off the ranking of coordinate systems from this graph. Table 2.2 displays the same information in tabular form, along with the choices of a_0 and S.

There is no unique optimal choice of a_0 and S for every curve shape which gives rise to the fastest computation of a given ℓ-bit integer. For example, using Jacobian coordinates the best result is achieved by precomputing odd coefficients up to 13 for an integer of bit length at most 300. For 400-bit integers the optimum uses $S = \{1, 2, 3, 5, \ldots, 17\}$ and in the 500-bit case also 19 is included.

None of the optimal results for $\ell \geq 200$ uses a set of precomputed points discussed in [DIM05] or [DI06]. The optimal coefficient sets in every case were those used in (fractional) sliding-window methods, i.e. the sets $\{1, 2, 3, 5, \ldots\}$.

Figure 2.2 shows, for each a_0 (horizontal axis) and each curve shape, the cost for $\ell = 256$ when S is chosen optimally. This graph demonstrates the importance of choosing the right bounds for a_0 and b_0 depending on the ratio of the doubling/tripling costs. We refer to Table 2.2 for the best choices of a_0 and S for each curve shape.

Table 2.2: Optimal parameters for each curve shape and each ℓ

ℓ	Curve shape	Mults	Mults/ℓ	a_0	a_0/ℓ	S
160	3DIK	1502.393800	9.389961	80	0.5	$\{1\}$
200	3DIK	1879.200960	9.396005	100	0.5	$\{1, 2, 3, 5, 7\}$
256	3DIK	2393.193800	9.348413	130	0.51	$\{1, 2, 3, 5, \ldots, 13\}$
300	3DIK	2794.431020	9.314770	160	0.53	$\{1, 2, 3, 5, \ldots, 13\}$
400	3DIK	3706.581360	9.266453	210	0.53	$\{1, 2, 3, 5, \ldots, 13\}$
500	3DIK	4615.646620	9.231293	270	0.54	$\{1, 2, 3, 5, \ldots, 17\}$
160	Edwards	1322.911120	8.268194	156	0.97	$\{1, 2, 3, 5, \ldots, 13\}$
200	Edwards	1642.867360	8.214337	196	0.98	$\{1, 2, 3, 5, \ldots, 15\}$
256	Edwards	2089.695120	8.162872	252	0.98	$\{1, 2, 3, 5, \ldots, 15\}$
300	Edwards	2440.611880	8.135373	296	0.99	$\{1, 2, 3, 5, \ldots, 15\}$
400	Edwards	3224.251900	8.060630	394	0.98	$\{1, 2, 3, 5, \ldots, 25\}$
500	Edwards	4005.977080	8.011954	496	0.99	$\{1, 2, 3, 5, \ldots, 25\}$
160	ExtJQuartic	1310.995340	8.193721	156	0.97	$\{1, 2, 3, 5, \ldots, 13\}$
200	ExtJQuartic	1628.386660	8.141933	196	0.98	$\{1, 2, 3, 5, \ldots, 15\}$
256	ExtJQuartic	2071.217580	8.090694	253	0.99	$\{1, 2, 3, 5, \ldots, 15\}$
300	ExtJQuartic	2419.026660	8.063422	299	1	$\{1, 2, 3, 5, \ldots, 21\}$
400	ExtJQuartic	3196.304940	7.990762	399	1	$\{1, 2, 3, 5, \ldots, 25\}$
500	ExtJQuartic	3972.191800	7.944384	499	1	$\{1, 2, 3, 5, \ldots, 25\}$

Continued on next page

Table 2.2 – continued from previous page

ℓ	Curve shape	Mults	Mults/ℓ	a_0	a_0/ℓ	S
160	Hessian	1560.487660	9.753048	100	0.62	$\{1, 2, 3, 5, \ldots, 13\}$
200	Hessian	1939.682780	9.698414	120	0.6	$\{1, 2, 3, 5, \ldots, 13\}$
256	Hessian	2470.643200	9.650950	150	0.59	$\{1, 2, 3, 5, \ldots, 13\}$
300	Hessian	2888.322160	9.627741	170	0.57	$\{1, 2, 3, 5, \ldots, 13\}$
400	Hessian	3831.321760	9.578304	240	0.6	$\{1, 2, 3, 5, \ldots, 17\}$
500	Hessian	4772.497740	9.544995	300	0.6	$\{1, 2, 3, 5, \ldots, 19\}$
160	InvEdwards	1290.333920	8.064587	156	0.97	$\{1, 2, 3, 5, \ldots, 13\}$
200	InvEdwards	1603.737760	8.018689	196	0.98	$\{1, 2, 3, 5, \ldots, 15\}$
256	InvEdwards	2041.223320	7.973529	252	0.98	$\{1, 2, 3, 5, \ldots, 15\}$
300	InvEdwards	2384.817880	7.949393	296	0.99	$\{1, 2, 3, 5, \ldots, 15\}$
400	InvEdwards	3152.991660	7.882479	399	1	$\{1, 2, 3, 5, \ldots, 25\}$
500	InvEdwards	3919.645880	7.839292	496	0.99	$\{1, 2, 3, 5, \ldots, 25\}$
160	JacIntersect	1438.808960	8.992556	150	0.94	$\{1, 2, 3, 5, \ldots, 13\}$
200	JacIntersect	1784.742200	8.923711	190	0.95	$\{1, 2, 3, 5, \ldots, 15\}$
256	JacIntersect	2266.135540	8.852092	246	0.96	$\{1, 2, 3, 5, \ldots, 15\}$
300	JacIntersect	2644.233000	8.814110	290	0.97	$\{1, 2, 3, 5, \ldots, 15\}$
400	JacIntersect	3486.773860	8.716935	394	0.98	$\{1, 2, 3, 5, \ldots, 25\}$
500	JacIntersect	4324.718620	8.649437	492	0.98	$\{1, 2, 3, 5, \ldots, 25\}$
160	Jacobian	1558.405080	9.740032	100	0.62	$\{1, 2, 3, 5, \ldots, 13\}$
200	Jacobian	1937.129960	9.685650	130	0.65	$\{1, 2, 3, 5, \ldots, 13\}$
256	Jacobian	2466.150480	9.633400	160	0.62	$\{1, 2, 3, 5, \ldots, 13\}$
300	Jacobian	2882.657400	9.608858	180	0.6	$\{1, 2, 3, 5, \ldots, 13\}$
400	Jacobian	3819.041260	9.547603	250	0.62	$\{1, 2, 3, 5, \ldots, 17\}$
500	Jacobian	4755.197420	9.510395	310	0.62	$\{1, 2, 3, 5, \ldots, 19\}$
160	Jacobian-3	1504.260200	9.401626	100	0.62	$\{1, 2, 3, 5, \ldots, 13\}$
200	Jacobian-3	1868.530560	9.342653	130	0.65	$\{1, 2, 3, 5, \ldots, 13\}$
256	Jacobian-3	2378.956000	9.292797	160	0.62	$\{1, 2, 3, 5, \ldots, 13\}$
300	Jacobian-3	2779.917220	9.266391	200	0.67	$\{1, 2, 3, 5, \ldots, 17\}$
400	Jacobian-3	3681.754460	9.204386	260	0.65	$\{1, 2, 3, 5, \ldots, 17\}$
500	Jacobian-3	4583.527180	9.167054	330	0.66	$\{1, 2, 3, 5, \ldots, 21\}$

The fastest systems are Edwards, ExtJQuartic, and InvEdwards. They need the lowest number of multiplications for values of a_0 very close to ℓ. These systems are using larger sets of precomputations than slower systems such as Jacobian-3 or Jacobian, and fewer triplings. The faster systems all come with particularly fast addition laws, making the precomputations less costly, and particularly fast doublings, making triplings less attractive. This means that currently double-base chains offer no or very little advantage for the fastest systems. See [BL07b] for a detailed description of single-base scalar multiplication on Edwards curves.

Not every curve can be represented by one of these fast systems. For curves in Jacobian coordinates values of a_0 around 0.6ℓ seem optimal and produce significantly

faster scalar multiplication than single-base representations.
Figure 2.3 shows, for a smaller range of a_0 (horizontal axis) and each choice of S, the cost for Jacobian-3 coordinates for $\ell = 200$. This graph demonstrates several interesting interactions between the doubling/tripling ratio, the choice of S, and the final results. Figure 2.4 is a similar graph for Edwards curves. The optimal scalar-multiplication method in that graph uses $a_0 \approx 195$ with coefficients in the set $\pm\{1, 2, 3, 5, 7, 11, 13, 15\}$. The penalty for using standard single-base sliding-window methods is negligible. On the other hand, triplings are clearly valuable if storage for precomputed points is extremely limited.

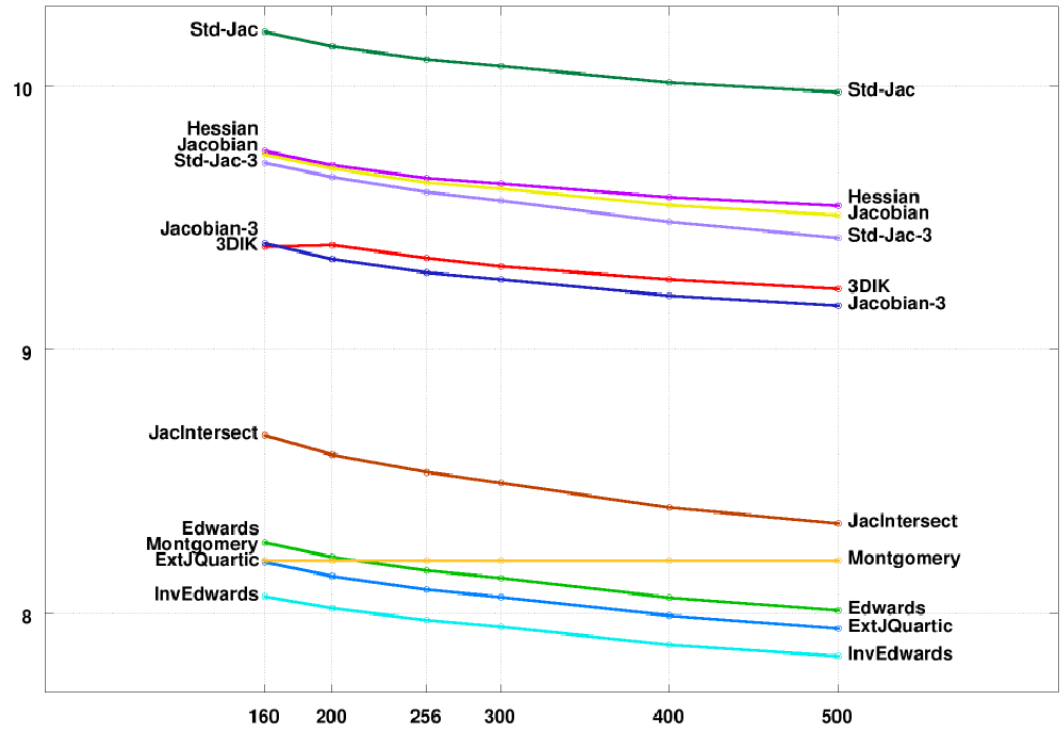

Figure 2.1: Multiplications per bit (all bits, all shapes). Horizontal axis is ℓ; the vertical axis gives the minimum cost per bit obtained when a_0 and S are chosen optimally.

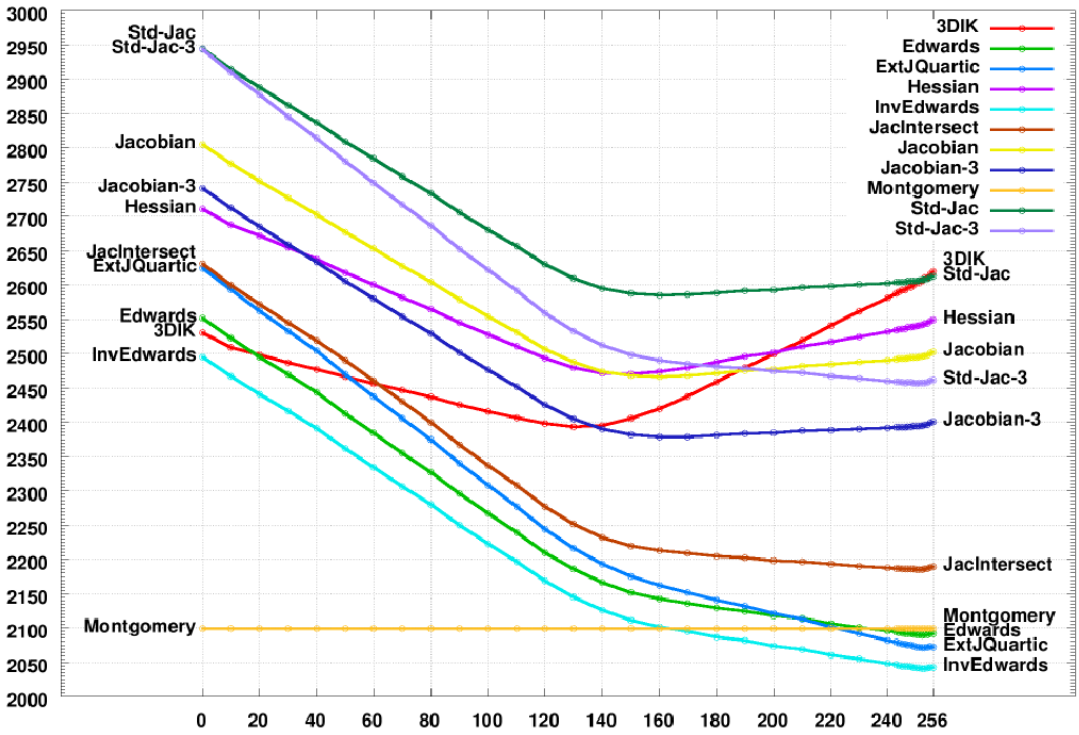

Figure 2.2: Importance of doubling/tripling ratio (256 bits, all shapes). Horizontal axis is a_0; the vertical axis gives the cost for $\ell = 256$ when S is chosen optimally.

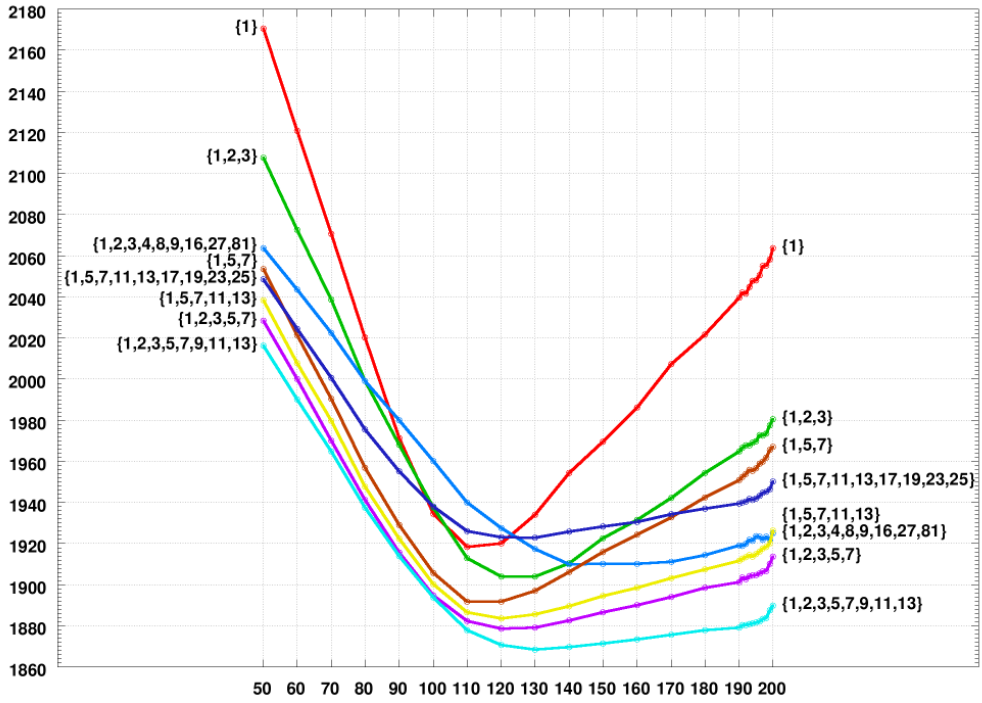

Figure 2.3: Importance of parameter choices (200 bits, Jacobian-3). The horizontal axis is a_0; the vertical axis gives the cost for Jacobian-3 coordinates for $\ell = 200$ and each choice of S.

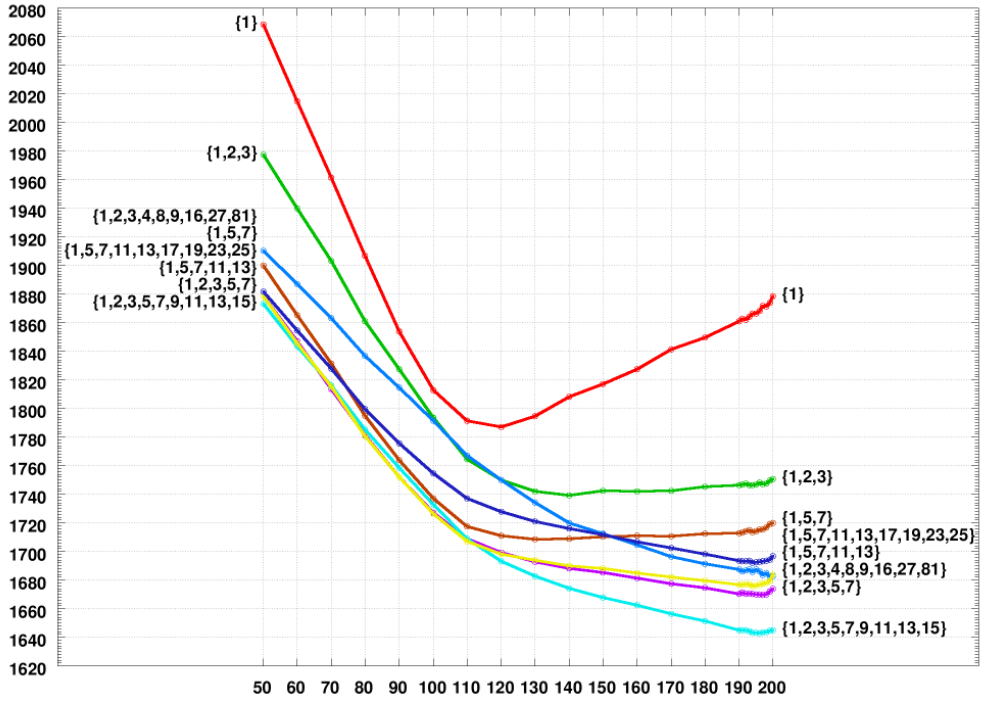

Figure 2.4: Importance of parameter choices (200 bits, Edwards). The horizontal axis is a_0; the vertical axis gives the cost for Edwards curves for $\ell = 200$ and each choice of S.

Chapter 3

ECM using Edwards curves

Factorization of integers is one of the most studied problems in algorithmic number theory and cryptology. One of the best general factorization methods available is the *Elliptic-Curve Method* (ECM), introduced in the 1987 article [Len87b] by Hendrik W. Lenstra, Jr. ECM plays an important role in factoring the "random" integers of interest to number theorists: it is not as fast as trial division and Pollard's rho method for finding tiny prime factors, but it is the method of choice for finding medium-size prime factors. ECM also plays an important role in factoring the "hard" integers of interest to cryptologists: those integers are attacked by sieving methods, which use ECM to find medium-size prime factors of auxiliary integers. ECM can also be used directly to find "large" prime factors; the current record (see [Zima]) is a 241-bit factor of the 1181-bit number $2^{1181} - 1$.
Implementations of ECM are available in most computer-algebra packages and have been the subject of countless articles. The state-of-the-art implementation is GMP-ECM [Z+10] which uses the GMP library [Gra] and which is described in detail in the article by Zimmermann and Dodson [ZD06].
This chapter discusses how to use Edwards curves for the Elliptic-Curve Method for factorization. The results presented here are based on the article "ECM using Edwards curves" [BBLP08] which is joint work with Bernstein, Birkner, and Lange. Bernstein implemented ECM with Edwards curves using the techniques described in [BBLP08] and also in parts in [BBL10]. The implementation uses the MPFQ library [GT] and is called "EECM-MPFQ" [Ber10a]. This chapter presents the mathematical background of EECM-MPFQ.
The main differences between the content of this chapter and [BBLP08] are the following.

- The background on Edwards curves in [BBLP08] is omitted from this chapter since it is treated in Chapter 1 of this thesis.
- This thesis omits parts of the discussion of EECM-MPFQ and omits the discussion of HECM which both can be found in [BBLP08, Section 4].
- This thesis omits the speedups for stage 2 of ECM for Edwards curves [BBLP08, Section 5], the translation of the Montgomery construction to Edwards curves [BBLP08, Section 7.7], and also the discussion of how to choose parameters [BBLP08, Section 10].

- This thesis contains an introduction to Pollard's $(p-1)$-method and a streamlined description of stage 1 of ECM. The description of the strategies for ECM's stage 1 in Section 3.1 essentially follows [BBLP08, Section 4].
- Section 3.2 elaborates on Mazur's theorem concerning the torsion groups of elliptic curves over the rationals. Also this thesis discusses in more detail how Kruppa's "$\sigma = 11$"-case relates to Edwards curves.

3.1 The Elliptic-Curve Method (ECM)

In this section we first describe Pollard's $(p-1)$-method which predates ECM and helps to understand ECM. The relation is that in ECM the multiplicative group is replaced by the group of rational points on an elliptic curve.

3.1.1 Pollard's $(p-1)$-method

Definition 3.1 (Smoothness)**.** A positive integer is called *B-smooth* if it is not divisible by any prime which is larger than a given bound $B \in \mathbb{Z}$. The integer B is called *smoothness bound*.

Algorithm 3.1: Pollard's $(p-1)$-method (stage 1)

Input: An integer $n > 1$ and a bound B_1.
Output: A nontrivial factor d of n, or "error."

1: $s \leftarrow \operatorname{lcm}\{1, 2, \ldots, B_1\}$
2: Choose an integer $0 < a < n$ uniformly at random.
3: $b \leftarrow a^s \pmod{n}$
4: $d \leftarrow \gcd(b-1, n)$
5: **If** $1 < d < n$ **then return** d.
6: **Else return** "error."

Pollard proposed two stages for his algorithm. Algorithm 3.1 describes "stage 1." Note that s is chosen as the least common multiple of integers up to B_1. Remark 3.5 in the following section discusses alternative ways of choosing s.
Stage 1 of the algorithm hopes for n to have a prime divisor p such that $a^s = 1$ in $(\mathbb{Z}/p\mathbb{Z})^*$. The algorithm is most likely to find p if $p-1$ is B_1-smooth. Then, by construction of s, it follows from Fermat's theorem that $a^s = 1$ in $(\mathbb{Z}/p\mathbb{Z})^*$. The computation of the gcd of $a^s - 1$ and n then reveals either p or a multiple of p. In the worst case the gcd equals n itself. This happens if n has many factors q with $q-1$ being B_1-smooth.
The second stage hopes for n to have a prime divisor p such that a^s has small prime order in $(\mathbb{Z}/p\mathbb{Z})^*$; specifically, order ℓ for some prime ℓ between B_1 and a second (upper) bound B_2. The state-of-the-art implementation of the $(p-1)$-method is described in Kruppa's Ph.D. thesis [Kru10] and uses in particular speedups of stage 2 described in [MK08]. The $(p-1)$-implementation is contained in the GMP-ECM

package [Z+10]. Zimmermann [Zimb] maintains a website listing the 10 largest factors found by the $(p-1)$-method. The current record is a 219-bit factor of the 1179-bit number $960^{119} - 1$.

3.1.2 Stage 1 of ECM

This section reviews the general idea of stage 1 of ECM and the state-of-the-art strategies used in GMP-ECM to perform the elliptic-curve computations in stage 1. Also a short outlook on stage 2 is given.

Algorithm 3.2: ECM stage 1

Input: An integer $n > 1$ and a bound B_1.
Output: A nontrivial factor d of n.

1: Choose a B_1-smooth integer s.
2: Choose an elliptic curve E defined over $\mathbb{Q}$.
3: Choose a rational function $\phi : E \to \mathbb{Q}$ that has a pole at the neutral element of E; for example choose ϕ as the Weierstrass x-coordinate.
4: Choose a point $P \in E(\mathbb{Q})$.
5: Try to compute $\phi([s]P)$ modulo n. Hope for an impossible division modulo n which should reveal a factor d of n. **Return** d.
6: **If** *no divisor can be revealed* **then** go back to Step 2.

Remark 3.2. Step 5 is carried out by choosing a sequence of additions, subtractions, multiplications, and divisions that, if carried out over $\mathbb{Q}$, would compute $\phi([s]P)$. Since $\mathbb{Z}/n\mathbb{Z}$ is a ring the algorithm only attempts to compute $\phi([s]P)$ modulo n. For a careful description of elliptic curves defined over rings we refer to Lenstra's article [Len87a].

Remark 3.3. An attempt to divide by a nonzero nonunit modulo n immediately reveals a factor of n. An attempt to divide by 0 modulo n is not quite as informative but usually allows a factor of n to be obtained without much extra work.

If n has a prime divisor q such that $[s]P$ is the neutral element of $E(\mathbb{Z}/q\mathbb{Z})$ then the stage-1 ECM computation will involve an impossible division modulo n. This occurs, in particular, whenever s is a multiple of the group size $\#E(\mathbb{Z}/q\mathbb{Z})$. As E varies randomly, $\#E(\mathbb{Z}/q\mathbb{Z})$ varies randomly (with some subtleties in its distribution; see, e.g., [McK99]) in the Hasse interval $[q - 2\sqrt{q} + 1, q + 2\sqrt{q} + 1]$. What makes ECM useful is that a surprisingly small s, allowing a surprisingly fast computation of $[s]P$, is a multiple of a surprisingly large percentage of the integers in the Hasse interval, and is a multiple of the order of P modulo q with (conjecturally) an even larger probability. See Section 3.5 for detailed statistics.

Example 3.4. For example, one could try to factor n as follows. Choose the curve $E : y^2 = x^3 - 2$, the Weierstrass x-coordinate as ϕ, the point $(x, y) = (3, 5)$, and the integer $s = 420$. Choose the following strategy to compute the x-coordinate of

$[420](3,5)$: use the standard affine-coordinate doubling formulas to compute $[2](3,5)$, then $[4](3,5)$, then $[8](3,5)$; use the standard affine-coordinate addition formulas to compute $[12](3,5)$; continue similarly through $[2](3,5)$, $[4](3,5)$, $[8](3,5)$, $[12](3,5)$, $[24](3,5)$, $[48](3,5)$, $[96](3,5)$, $[192](3,5)$, $[384](3,5)$, $[408](3,5)$, $[420](3,5)$. Carry out these computations modulo n, hoping for a division by a nonzero nonunit modulo n.

The denominator of the x-coordinate of $[420](3,5)$ in $E(\mathbb{Q})$ has many small prime factors: 2, 3, 5, 7, 11, 19, 29, 31, 41, 43, 59, 67, 71, 83, 89, 109, 163, 179, 181, 211, 223, 241, 269, 283, 383, 409, 419, 433, 523, 739, 769, 811, 839, etc. If n shares any of these prime factors then the computation of $[420](3,5)$ will encounter an impossible division modulo n. To verify the presence of (e.g.) the primes 769, 811, and 839 one can observe that $[420](3,5)$ is the neutral element in each of the groups $E(\mathbb{Z}/769\mathbb{Z})$, $E(\mathbb{Z}/811\mathbb{Z})$, $E(\mathbb{Z}/839\mathbb{Z})$; the order of $(3,5)$ turns out to be 7, 42, 35 respectively. Note that the group orders are 819, 756, and 840, none of which divide 420.

Remark 3.5 (The standard choice of s)**.** Pollard in [Pol74, page 527] suggested choosing s as "the product of all the primes $p_i \leq L$ each to some power $c_i \geq 1$. There is some freedom in the choice of the c_i but the smallest primes should certainly occur to some power higher than the first."

Pollard's prime bound "L" is now called B_1. One possibility is to choose, for each prime $\pi \leq B_1$, the largest power of π in the interval $[1, n + 2\sqrt{n} + 1]$. Then $[s]P$ is the neutral element in $E(\mathbb{Z}/q\mathbb{Z})$ if and only if the order of P is "B_1-smooth". This possibility is theoretically pleasing but clearly suboptimal.

Brent in [Bre86, Section 5] said that "in practice we choose" the largest power of π in the interval $[1, B_1]$ "because this significantly reduces the cost of a trial without significantly reducing the probability of success." GMP-ECM uses the same strategy; see [ZD06, page 529].

Remark 3.6 (The standard prime-by-prime strategy)**.** Pollard in [Pol74, page 527] said that one "can choose between using the primes p_i in succession or computing P in advance and performing a single power operation." Pollard's "P" is s in the notation of this thesis.

As far as we know, all ECM implementations use the first strategy, working with one prime at a time. Brent in [Bre86, Section 5] wrote "Actually, E [i.e., s in our notation] is not computed. Instead ... repeated operations of the form $P := P^k$ [i.e., $[k]P$ in our notation], where k ... is a prime power." Montgomery in [Mon87, page 249] wrote "It is unnecessary to compute R [i.e., s in our notation] explicitly." Zimmermann and Dodson in [ZD06, page 529] wrote "That big product is not computed as such" and presented the prime-by-prime loop used in GMP-ECM.

Remark 3.7 (The standard elliptic-curve coordinate system)**.** Chudnovsky and Chudnovsky in [CC86, Section 4] wrote "The crucial problem becomes the choice of the model of an algebraic group variety, where computations mod p are the least time consuming." They presented explicit formulas for computations on several different shapes of elliptic curves.

Section 1.2.6 introduced Montgomery curves, i.e., elliptic curves of the form $By^2 = x^3+Ax^2+x$ which Montgomery introduced particularly for speeding up ECM implementations. Montgomery suggested what are now called "Montgomery coordinates": a point (x_1, y_1) on $By^2 = x^3 + Ax^2 + x$ is represented as a pair $(X_1 : Z_1)$ such that $X_1/Z_1 = x_1$. This representation does not distinguish (x_1, y_1) from $(x_1, -y_1)$, so it does not allow addition, but it does allow "differential addition," i.e., computation of $P + Q$ given P, Q, and $P - Q$. In particular, Montgomery presented explicit formulas to compute $P, [2k]P, [(2k+1)]P$ from $P, [k]P, [k+1]P$ using $6\mathbf{M}+4\mathbf{S}+1\mathbf{D}$, or $5\mathbf{M}+4\mathbf{S}+1\mathbf{D}$ if P is given with $Z_1 = 1$, or $4\mathbf{M}+4\mathbf{S}+1\mathbf{D}$ if P is a very small point such as $(X_1 : Z_1) = (3, 5)$. One can find earlier formulas for the same computation in [CC86, formula (4.19)], but Montgomery's formulas are faster.
As far as we know, all subsequent ECM implementations have used Montgomery coordinates. In particular, GMP-ECM uses Montgomery coordinates for stage 1, with "PRAC," a particular differential addition chain introduced by Montgomery in [Mon83].

Remark 3.8 (Note on stage 2)**.** There is an analogue for ECM of the second stage in Pollard's $(p - 1)$-algorithm. Stage 2 hopes for n to have a prime divisor q such that $[s]P$ has small prime order in $E(\mathbb{Z}/q\mathbb{Z})$: specifically, order ℓ for some prime ℓ between B_1 and B_2. Here B_1 is the stage-1 parameter and B_2 is a new stage-2 parameter. The most obvious way to check for a small order of $[s]P$ is a prime-by-prime approach, computing $[\ell s]P$ modulo n for each prime ℓ.
If ℓ' is the next prime after ℓ then one can move from $[\ell s]P$ to $[\ell' s]P$ by adding a precomputed point $[(\ell'-\ell)s]P$. Computing all $[\ell s]P$ in this way takes about $B_2/\log B_2 - B_1/\log B_1$ elliptic-curve additions modulo n: there are about $B_2/\log B_2 - B_1/\log B_1$ primes ℓ, and the time for precomputation is quite small, since the differences $\ell' - \ell$ are generally quite small.
Section 5 in [BBLP08] discusses standard speedups such as the baby-step-giant-step approach, fast polynomial arithmetic, higher-degree baby steps and giant steps and it also discusses how the use of Edwards curves in extended coordinates speeds up the computations of stage 2.

3.1.3 Speedups in EECM-MPFQ

EECM-MPFQ breaks with stage-1 tradition as described in the previous section in three ways:

- EECM-MPFQ uses twisted Edwards curves $ax^2+y^2 = 1+dx^2y^2$ with extended Edwards coordinates with $\phi = 1/x$ whereas GMP-ECM uses Montgomery curves with Montgomery coordinates. See below for performance results.

- EECM-MPFQ handles the prime factors π of s in a batch, whereas GMP-ECM handles each prime factor separately. EECM-MPFQ always uses a single batch: it computes the entire product s and then replaces P with $[s]P$. The large batches save time, as discussed below; the computation of s takes negligible time.

- EECM-MPFQ uses "signed sliding fractional window" addition-subtraction chains which were defined in Section 2.2.3; see also [Doc05] and [BL08]. These chains compute $P \mapsto [s]P$ using only 1 doubling and ϵ additions for each bit of s. Here ϵ converges to 0 as s increases in length; this is why larger batches save time. The savings are amplified by the fact that an addition is somewhat more expensive than a doubling. Note that these chains are not compatible with Montgomery coordinates; they are shorter than any differential addition chain can be.

EECM-MPFQ follows tradition in its choice of s. Our experiments have not found significant speedups from other choices of s: for example, allowing prime powers in the larger interval $[1, B_1^{1.5}]$ has negligible extra cost when B_1 is large, but it also appears to have negligible benefit.
The addition-subtraction chains used in EECM-MPFQ are the chains $C_m(s)$ defined in [BL08, Section 3]:
Given B_1, EECM-MPFQ computes s, computes $C_m(s)$ for various choices of the chain parameter m, and keeps the lowest-cost chain that it finds in a simple measure of cost. (Variations in the cost measure do not lead to noticeably better chains.) The total time spent on this computation is small: for example, under a second for $B_1 = 1048576$. The resulting chain is reused for many curves and many inputs n.
For a description of how CPU cycles are measured for GMP-ECM and EECM-MPFQ we refer to Section 4 in [BBLP08].
This thesis restricts to discussing improvements of stage 1 of ECM. Note that EECM-MPFQ contains both stage 1 and stage 2.

3.2 Edwards curves with large torsion

This section explains which curves are used in EECM-MPFQ. Curves having 12 or 16 torsion points over $\mathbb{Q}$ are guaranteed to have 12 or 16 as divisors of their group orders modulo primes (of good reduction), improving the smoothness chance of the group orders and thus improving the success chance of ECM. We show how to use analogous improvements for Edwards curves. Note that [BBLP08, Section 9] discusses the impact of large torsion in the case of EECM in detail.
The proofs given in this section make use of the characterization of points of small order on (twisted) Edwards curves in Section 1.2.5.

3.2.1 Elliptic curves over the rationals

The rational points of finite order on an elliptic curve E over a field k form a subgroup of $E(k)$ which is called the *torsion group* and denoted by $E_{\mathrm{tor}}(k)$. Mordell proved that the rational points $E(\mathbb{Q})$ form a finitely generated group; specifically that $E(\mathbb{Q}) \cong E_{\mathrm{tor}}(\mathbb{Q}) \times \mathbb{Z}^r$. Here r is a nonnegative integer called the *rank* of E. The proof of Mordell's theorem goes beyond the scope of this thesis; see Chapter VIII in [Sil86] for a proof of the more general *Mordell–Weil theorem*.

The Elliptic-Curve Method starts by selecting an elliptic curve E over the rationals. One hopes that reducing E modulo a possible divisor q of n results in $\#E(\mathbb{Z}/q\mathbb{Z})$ being smooth. A common way to influence the smoothness is to construct curves with prescribed torsion and positive rank having powers of 2 and 3 dividing the group size $\#E(\mathbb{Q})$.
Mazur characterized the torsion group of elliptic curves over the rationals. This thesis states Mazur's theorem as in [Sil86] where the proof is omitted due to its complexity. The theorem in more generality and including proofs can be found in [Maz77] and [Maz78].

Theorem 3.9. *The torsion group $E_{tor}(\mathbb{Q})$ of any elliptic curve E is isomorphic to one of 15 finite groups, specifically*

$$E_{tor}(\mathbb{Q}) \cong \begin{cases} \mathbb{Z}/m\mathbb{Z}, & m \in \{1,2,3,4,5,6,7,8,9,10,12\}, \quad \text{or} \\ \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2m\mathbb{Z}, & m \in \{1,2,3,4\}. \end{cases}$$

Since any elliptic curve in Edwards form has a point of order 4 it follows that the torsion group of an Edwards curve is isomorphic to either $\mathbb{Z}/4\mathbb{Z}$, $\mathbb{Z}/8\mathbb{Z}$, $\mathbb{Z}/12\mathbb{Z}$, $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$, or $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$.
The most interesting cases for ECM are $\mathbb{Z}/12\mathbb{Z}$ and $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$, since they force the group orders of E modulo primes p (of good reduction) to be divisible by 12 and 16 respectively. This section shows which conditions an Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{Q}$ must satisfy to have torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$ or $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$. We give parametrizations for both cases.
Computations in extended Edwards coordinates would benefit from using twisted Edwards curves with $a = -1$. We show that such curves cannot have $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$ or $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$.
We first present the constructions and then show the impossibility results.

3.2.2 Torsion group $\mathbb{Z}/12\mathbb{Z}$

Theorem 3.10 states a genus-0 cover of the set of Edwards curves over $\mathbb{Q}$ with torsion group $\mathbb{Z}/12\mathbb{Z}$. Theorem 3.11 identifies all the points of finite order on such curves. Theorem 3.12 states a rational cover.

Theorem 3.10. *If $y_3 \in \mathbb{Q} \setminus \{-2, -1/2, 0, \pm 1\}$ and $x_3 \in \mathbb{Q}$ satisfy $x_3^2 = -(y_3^2 + 2y_3)$ then the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{Q}$, where $d = -(2y_3 + 1)/(x_3^2y_3^2)$, has (x_3, y_3) as a point of order 3 and has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$. Conversely, every Edwards curve over $\mathbb{Q}$ with a point of order 3 arises in this way.*

Proof. Assume that such a y_3 and x_3 exist. Then d is defined and not equal to 0 or 1, and $x_3^2 + y_3^2 = -2y_3 = 1 + dx_3^2y_3^2$. By Theorem 1.32, (x_3, y_3) is a point of order 3 on $\overline{E}_{\mathrm{E},1,d}(\mathbb{Q})$. Since each Edwards curve has a point of order 4 the torsion group must contain a copy of $\mathbb{Z}/12\mathbb{Z}$. By Mazur's theorem the torsion group cannot be larger.

Conversely, if $\overline{E}_{\mathrm{E},1,d}(\mathbb{Q})$ has a point of order 3, then by Theorem 1.32 the point can be written as (x_3, y_3) where $x_3^2 + y_3^2 = 1 + dx_3^2y_3^2 = -2y_3$. Hence $x_3^2 = -(y_3^2 + 2y_3)$. Note that $x_3 \neq 0$, since otherwise $y_3^2 = 1 = -2y_3$; and note that $y_3 \notin \{0, -2\}$ since otherwise $x_3 = 0$. Now $d = -(2y_3 + 1)/(x_3^2y_3^2)$. Finally note that $y_3 \notin \{-1/2, \pm 1\}$ since otherwise $d \in \{0, 1\}$, contradicting the definition of an Edwards curve.

□

Theorem 3.11. *Let $x^2+y^2 = 1+dx^2y^2$ be an Edwards curve over $\mathbb{Q}$ with $E_{\mathrm{tor}}(\mathbb{Q}) \cong \mathbb{Z}/12\mathbb{Z}$ and let $P_3 = (x_3, y_3)$ be a point of order 3 on the curve.*
The 12 torsion points on the curve and their respective orders are as follows:

point	$(0,1)$	$(0,-1)$	$(\pm x_3, y_3)$	$(\pm 1, 0)$	$(\pm x_3, -y_3)$	$(\pm y_3, \pm x_3)$
order	1	2	3	4	6	12

Proof. The points of order 6 are obtained as $(\pm x_3, y_3) + (0, -1)$, the points of order 12 by adding $(\pm 1, 0)$ to the points of order 3. □

Theorem 3.12. *If $u \in \mathbb{Q} \setminus \{0, \pm 1\}$ then the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{Q}$, where*

$$x_3 = \frac{u^2 - 1}{u^2 + 1}, \quad y_3 = -\frac{(u-1)^2}{u^2+1}, \quad d = \frac{(u^2+1)^3(u^2-4u+1)}{(u-1)^6(u+1)^2}$$

has (x_3, y_3) as a point of order 3 and has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$. Conversely, every Edwards curve over $\mathbb{Q}$ with a point of order 3 arises in this way. The parameters u and $1/u$ give the same value of d.

Proof. Multiply the identity $(u+1)^2 + (u-1)^2 = 2(u^2+1)$ by $(u-1)^2/(u^2+1)^2$ to see that $x_3^2 + y_3^2 = -2y_3$, and observe that

$$d = \frac{2(u-1)^2 - (u^2+1)}{u^2+1} \cdot \frac{(u^2+1)^2}{(u^2-1)^2} \cdot \frac{(u^2+1)^2}{(u-1)^4} = \frac{-2y_3 - 1}{x_3^2y_3^2}.$$

Furthermore $y_3 \notin \{-2, -1/2, 0, \pm 1\}$ since $u \in \mathbb{Q} \setminus \{0, \pm 1\}$. By Theorem 3.10, the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{Q}$ has (x_3, y_3) as a point of order 3 and has torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$.
Conversely, assume that the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ has a point of order 3. By Theorem 3.10, the curve has a point (x_3, y_3) of order 3 for some $y_3 \in \mathbb{Q} \setminus \{-2, -1/2, 0, \pm 1\}$ and $x_3 \in \mathbb{Q}$ satisfying $x_3^2 = -(y_3^2+2y_3)$ and $d = -(2y_3+1)/(x_3^2y_3^2)$. Note that $(x_3, y_3 + 1)$ is a point on the unit circle.
If $x_3 = \pm 1$ then $y_3 + 1 = 0$ so $d = -(2y_3 + 1)/(x_3^2y_3^2) = 1$; but Edwards curves have $d \neq 1$. Hence $x_3 \neq \pm 1$. Furthermore $x_3 \neq 0$ since every point with x-coordinate 0 has order 1 or 2.
Define u as the slope of the line between $(1, 0)$ and $(x_3, -(y_3 + 1))$; i.e., $u = (y_3 + 1)/(1-x_3)$. Substitute $y_3+1 = u(1-x_3)$ into $(y_3+1)^2 = 1-x_3^2$ to obtain $u^2(1-x_3)^2 = 1 - x_3^2 = (1+x_3)(1-x_3)$, i.e., $u^2(1-x_3) = 1+x_3$, i.e., $x_3 = (u^2-1)/(u^2+1)$. Then $u \notin \{0, \pm 1\}$ since $x_3 \notin \{0, -1\}$. Furthermore $y_3 = u(1-x_3)-1 = u(2/(u^2+1))-1 =$

$-(u-1)^2/(u^2+1)$ and as above $d = (2y_3+1)/(x_3^2y_3^2) = (u^2+1)^3(u^2-4u+1)/((u-1)^6(u+1)^2)$.
The value of d is invariant under the change $u \mapsto 1/u$ since

$$\frac{(1+u^2)^3(1-4u+u^2)}{(1-u)^6(1+u)^2} = \frac{(u^2+1)^3(u^2-4u+1)}{(u-1)^6(u+1)^2}.$$

□

Solving the equation $d(u') = d(u)$ for u' in terms of u over the rationals shows that $u \mapsto 1/u$ is the only rational transformation leaving d invariant that works independently of u.

3.2.3 Torsion group $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$

Theorem 3.13 states a genus-0 cover of the set of Edwards curves over $\mathbb{Q}$ with torsion group $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$. Theorem 3.14 identifies all the affine points of finite order on such curves. Theorem 3.15 states a rational cover and identifies the degree of the cover.
There are actually two types of curves in Theorem 3.13: points of order 8 double to $(\pm 1 : 0)$ on curves of the first type, or to $((1 : \pm\sqrt{d}), (1 : 0))$ on curves of the second type. Curves of the second type are birationally equivalent to curves of the first type by Remark 1.17. Subsequent theorems consider only the first type.

Theorem 3.13. *If $x_8 \in \mathbb{Q} \setminus \{0, \pm 1\}$ and $d = (2x_8^2 - 1)/x_8^4$ is a square in $\mathbb{Q}$ then the Edwards curve $x^2+y^2 = 1+dx^2y^2$ over $\mathbb{Q}$ has $(x_8, \pm x_8)$ as points of order 8 doubling to $(\pm 1, 0)$, and has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$. Conversely, every Edwards curve over $\mathbb{Q}$ with $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ and a point of order 8 doubling to $(\pm 1, 0)$ arises in this way.*
If $\bar{x}_8 \in \mathbb{Q} \setminus \{0, \pm 1\}$ and $d = 1/(\bar{x}_8^2(2 - \bar{x}_8^2))$ is a square in $\mathbb{Q}$ then the Edwards curve $x^2+y^2 = 1+dx^2y^2$ over $\mathbb{Q}$ has $(\bar{x}_8, \pm 1/(\bar{x}_8\sqrt{d}))$ as points of order 8 doubling to $((1 : \pm\sqrt{d}), (1 : 0))$, and has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$. Conversely, every Edwards curve over $\mathbb{Q}$ with $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ and a point of order 8 doubling to $((1 : \pm\sqrt{d}), (1 : 0))$ arises in this way.
Every Edwards curve over $\mathbb{Q}$ with $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ arises in one of these two ways.

Proof. Any such x_8 yields $d \neq 0, 1$, so $x^2 + y^2 = 1 + dx^2y^2$ is an Edwards curve. By Theorems 1.30 and 1.31, the curve has points $(0, -1)$ and $((1 : 0), (1 : \pm\sqrt{d}))$ of order 2, and points $(x_8, \pm x_8)$ of order 8 doubling to $(\pm 1, 0)$. Similarly, any such $\bar{x}_8$ yields an Edwards curve with points $(0, -1)$ and $((1 : 0), (1 : \pm\sqrt{d}))$ of order 2 and $(\bar{x}_8, \pm 1/(\bar{x}_8\sqrt{d}))$ of order 8 doubling to $((1 : \pm\sqrt{d}), (1 : 0))$.
In both cases the torsion group contains a copy of $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$. By Mazur's theorem the torsion group cannot be larger.
Conversely, assume that $x^2+y^2 = 1+dx^2y^2$ is an Edwards curve with $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$. There are four elements of order 4 in $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$,

all doubling to the same element, so there are four order-4 points on the curve, all doubling to the same point.
The points $(\pm 1, 0)$ have order 4 and double to $(0, -1)$, so the other two points of order 4 also double to $(0, -1)$. By Theorem 1.30, those other two points must be $((1 : \pm\sqrt{d}), (1 : 0))$, and d must be a square.
Now any point of order 8 must double to $(\pm 1, 0)$ or to $((1 : \pm\sqrt{d}), (1 : 0))$. In the first case, by Theorem 1.31, the point is $(x_8, \pm x_8)$ for some root x_8 of $dx_8^4 - 2x_8^2 + 1$; hence $x_8 \notin \{0, \pm 1\}$ and $d = (2x_8^2 - 1)/x_8^4$. In the second case, by Theorem 1.31, the point is $(\bar{x}_8, \pm 1/(\bar{x}_8\sqrt{d}))$ for some root $\bar{x}_8$ of $d\bar{x}_8^4 - 2d\bar{x}_8^2 + 1$; hence $\bar{x}_8 \notin \{0, \pm 1\}$ and $d = 1/(\bar{x}_8^4 - 2\bar{x}_8^2)$. □

Recall Remark 1.17, namely that if d is a square, then the Edwards curves $x^2 + y^2 = 1 + dx^2y^2$ and $\bar{x}^2 + \bar{y}^2 = 1 + (1/d)\bar{x}^2\bar{y}^2$ are birationally equivalent via the map $\bar{x} = x\sqrt{d}, \bar{y} = 1/y$ with inverse $x = \bar{x}/\sqrt{d}, y = 1/\bar{y}$. The map fixes $(0, \pm 1)$. In particular, each curve of the second type in Theorem 3.13 is birationally equivalent to a curve of the first type. Indeed, assume that $\bar{x}_8 \in \mathbb{Q} \setminus \{0, \pm 1\}$ and that $d = 1/(\bar{x}_8^2(2 - \bar{x}_8^2))$ is a square in $\mathbb{Q}$. Define $x_8 = \bar{x}_8\sqrt{d}$. Then $x_8^2 = 1/(2 - \bar{x}_8^2)$, so $(2x_8^2 - 1)/x_8^4 = (2/(2 - \bar{x}_8^2) - 1)(2 - \bar{x}_8^2)^2 = \bar{x}_8^2(2 - \bar{x}_8^2) = 1/d$, which is a square; furthermore, $x_8 \notin \{0, \pm 1\}$ since $2 - \bar{x}_8^2 \neq 1$ since $\bar{x}_8 \notin \{\pm 1\}$. Hence $x^2 + y^2 = 1 + (1/d)x^2y^2$ is a curve of the first type. The curve $x^2 + y^2 = 1 + dx^2y^2$ is birationally equivalent to $\bar{x}^2 + \bar{y}^2 = 1 + (1/d)\bar{x}^2\bar{y}^2$ by Remark 1.17. Consequently, we can restrict our attention to curves of the first type, i.e., curves on which the points of order 8 double to $(\pm 1, 0)$.

Theorem 3.14. *Assume that $x_8 \in \mathbb{Q} \setminus \{0, \pm 1\}$ and that $d = (2x_8^2 - 1)/x_8^4$ is a square in $\mathbb{Q}$. Then there are 16 points of finite order on $\overline{E}_{\mathrm{E},1,d}$ over $\mathbb{Q}$. The affine points of finite order are as follows:*

point	$(0, 1)$	$(0, -1)$	$(\pm 1, 0)$	$(\pm x_8, \pm x_8)$	$\left(\pm 1/(x_8\sqrt{d}), \pm 1/(x_8\sqrt{d})\right)$
order	1	2	4	8	8

where the signs are taken independently.

Proof. Theorem 1.30 (with $a = 1$) shows that the 4 affine points $(0, 1)$, $(0, -1)$, and $(\pm 1, 0)$ are on $\overline{E}_{\mathrm{E},1,d}$ and have the stated orders. It also shows that the 2 non-affine points $((1 : 0), (1 : \pm\sqrt{d}))$ have order 2 and that the 2 non-affine points $((1 : \pm\sqrt{d}), (1 : 0))$ have order 4. Theorem 1.31 shows that the other affine points listed are 8 distinct points on $\overline{E}_{\mathrm{E},1,d}$ and have order 8. The torsion group has exactly 16 elements by Theorem 3.13. □

Theorem 3.15. *If $u \in \mathbb{Q} \setminus \{0, -1, -2\}$ then the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{Q}$, where*

$$x_8 = \frac{u^2 + 2u + 2}{u^2 - 2}, \qquad d = \frac{2x_8^2 - 1}{x_8^4},$$

has (x_8, x_8) as a point of order 8 and has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$.

Conversely, every Edwards curve over $\mathbb{Q}$ with torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ on which the points of order 8 double to $(\pm 1, 0)$ is expressible in this way. The parameters u, $2/u$, $-2(u+1)/(u+2)$, $-(2+u)/(1+u)$, $-(u+2)$, $-2/(u+2)$, $-u/(u+1)$, and $-2(u+1)/u$ give the same value of d and they are the only values giving this d.

Proof. Divide the identity $2(u^2+2u+2)^2-(u^2-2)^2 = (u^2+4u+2)^2$ by $(u^2-2)^2$ to see that $2x_8^2-1 = (u^2+4u+2)^2/(u^2-2)^2$. Hence d is a square. Furthermore $x_8 \neq 0$ since $u^2+2u+2 \neq 0$; $x_8 \neq 1$ since $u \neq -2$; and $x_8 \neq -1$ since $u \notin \{0,-1\}$. By Theorem 3.13, the curve $\overline{E}_{\mathrm{E},1,d}$ has (x_8, x_8) as a point of order 8, and has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$.
Conversely, assume that an Edwards curve has torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ and has a point of order 8 doubling to $(\pm 1, 0)$. By Theorem 3.13, the curve can be expressed as $\overline{E}_{\mathrm{E},1,d}$ for some $x_8 \in \mathbb{Q} \setminus \{0, \pm 1\}$ such that $d = (2x_8^2-1)/x_8^4$ is a square in $\mathbb{Q}$; i.e., such that $2x_8^2-1$ is a square in $\mathbb{Q}$.
Choose $r \in \mathbb{Q}$ such that $2x_8^2 - 1 = r^2$. Define u as the slope of the line between $(1,-1)$ and (x_8, r): i.e., $u = (r+1)/(x_8-1)$. Substitute $r = u(x_8-1)-1$ into $2(x_8^2-1) = (r+1)(r-1)$ to obtain $2(x_8^2-1) = u(x_8-1)(u(x_8-1)-2)$, i.e., $2(x_8+1) = u(u(x_8-1)-2)$, i.e., $2x_8+2 = u^2x_8 - u^2 - 2u$, i.e., $x_8 = (u^2+2u+2)/(u^2-2)$. Finally $u \notin \{0,-1\}$ since $x_8 \neq -1$, and $u \neq -2$ since $x_8 \neq 1$.
The identity

$$\begin{aligned}(d(u) - d(v))((u+1)^2+1)^4((v+1)^2+1)^4 \\ = 16(u-v)(uv-2)((u+2)v+2(u+1))(u+2+(u+1)v) \\ \cdot (u+v+2)((u+2)v+2)(u+(u+1)v)(uv+2(u+1))\end{aligned}$$

immediately shows that if v is any of the values $u, 2/u, \ldots$ listed in the theorem then $d(v) = d(u)$. Conversely, if v is not one of those values then none of the factors $u-v, uv-2, \ldots$ are 0 so $d(v) \neq d(u)$.

□

3.2.4 Impossibility results

The following theorem shows that the only way for a twisted Edwards curve to have exactly 12 torsion points is to have torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$. The next two theorems consider twisted Edwards curves with $a = -1$ and show that these cannot have $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$ or $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$. The last theorem shows that a twisted Edwards curve cannot have exactly 10 torsion points.

Theorem 3.16. *There exists no twisted Edwards curve over $\mathbb{Q}$ with torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/6\mathbb{Z}$.*

Proof. Let a, d be distinct nonzero elements of $\mathbb{Q}$. Suppose that the twisted Edwards curve $\overline{E}_{\mathrm{E},a,d} : ax^2+y^2 = 1+dx^2y^2$ has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/6\mathbb{Z}$.

There are three elements of order 2 in $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/6\mathbb{Z}$, so there are three points of order 2 in $\overline{E}_{\mathrm{E},a,d}(\mathbb{Q})$. By Theorem 1.30 the only possible points of order 2 are $(0,-1)$ and $((1:0),(\pm\sqrt{a/d}:1))$. Hence $\sqrt{a/d} \in \mathbb{Q}$.
There are also elements of order 3 in $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/6\mathbb{Z}$. Choose a point of order 3 in $\overline{E}_{\mathrm{E},a,d}(\mathbb{Q})$. By Theorem 1.32 this point can be expressed as (x_3, y_3) where $ax_3^2+y_3^2 = 1+dx_3^2y_3^2 = -2y_3$.
Write $u = 1+y_3$. Then $1-u^2 = -2y_3-y_3^2 = ax_3^2$. Starting from $dx_3^2y_3^2 = ax_3^2+y_3^2-1$, replace x_3^2 by $(1-u^2)/a$ and replace y_3 by $u-1$ to see that $(d/a)(1-u^2)(u-1)^2 = (1-u^2)+(u-1)^2-1 = 1-2u$. Hence $s^2 = 4(1-2u)(1-u^2)$ where $s = 2(1-u^2)(u-1)\sqrt{d/a} \in \mathbb{Q}$.
In other words, $(2u, s)$ is a $\mathbb{Q}$-rational point (σ, τ) on the elliptic curve $\tau^2 = \sigma^3 - \sigma^2 - 4\sigma + 4$. This elliptic curve has rank 0 over $\mathbb{Q}$, and has exactly 7 affine points over $\mathbb{Q}$, as one can verify by typing

```
E=EllipticCurve(QQ,[0,-1,0,-4,4])
print E.rank()
print E.torsion_points()
```

into the Sage computer-algebra system [S+10]. Specifically, (σ, τ) must be one of $(\pm 2, 0), (0, \pm 2), (1, 0), (4, \pm 6)$. Hence $u \in \{\pm 1, 0, 1/2, 2\}$. In each case $(a : d) = ((1-u^2)(u-1)^2 : 1-2u) \in \{(1:1), (0:1), (1:0)\}$, contradicting the assumption that a, d are distinct nonzero elements of $\mathbb{Q}$. □

Theorem 3.17. *There exists no twisted Edwards curve of the form $ax^2+y^2 = 1+dx^2y^2$ over $\mathbb{Q}$ with $a = -1$ and torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$.*

Proof. Suppose that the twisted Edwards curve $\overline{E}_{\mathrm{E},-1,d} : -x^2+y^2 = 1+dx^2y^2$ has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$.
There is a unique element of order 2 in $\mathbb{Z}/12\mathbb{Z}$, so $(0,-1)$ is the only point of order 2 on $\overline{E}_{\mathrm{E},-1,d}(\mathbb{Q})$. Furthermore, there are elements of order 4 in $\mathbb{Z}/12\mathbb{Z}$, so there are points on $\overline{E}_{\mathrm{E},-1,d}(\mathbb{Q})$ doubling to $(0,-1)$. By Theorem 1.30 the only possibilities for such points are $((1:\pm\sqrt{a}),(0:1))$ or $((1:\pm\sqrt{d}),(1:0))$. Hence a or d is a square in $\mathbb{Q}$; but $a = -1$ is not a square in $\mathbb{Q}$, so d is a square in $\mathbb{Q}$.
There are also elements of order 3 in $\mathbb{Z}/12\mathbb{Z}$. As in the proof of Theorem 3.16 there exists $u \in \mathbb{Q}$ such that $(d/a)(1-u^2)(u-1)^2 = 1-2u$. Here $a = -1$ so $s^2 = -4(1-u^2)(1-2u)$ where $s = 2(1-u^2)(u-1)\sqrt{d} \in \mathbb{Q}$.
In other words, $(-2u, s)$ is a $\mathbb{Q}$-rational point on the elliptic curve $\tau^2 = \sigma^3+\sigma^2-4\sigma-4$. This elliptic curve has rank 0 over $\mathbb{Q}$, and has exactly 3 affine points over $\mathbb{Q}$: specifically, (σ, τ) must be one of $(\pm 2, 0), (-1, 0)$. Hence $u \in \{\pm 1, 1/2\}$. If $u \in \{\pm 1\}$ then $0 = (d/a)(1-u^2)(u-1)^2 = 1-2u \neq 0$, contradiction; if $u = 1/2$ then $0 = 1-2u = (d/a)(1-u^2)(u-1)^2 \neq 0$, contradiction. □

Theorem 3.18. *There exists no twisted Edwards curve of the form $ax^2+y^2 = 1+dx^2y^2$ over $\mathbb{Q}$ with $a = -1$ and torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$.*

Proof. Suppose that the twisted Edwards curve $\overline{E}_{\mathrm{E},-1,d} : -x^2+y^2 = 1+dx^2y^2$ has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$.

The torsion group contains exactly three elements of order 2, so $\sqrt{a/d} \in \mathbb{Q}$ as in the proof of Theorem 3.16; i.e., $\sqrt{-d} \in \mathbb{Q}$. Consequently d is not a square in $\mathbb{Q}$.
The torsion group also contains exactly 4 elements of order 4. These elements cannot double to $(0,-1)$: otherwise they would have the form $((1:\pm\sqrt{-1}),(0:1))$ or $((1:\pm\sqrt{d}),(1:0))$ by Theorem 1.30, but neither -1 nor d is a square in $\mathbb{Q}$. The elements of order 4 therefore double to $((1:0),(\pm\sqrt{-1/d}:1))$.
If $s^2 = -1/d$ then the elements of order 4 doubling to $((1:0),(s:1))$ are $(\pm\sqrt{s},\pm\sqrt{s})$ by Theorem 1.30, where the $\pm$ signs are assumed independently. In particular, if such elements are defined over $\mathbb{Q}$, then $\pm\sqrt{s} \in \mathbb{Q}$, so s is a square in $\mathbb{Q}$, so $-1/d$ is a fourth power in $\mathbb{Q}$, say f^4. Now $(\pm f, \pm f)$ are points of order 4 doubling to $((1:0),(f^2:1))$, and there are no other points of order 4.
The torsion group contains a point P_8 of order 8. This point doubles to $(\pm f, \pm f)$. Assume without loss of generality that $[2]P_8 = (\pm f, f)$: otherwise replace f by $-f$. Further assume without loss of generality that $[2]P_8 = (f, f)$: otherwise replace P_8 by $-P_8$. Any point having a zero coordinate has order at most 4, so P_8 must be an affine point, say (x_8, y_8), with $x_8 \neq 0$ and $y_8 \neq 0$.
Now $[2]P_8 = (f,f)$ implies $(2x_8y_8)/(-x_8^2+y_8^2) = f = (y_8^2+x_8^2)/(2+x_8^2-y_8^2)$, with $-x_8^2+y_8^2 \neq 0$ and $2+x_8^2-y_8^2 \neq 0$. In particular, $(y_8^2+x_8^2)(-x_8^2+y_8^2) = (2x_8y_8)(2+x_8^2-y_8^2)$, so $(y_8^2-x_8^2)(x_8^2+y_8^2+2x_8y_8) = 4x_8y_8$; i.e., $(y_8^2-x_8^2)r^2 = 4x_8y_8$ where $r = x_8+y_8$.
Define $s = 2(y_8^2+x_8^2)/(y_8^2-x_8^2)$. Then

$$s^2 - 4 = \frac{4((y_8^2+x_8^2)^2-(y_8^2-x_8^2)^2)}{(y_8^2-x_8^2)^2} = \frac{16y_8^2x_8^2}{(y_8^2-x_8^2)^2} = r^4$$

so $(s+r^2)^2 - 4 = 2r^2(s+r^2)$; consequently $((s+r^2)/2, r(s+r^2)/2)$ is a rational point on the elliptic curve $\tau^2 = \sigma^3 - \sigma$. This curve has rank 0 over $\mathbb{Q}$ and exactly 3 affine points over $\mathbb{Q}$, namely $(\pm 1, 0)$ and $(0,0)$. Hence $r(s+r^2) = 0$; consequently $0 = r(s+r^2)(s-r^2) = r(s^2-r^4) = 4r$, so $r = 0$, so $x_8 + y_8 = 0$, contradicting $-x_8^2+y_8^2 \neq 0$. □

Theorem 3.19. *There exists no twisted Edwards curve over $\mathbb{Q}$ with torsion group isomorphic to $\mathbb{Z}/10\mathbb{Z}$.*

Proof. Suppose that the twisted Edwards curve $\overline{E}_{\mathrm{E},a,d} : ax^2+y^2 = 1+dx^2y^2$ has $\mathbb{Q}$-torsion group isomorphic to $\mathbb{Z}/10\mathbb{Z}$. This means in particular that there exists a point $P_5 \in \overline{E}_{\mathrm{E},a,d}(\mathbb{Q})$ of order 5. Points at infinity have order at most 4 by Theorem 1.30, so $P_5 = (x_5, y_5)$ for some $x_5, y_5 \in \mathbb{Q}$. Points with a zero coordinate also have order at most 4 by Theorem 1.30, so $x_5 \neq 0$ and $y_5 \neq 0$. Note also that $y_5 \notin \{-1, 1\}$ since $x_5 \neq 0$.
Apply the doubling formulas twice to see that the x-coordinate of $[4]P_5$ satisfies

$$x([4]P_5) - (-x_5) = \frac{x_5\,(ax_5^2+y_5^2-2y_5)\,F}{a^4x_5^8 + 4y_5^2a^3x_5^6 + (6y_5^4-16y_5^2)a^2x_5^4 + (4y_5^6-16y_5^4+16y_5^2)ax_5^2 + y_5^8},$$

where $F = a^3x_5^6 + (3y_5^2 + 6y_5)a^2x_5^4 + (3y_5^4 + 4y_5^3 - 4y_5^2 - 8y_5)ax_5^2 + y_5^6 - 2y_5^5 - 4y_5^4$. The equation $[4]P_5 = -P_5$ implies $x([4]P_5) - (-x_5) = 0$, so $x_5(ax_5^2 + y_5^2 - 2y_5)F = 0$.
Case 1: $ax_5^2 + y_5^2 = 2y_5$. Then $(x_5, -y_5)$ is a curve point of order 3 by Theorem 1.32, contradicting the hypothesis that the torsion group is isomorphic to $\mathbb{Z}/10\mathbb{Z}$.
Case 2: $F = 0$. Define $q = (ax_5^2 + y_5^2 + 2y_5)/y_5$ and $r = q/(y_5 + 1)$. The identity $rq^2 - (r^2 + 8)q + 16 = F/(y_5^2(y_5 + 1)^2)$ then implies $rq^2 - (r^2 + 8)q + 16 = 0$.
Define $U = q - r$, $V = q - r - 4$, and $W = 4 - q - r$. Then $(U, V, W) \neq (0, 0, 0)$, and $V^2W - U^3 - U^2W + UW^2 = 4(rq^2 - (r^2 + 8)q + 16) = 0$, so $(U : V : W)$ is a rational point on the elliptic curve $\tau^2 = \sigma^3 + \sigma^2 - \sigma$. This curve has rank 0 over $\mathbb{Q}$ and exactly 6 points over $\mathbb{Q}$, namely $(\pm 1, \pm 1)$, $(0, 0)$, and $(0 : 1 : 0)$, so $(U : V : W)$ is one of those points.
The points $(1, 1)$ and $(-1, -1)$ and $(0, 0)$ are excluded since $U \neq V$. The point $(1, -1)$ implies $(q, r) = (2, 0)$, contradicting $r = q/(y_5 + 1)$. The point $(-1, 1)$ implies $(q, r) = (4, 2)$, again contradicting $r = q/(y_5 + 1)$ since $y_5 \neq 1$. Finally, the point $(0 : 1 : 0)$ implies $(q, r) = (2, 2)$, again contradicting $r = q/(y_5 + 1)$ since $y_5 \neq 0$. □

3.3 Edwards curves with large torsion and positive rank

Atkin and Morain in [AM93] found an infinite family of elliptic curves over $\mathbb{Q}$ with torsion group $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ and with explicit non-torsion points. Montgomery in [Mon87, page 263] had earlier found an analogous family for $\mathbb{Z}/12\mathbb{Z}$. Suyama in [Suy85] had earlier given an infinite sequence of Montgomery curves with explicit non-torsion points and with group order divisible by 12 over any prime field. GMP-ECM uses Suyama curves; see [ZD06]. See [Mon92, Section 6] for further $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ constructions.
In this section we translate the Atkin–Morain and Montgomery constructions from Weierstrass curves to Edwards curves. We also translate the Suyama construction to twisted Edwards curves. This section relies on the equivalence between Montgomery curves and twisted Edwards curves which we discussed in Section 1.2.6.

3.3.1 The Atkin–Morain construction

The Atkin–Morain family is parametrized by points (s, t) on a particular elliptic curve $T^2 = S^3 - 8S - 32$. Atkin and Morain suggest computing multiples (s, t) of $(12, 40)$, a non-torsion point on this curve. Beware that these points have rapidly increasing height. Here the *height of a rational number* p/q is $\max\{|p|, |q|\}$ for coprime p and q; the *height of a point* on an elliptic curve represented in affine coordinates is defined as the maximum of the natural logarithm of the height of the x and y-coordinate of that point (see also [Sil86, page 202]).

Theorem 3.20 (Atkin, Morain)**.** *Let (s,t) be a rational point on the curve $T^2 = S^3 - 8S - 32$. Define $\alpha = ((t+25)/(s-9)+1)^{-1}$, $\beta = 2\alpha(4\alpha+1)/(8\alpha^2-1)$, $c = (2\beta-1)(\beta-1)/\beta$, and $b = \beta c$. Then the elliptic curve*

$$E_\alpha : V^2 = U^3 + \frac{((c-1)^2-4b)}{4}U^2 + \frac{b(c-1)}{2}U + \frac{b^2}{4}$$

has torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z}\times\mathbb{Z}/8\mathbb{Z}$ and a point with U-coordinate $-(2\beta-1)/4$.

Theorem 3.21. *Let (s,t) be a rational point on the curve $T^2 = S^3 - 8S - 32$. Define α and β as in Theorem 3.20. Define $d = (2(2\beta-1)^2-1)/(2\beta-1)^4$. Then the Edwards curve $x^2+y^2 = 1+dx^2y^2$ has torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z}\times\mathbb{Z}/8\mathbb{Z}$ and a point (x_1,y_1) with $x_1 = (2\beta-1)(4\beta-3)/(6\beta-5)$ and $y_1 = (2\beta-1)(t^2+50t-2s^3+27s^2-104)/((t+3s-2)(t+s+16))$.*

Proof. Put $x_8 = 2\beta-1$. By construction x_8 satisfies $(2x_8^2-1)/x_8^4 = d$. Furthermore

$$d = \frac{(8\alpha^2-1)^2(8\alpha^2+8\alpha+1)^2}{(8\alpha^2+4\alpha+1)^4},$$

so d is a square. By Theorem 3.13, the Edwards curve has torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z}\times\mathbb{Z}/8\mathbb{Z}$. Finally, a straightforward calculation shows that $x_1^2+y_1^2 = 1+dx_1^2y_1^2$. □

The point with U-coordinate $-(2\beta-1)/4$ in Theorem 3.20 is generically a non-torsion point. The V-coordinate of the point is not stated explicitly in [AM93]. The point (x_1,y_1) in Theorem 3.21 is the corresponding point on the Edwards curve.

3.3.2 The Suyama construction

The Suyama family has lower torsion but a simpler parametrization. We briefly review Suyama's family and present an analogous result for twisted Edwards curves.

Theorem 3.22 (Suyama)**.** *Let $\sigma > 5$ be a rational number. Define*

$$\alpha = \sigma^2-5, \quad \beta = 4\sigma, \quad W_1 = \beta^3, \quad A = \frac{(\beta-\alpha)^3(3\alpha+\beta)}{4\alpha^3\beta} - 2, \quad B = \frac{\alpha}{W_1}.$$

Then the $\mathbb{Q}$-torsion group of the elliptic curve $E_{\mathrm{M},A,B} : Bv^2 = u^3 + Au^2 + u$ has a subgroup isomorphic to $\mathbb{Z}/6\mathbb{Z}$.
Define $V_1 = (\sigma^2-1)(\sigma^2-25)(\sigma^4-25)$. Then $(u_1,v_1) = (\alpha^3/W_1, V_1/W_1)$ is a non-torsion point on $E_{\mathrm{M},A,B}$.

Theorem 3.23. *Let $\sigma > 5$ be a rational number. Define α,β,V_1 as in Theorem 3.22. Define $a = (\beta-\alpha)^3(3\alpha+\beta)$ and $d = (\beta+\alpha)^3(\beta-3\alpha)$. Then the $\mathbb{Q}$-torsion group of the twisted Edwards curve $ax^2+y^2 = 1+dx^2y^2$ has a subgroup isomorphic to $\mathbb{Z}/6\mathbb{Z}$, and $(x_1,y_1) = (\alpha\beta/(2V_1), (\alpha^3-\beta^3)/(\alpha^3+\beta^3))$ is a non-torsion point on the curve.*

Proof. Define W_1, A, B as in Theorem 3.22. Then $2(a+d)/(a-d) = A$ and $4/(a-d) = B\beta^2/(2\alpha^2)^2$. The twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$ is birationally equivalent to the Montgomery curve $(B\beta^2/(2\alpha^2)^2)v^2 = u^3 + Au^2 + u$, which in turn is isomorphic to the Montgomery curve $Bv^2 = u^3 + Au^2 + u$, so its $\mathbb{Q}$-torsion group has a subgroup isomorphic to $\mathbb{Z}/6\mathbb{Z}$ by Theorem 3.22.
Define u_1, v_1 as in Theorem 3.22. Then (u_1, v_1) is a non-torsion point on $Bv^2 = u^3 + Au^2 + u$, so $(u_1, v_1(2\alpha^2)/\beta)$ is a non-torsion point on $(B\beta^2/(2\alpha^2)^2)v^2 = u^3 + Au^2 + u$. Mapping this point to $E_{\mathrm{E},a,d}$ yields exactly (x_1, y_1):

$$x_1 = \frac{u_1}{v_1(2\alpha^2)/\beta} = \frac{\alpha^3}{V_1(2\alpha^2)/\beta} = \frac{\alpha\beta}{2V_1} \quad \text{and} \quad y_1 = \frac{u_1 - 1}{u_1 + 1} = \frac{\alpha^3 - \beta^3}{\alpha^3 + \beta^3}.$$

Hence (x_1, y_1) is a non-torsion point on $ax^2 + y^2 = 1 + dx^2y^2$.

□

Remark 3.24 (Small factors in the group order)**.** Most Suyama curves have $\mathbb{Q}$-torsion group only $\mathbb{Z}/6\mathbb{Z}$. Montgomery in [Mon92, Section 6] selected various curves with torsion group $\mathbb{Z}/12\mathbb{Z}$, computed the group orders modulo primes p in the interval $[10^4, 10^5]$, and found that the average exponents of 2 and 3 in the group orders were almost exactly 11/3 and 5/3 respectively. For most Suyama curves with torsion group $\mathbb{Z}/6\mathbb{Z}$ the averages are only 10/3 and 5/3. Kruppa noticed that the Suyama curve with $\sigma = 11$ has the average exponents 11/3 and 5/3. His findings were published in his Ph.D. thesis [Kru10] in 2010, though the discovery was already in 2007. After discussions with Zimmermann and Kruppa during a research stay in Nancy in November 2007 motivated by the "$\sigma = 11$"-case we performed an analogous computation for primes in $[10^6, 2 \cdot 10^6]$, using Edwards curves with torsion group $\mathbb{Z}/12\mathbb{Z}$ constructed as in Section 3.2. It turned out that Edwards curves produce an even closer match to 11/3 and 5/3.

3.4 Edwards curves with small parameters, large torsion, and positive rank

One way to save time in computations on twisted Edwards curves is to choose small curve parameters a and d and a small-height non-torsion base point $(X_1 : Y_1 : Z_1)$. Another way to save time is to construct curves with large $\mathbb{Q}$-torsion group and positive rank; see Section 3.3. Unfortunately, essentially all of the curves constructed in Section 3.3 have a, d, X_1, Y_1, Z_1 of large height.
This section combines these two time-saving techniques, finding twisted Edwards curves that simultaneously have small parameters a, d, a small-height non-torsion point $(X_1 : Y_1 : Z_1)$, and large torsion over $\mathbb{Q}$.
The search for small curves for the article [BBLP08] resulted in more than 100 Edwards curves with small-height curve coefficient d and small-height non-torsion points and at least 12 torsion points over $\mathbb{Q}$. See the website [BBLP] for the complete list.

The number of d's below height H appears to grow as roughly $\log_2 H$; for comparison, the Atkin-Morain procedure discussed in Section 3.3 generates only about $\sqrt{\log_2 H}$ examples below height H. Of course, one can easily write down many more small curves if one is willing to sacrifice some torsion.

3.4.1 Torsion group $\mathbb{Z}/12\mathbb{Z}$

Theorem 3.12 gives a complete parametrization of all Edwards curves with torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$. Any rational point $(u, x_3, y_3, d, x_1, y_1)$ on the surface described by

$$x_3 = \frac{u^2-1}{u^2+1},\ y_3 = -\frac{(u-1)^2}{u^2+1},\ d = \frac{(u^2+1)^3(u^2-4u+1)}{(u-1)^6(u+1)^2},\ x_1^2 + y_1^2 = 1 + dx_1^2y_1^2$$

gives us a suitable curve for ECM if $u \notin \{0, \pm 1\}$ and (x_1, y_1) is not a torsion point. Theorem 3.11 lists all affine torsion points.
Assume without loss of generality that $|u| > 1$: otherwise replace u by $1/u$, obtaining the same d. Write u as a/b for integers a, b satisfying $0 < |b| < a$. Define $e = (a^2 - b^2)/x_1$ and $f = -(a-b)^2/y_1$, and assume without loss of generality that e, f are integers; otherwise scale a, b appropriately. The curve equation $x_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ now implies, after some simplification, the $(1, 1, 2, 2)$-weighted-homogeneous equation

$$(e^2 - (a^2-b^2)^2)(f^2 - (a-b)^4) = 16a^3b^3(a^2 - ab + b^2).$$

There are many small solutions to this equation, and thus we find many of the desired Edwards curves, as follows. We considered a range of positive integers a. For each a we enumerated integers b with $0 < |b| < a$. For each (a, b) we enumerated all divisors of $16a^3b^3(a^2 - ab + b^2)$ and added $(a^2 - b^2)^2$ to each divisor. For each sum of the form e^2 we added $(a-b)^4$ to the complementary divisor, checked for a square, checked that the corresponding (x_1, y_1) was a non-torsion point, etc.
After about a week of computation on some computers at LORIA we had found 78 different values of d and checked that we had 78 different j-invariants. Here are two examples:

- the very small solution $(a, b, e, f) = (3, 2, 23, 7)$ produces the order-3 point $(5/13, -1/13)$ and the non-torsion point $(5/23, -1/7)$ on the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ where $d = -11 \cdot 13^3/5^2$;

- the solution $(a, b, e, f) = (15180, -7540, 265039550, 161866240)$ produces the non-torsion point $(3471616/5300791, -201640/63229)$ on the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ where $d = 931391 \cdot 359105^3/140003330048^2$.

3.4.2 Torsion group $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$

Theorem 3.15 gives a complete parametrization of all Edwards curves with torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ and with a point of order 8 doubling to $(\pm 1, 0)$.

Any rational point (u, x_8, d, x_1, y_1) on the surface described by $x_8 = (u^2 + 2u + 2)/(u^2 - 2)$, $d = (2x_8^2 - 1)/x_8^4$, and $x_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ gives us a suitable curve for ECM if $u \notin \{0, -1, -2\}$ and (x_1, y_1) is not a torsion point. Theorem 3.14 lists all affine torsion points.

We consider only $u > \sqrt{2}$. Various transformations of u listed in Theorem 3.15 show that this does not lose any generality: if $0 < u < \sqrt{2}$ then $2/u > \sqrt{2}$, and $2/u$ produces the same curve; if $u < -2$ then $-(u + 2) > 0$, and $-(u + 2)$ produces the same curve; if $-2 < u < -1$ then $-2(u + 1)/(u + 2) > 0$, and $-2(u + 1)/(u + 2)$ produces the same curve; if $-1 < u < 0$ then $-u/(u + 1) > 0$, and $-u/(u + 1)$ produces the same curve.

Write $u = a/b$, $x_1 = (a^2 + 2ab + 2b^2)/e$, and $y_1 = (a^2 + 2ab + 2b^2)/f$ where a, b, e, f are integers. Then a, b, e, f satisfy the $(1, 1, 2, 2)$-weighted-homogeneous equation

$$(e^2 - (a^2 + 2ab + 2b^2)^2)(f^2 - (a^2 + 2ab + 2b^2)^2) = (4ab(a + b)(a + 2b))^2.$$

We found many small solutions to this equation, and thus many of the desired Edwards curves, by a procedure similar to the procedure used for $\mathbb{Z}/12\mathbb{Z}$. We considered a range of positive integers a. For each a we enumerated integers b between 1 and $\lfloor a/\sqrt{2} \rfloor$. For each (a, b) we enumerated all divisors of $(4ab(a + b)(a + 2b))^2$, added $(a^2 + 2ab + 2b^2)^2$ to each divisor, and searched for squares.

After about a week of computation on some computers at LORIA, we had found 25 different values of d and checked that we had 25 different j-invariants. Here are two examples:

- the very small solution $(a, b, e, f) = (3, 1, 19, 33)$ produces the order-8 point $(17/7, 17/7)$ and the non-torsion point $(17/19, 17/33)$ on the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ where $d = 161^2/17^4$;
- the solution $(a, b, e, f) = (24882, 9009, 258492663, 580153002)$ produces the non-torsion point $(86866/18259, 8481/4001)$ on the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ where $d = 5657719^2/3341^4$.

3.5 The impact of large torsion

This section reports various measurements of the success probability of EECM-MPFQ. These measurements demonstrate the importance of choosing a curve with a large torsion group. They also demonstrate the inaccuracy of several common methods of estimating the success probability of ECM.

3.5.1 Impact of torsion for 20-bit and 30-bit primes

There are exactly 38635 primes between 2^{19} and 2^{20}. As an experiment we fed each of these primes to EECM-MPFQ with $B_1 = 256$ and $d_1 = 1$; note that d_1 is a stage-2 parameter described in detail in [BBLP08, Section 5]. The details of our experiments

are summarized on [BBLP]. It turned out that the first curve configured into EECM-MPFQ finds 12467, i.e., 32.27%, of these primes. This curve is the Edwards curve $x^2 + y^2 = 1 - (24167/25)x^2y^2$, with base point $P = (5/23, -1/7)$; this curve has torsion group isomorphic to $\mathbb{Z}/12\mathbb{Z}$.

We then modified EECM-MPFQ to instead start with the curve $x^2 + y^2 = 1 + (25921/83521)x^2y^2$, with base point $P = (13/7, 289/49)$, and repeated the same experiment. This curve has torsion group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$; it is one of the curves that EECM-MPFQ normally tries, although not the first in the list. This curve finds 32.84% of the primes.

We then made a more drastic modification to EECM-MPFQ, trying two new curves with smaller torsion groups. The curve $x^2 + y^2 = 1 + (1/36)x^2y^2$, with base point $P = (8, 9)$, has torsion group only $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$ and finds only 27.49% of the primes, losing a factor 1.17 compared to the original $\mathbb{Z}/12\mathbb{Z}$ curve. The curve $x^2 + y^2 = 1 + (1/3)x^2y^2$, with base point $P = (2, 3)$, has torsion group only $\mathbb{Z}/4\mathbb{Z}$ and finds only 23.47% of the primes, losing a factor 1.37 compared to the original $\mathbb{Z}/12\mathbb{Z}$ curve.

As a larger experiment we replaced the 38635 20-bit primes by a random sample of 65536 distinct 30-bit primes and increased (B_1, d_1) from $(256, 1)$ to $(1024, 1)$. The same four curves again had remarkably different performance:

- 12.16% of the primes were found by the $\mathbb{Z}/12\mathbb{Z}$ curve.
- 11.98% of the primes were found by the $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ curve.
- 9.85% of the primes were found by the $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$ curve.
- 9.00% of the primes were found by the $\mathbb{Z}/4\mathbb{Z}$ curve.

For comparison, GMP-ECM with a typical Suyama curve (specifically $\sigma = 10$) finds 11.68% of the same primes. We also tried GMP-ECM's Pollard $p-1$ option; it found 6.35% of the same primes. Normally the $p - 1$ method is assumed to be a helpful first step before ECM, because it uses fewer multiplications per bit than an elliptic curve, but we comment that this benefit is reduced by the $p-1$ curve (a hyperbola) having torsion group only $\mathbb{Z}/2\mathbb{Z}$.

Figures 3.1 and 3.2 show the results of similar measurements for the same four EECM curves for many prime powers B_1: specifically, every prime power $B_1 \leq 500$ for the 20-bit primes, and every prime power $B_1 \leq 2000$ for the 30-bit primes. The figures show that $\mathbb{Z}/12\mathbb{Z}$ and $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ are consistently better than $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$ (lower) and $\mathbb{Z}/4\mathbb{Z}$ (lower).

The figures also include measurements for the same GMP-ECM Suyama curve and $p-1$ (lower). When B_1 is large, the EECM-MPFQ $\mathbb{Z}/12\mathbb{Z}$ and $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ curves find significantly more primes than the GMP-ECM Suyama curve.

Note that [BBLP08] contains colored versions of these two graphs; also the digital version of this thesis contains colored graphs.

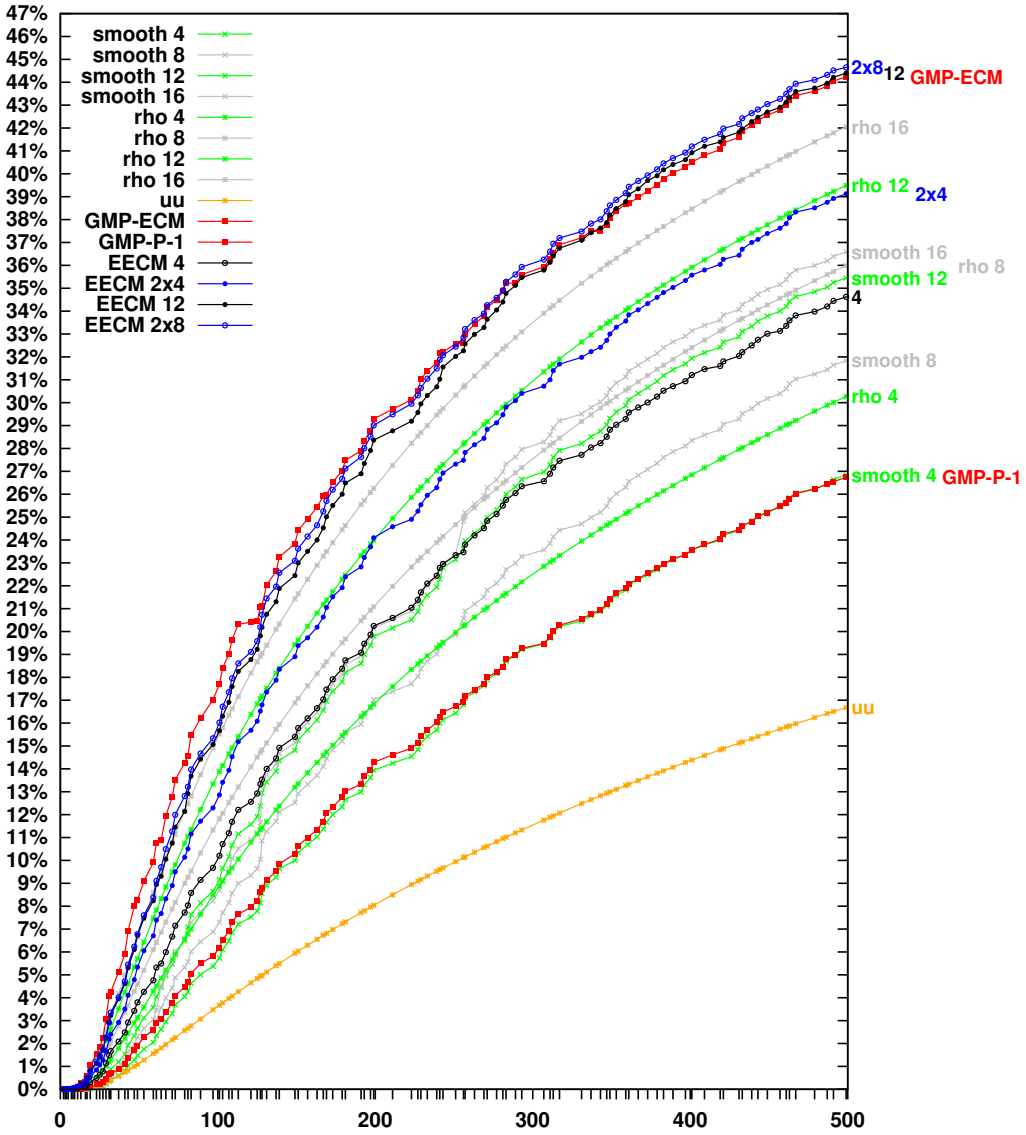

Figure 3.1: For the set of all 38635 20-bit primes: Measured stage-1 success probabilities for six curves, and nine estimates. Horizontal axis is B_1. Vertical axis is probability. Graphs from top to bottom on right side: (bumpy) EECM-MPFQ with a $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ curve; (bumpy) EECM-MPFQ with a $\mathbb{Z}/12\mathbb{Z}$ curve; (bumpy) GMP-ECM with a Suyama curve; (smooth) the ρ approximation to smoothness probability for $[1, 2^{20}/16]$; (smooth) the ρ approximation for $[1, 2^{20}/12]$; (bumpy) EECM-MPFQ with a $\mathbb{Z}/2\mathbb{Z}\times\mathbb{Z}/4\mathbb{Z}$ curve; (bumpy) powersmoothness probability for $16\mathbb{Z} \cap [2^{19}, 2^{20}]$; (smooth) the ρ approximation for $[1, 2^{20}/8]$; (bumpy) powersmoothness probability for $12\mathbb{Z} \cap [2^{19}, 2^{20}]$; (bumpy) EECM-MPFQ with a $\mathbb{Z}/4\mathbb{Z}$ curve; (bumpy) powersmoothness probability for $8\mathbb{Z} \cap [2^{19}, 2^{20}]$; (smooth) the ρ approximation for $[1, 2^{20}/4]$; (bumpy) powersmoothness probability for $4\mathbb{Z} \cap [2^{19}, 2^{20}]$; (bumpy) GMP-ECM with $p - 1$; (smooth) the u^{-u} approximation for $[1, 2^{20}]$.

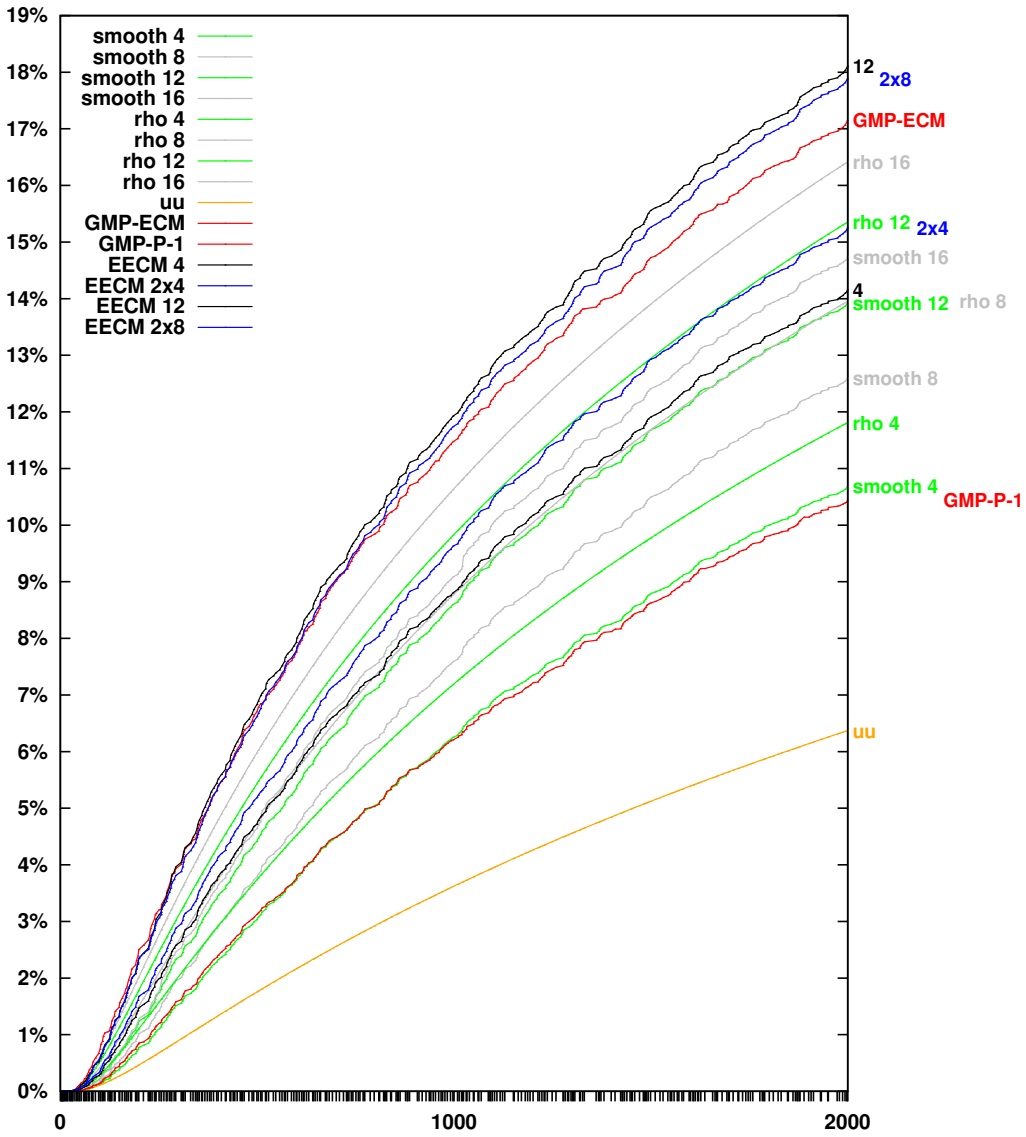

Figure 3.2: For a sample of 65536 30-bit primes: Measured stage-1 success probabilities for six curves, and nine estimates. Horizontal axis is B_1. Vertical axis is probability. Graphs from top to bottom on right side: (bumpy) EECM-MPFQ with a $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ curve; (bumpy) EECM-MPFQ with a $\mathbb{Z}/12\mathbb{Z}$ curve; (bumpy) GMP-ECM with a Suyama curve; (smooth) the ρ approximation to smoothness probability for $[1, 2^{30}/16]$; (smooth) the ρ approximation for $[1, 2^{30}/12]$; (bumpy) EECM-MPFQ with a $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$ curve; (bumpy) powersmoothness probability for $16\mathbb{Z} \cap [2^{29}, 2^{30}]$; (bumpy) EECM-MPFQ with a $\mathbb{Z}/4\mathbb{Z}$ curve; (smooth) the ρ approximation for $[1, 2^{30}/8]$; (bumpy) powersmoothness probability for $12\mathbb{Z} \cap [2^{29}, 2^{30}]$; (bumpy) powersmoothness probability for $8\mathbb{Z} \cap [2^{29}, 2^{30}]$; (smooth) the ρ approximation for $[1, 2^{30}/4]$; (bumpy) powersmoothness probability for $4\mathbb{Z} \cap [2^{29}, 2^{30}]$; (bumpy) GMP-ECM with $p - 1$; (smooth) the u^{-u} approximation for $[1, 2^{30}]$.

3.5.2 Review of methods of estimating the success probability

Consider the fraction of primes $p \in [L, R]$ found by stage 1 of ECM with a particular curve E, point $P \in E(\mathbb{Q})$, and smoothness bound B_1. Assume that E is chosen to guarantee t as a divisor of $E(\mathbb{F}_p)$.
Standard practice in the literature is to estimate this fraction through the following series of heuristic approximations:

$$\begin{aligned}
&\Pr[\text{uniform random prime } p \in [L,R] \text{ has } B_1\text{-powersmooth } \#\langle P \text{ in } E(\mathbb{F}_p)\rangle] \\
&\overset{?}{\approx} \Pr[\text{uniform random prime } p \in [L,R] \text{ has } B_1\text{-powersmooth } \#E(\mathbb{F}_p)] \\
&\overset{?}{\approx} \Pr[\text{uniform random} \in t\mathbb{Z} \cap [(\sqrt{L}-1)^2, (\sqrt{R}+1)^2] \text{ is } B_1\text{-powersmooth}] \\
&\overset{?}{\approx} \Pr[\text{uniform random} \in t\mathbb{Z} \cap [L,R] \text{ is } B_1\text{-powersmooth}] \\
&\overset{?}{\approx} \Pr[\text{uniform random} \in t\mathbb{Z} \cap [1,R] \text{ is } B_1\text{-powersmooth}] \\
&\overset{?}{\approx} \Pr[\text{uniform random} \in \mathbb{Z} \cap [1,R/t] \text{ is } B_1\text{-powersmooth}] \\
&\overset{?}{\approx} \rho(u) \text{ where } B_1^u = R/t \\
&\overset{?}{\approx} 1/u^u.
\end{aligned}$$

Here "B_1-powersmooth" means "having no prime-power divisors larger than B_1," and ρ is Dickman's rho function introduced in [Dic30]. Similar comments apply to stage 2, with B_1-powersmoothness replaced by a more complicated notion of smoothness and with ρ replaced by a standard generalization.
For example, Montgomery in [Mon92, Section 7] estimated the success chance of a curve with 16 torsion points over $\mathbb{Q}$ as the B_1-powersmoothness chance for a uniform random integer in $[1, p/16]$. Similarly, Silverman and Wagstaff in [SW93] estimated the success chance of a Suyama curve as the B_1-powersmoothness chance for a uniform random integer in $[1, p/12]$, following Brent's comment in [Bre86, Section 9.3] that choosing a Suyama curve "effectively reduces p to $p/12$ in the analysis." (As mentioned in Section 3.3, a typical Suyama curve has only 6 torsion points over $\mathbb{Q}$, but a Suyama curve modulo p is guaranteed to have order in $12\mathbb{Z}$.) Brent, Montgomery, et al. used Dickman's rho function to estimate the B_1-powersmoothness chance for a uniform random integer.

Inaccuracy of the estimates. There are many reasons to question the accuracy of the above approximations:

- Dickman's rho function ρ is asymptotically $1/u^u$ in the loose sense that

  $$(\log \rho(u))/(-u \log u) \to 1 \quad \text{as} \quad u \to \infty,$$

  but is not actually very close to $1/u^u$: for example, $\rho(2) \approx 1.11/2^2$, $\rho(3) \approx 1.31/3^3$, and $\rho(4) \approx 1.26/4^4$.

- For each $u \geq 0$, the B_1-smoothness probability for an integer in $[1, B_1^u]$ converges to $\rho(u)$ as $B_1 \to \infty$, and the same is true for B_1-powersmoothness, but the convergence is actually quite slow.

- Multiplying an element of $\mathbb{Z} \cap [1, R/16]$ by 16 never gains powersmoothness but can lose powersmoothness when the original exponent of 2 was large, not an uncommon event among powersmooth integers.

- The ratio of smoothness probabilities for (e.g.) $[1, B_1^u]$ and $[(1/2)B_1^u, B_1^u]$ converges to 1 as $B_1 \to \infty$, but the convergence is again quite slow.

- Lenstra commented in [Len87b, page 660] that an elliptic curve has even order with probability approximately 2/3, not 1/2. Many subsequent reports (for example, by Brent in [Bre86, Table 3] and McKee in [McK99, Section 2]) have lent support to the idea that elliptic-curve orders are somewhat more likely to be smooth than uniform random integers.

- The group order $\#E(\mathbb{F}_p)$ is a multiple of the point order $\#\langle P \text{ in } E(\mathbb{F}_p)\rangle$. The ratio is usually small but often enough to change powersmoothness, as illustrated by Example 3.4 in Section 3.1.2.

The overall error is not extremely large but can easily be large enough to interfere with optimization.
Recall that the curve $x^2 + y^2 = 1 - (24167/25)x^2y^2$, with 12 torsion points, finds 32.27% of the primes in $[2^{19}, 2^{20}]$ with $B_1 = 256$ and $d_1 = 1$; and that changing to three other curves with 16, 8, and 4 torsion points changes 32.27% to 32.84%, 27.49%, and 23.47% respectively. We computed several of the standard estimates for these four success probabilities:

- A uniform random element of $12\mathbb{Z} \cap [2^{19}, 2^{20}]$ has a 23.61% chance of being 256-powersmooth. Note that this probability drastically underestimates the actual ECM smoothness chance. Changing 12 to $16, 8, 4$ changes 23.61% to 24.82%, 20.58%, and 16.80% respectively.

- A uniform random element of $12\mathbb{Z} \cap [1, 2^{20}]$ has a 30.03% chance of being 256-powersmooth. Changing 12 to $16, 8, 4$ changes 30.03% to 31.30%, 26.43%, and 21.86% respectively.

- A uniform random element of $\mathbb{Z} \cap [1, 2^{20}/12]$ has a 30.77% chance of being 256-powersmooth. Changing 12 to $16, 8, 4$ changes 30.77% to 33.37%, 27.37%, and 22.25% respectively.

- If $u = (\log(2^{20}/12))/\log 256$ then $\rho(u) \approx 28.19\%$. Changing 12 to $16, 8, 4$ changes 28.19% to 30.69%, 24.98%, and 20.24% respectively.

- If $u = (\log(2^{20}/12))/\log 256$ then $u^{-u} \approx 22.88\%$. Changing 12 to $16, 8, 4$ changes 22.88% to 25%, 20.15%, and 16.13% respectively.

These approximations make 16 seem better than 12 by factors of 1.051, 1.042, 1.085, 1.089, and 1.093, when in fact 16 is better than 12 by a factor of only 1.018. Figure 3.1 includes, for many prime powers B_1, the B_1-powersmoothness chance of a uniform random element of $t\mathbb{Z} \cap [2^{19}, 2^{20}]$ for four values of t, and $\rho((\log(2^{20}/t))/\log B_1)$ for four values of t. Figure 3.2 includes analogous results for 30-bit primes. It is clear that the ρ value is a poor approximation to the powersmoothness chance, and that the powersmoothness chance is a poor approximation to the ECM success chance. One can ask whether better approximations are possible. We comment that a fast algorithm to compute tight bounds on smoothness probabilities appeared in [Ber02], and that the same algorithm can be adapted to handle powersmoothness, local conditions such as more frequent divisibility by 2, etc. However, one can also ask whether approximations are necessary in the first place. ECM is most frequently used to find rather small primes (for example, inside the number-field sieve), and for those primes one can simply measure ECM's performance by experiment.

Part II

Code-based cryptography

Chapter 4

Linear codes for cryptography

The background for code-based cryptography is the theory of *error-correcting codes* which came up in the middle of the 20th century, motivated by Claude Shannon's article "A mathematical theory of communication" [Sha48]. Shannon lay the foundation for *information theory* which deals with the transmission of messages over noisy channels. A *message* is a string of symbols which can be corrupted during a transmission in the sense that instead of symbol a symbol b is received. In order to detect or even correct such a transmission *error* a message word is sent together with some redundant information. Specifically, all possible message words of fixed length over a given alphabet are mapped injectively into a set of *codewords*. This process is called *encoding*. Instead of sending a message directly, the corresponding codeword is sent over the channel. During this transmission some errors might occur. On the other end of the channel the receiver tries to retrieve the codeword by correcting those errors using a *decoding algorithm*. If the decoding process is successful the receiver can retrieve the original message from the codeword. The efficiency of such a decoding algorithm depends on the used *code*. There are many families of codes allowing fast and efficient error correction.
Classically, error-correcting codes are used for communication over noisy channels which appear in all forms of telecommunication but codes are also of interest for storing data. McEliece in [McE78] proposed to use error-correcting codes for public-key cryptography. In the public-key setting the data transmission is assumed to be error-free and errors are deliberately introduced into a codeword as part of the encryption process. McEliece suggested to create a public and a hidden code. The hidden code is part of the private key and allows fast error correction. The public code on the other hand does not come with any structure or obvious decoding algorithm. Message encryption means mapping the message to a codeword in the public code and then adding errors to it. Message decryption relies on the secret connection between public and hidden code. After a conversion step the errors in the ciphertext can be corrected using the decoding algorithm for the hidden code and then the original message can be derived using linear algebra.
This chapter gives an overview on linear error-correcting codes and introduces the McEliece cryptosystem. The organization of this chapter is as follows:

- Section 4.1 introduces basic notation from coding theory.
- The introduction to Goppa codes in Section 4.1.3 is taken in parts from

[BLP11] which is joint work with Bernstein and Lange.

- Section 4.2 gives the basic definitions of the McEliece and Niederreiter cryptosystem and discusses possible attack scenarios. The description how to reduce general decoding to the problem of finding low-weight words is essentially taken from [BLP08] which is joint work with Bernstein and Lange.

4.1 Linear codes

The McEliece cryptosystem uses error-correcting codes, more specifically classical Goppa codes. This section provides the reader with the necessary background. For a more extensive background on coding theory the reader might want to refer to [MS77] and [HP03].

4.1.1 Basic concepts

Fix a finite field $\mathbb{F}_q$. Unless stated otherwise elements of $\mathbb{F}_q^n$ are viewed as $1 \times n$ matrices; the transpose of $v \in \mathbb{F}_q^n$ is denoted by v^t. Similarly the transpose of any matrix Q is denoted by Q^t.

Definition 4.1. Given a matrix $G \in \mathbb{F}_q^{k \times n}$ of rank k. The set $C = \left\{mG : m \in \mathbb{F}_q^k\right\}$ is called a *linear code* of *length* n and *dimension* k, specifically the linear code *generated by* G. The matrix G is called a *generator matrix* for this code as the rows of G form a basis of the code. The elements of C are called *codewords*. A linear code can also be viewed as the kernel of a matrix. In particular, if the linear code C equals $\left\{c \in \mathbb{F}_q^n : Hc^t = 0\right\}$ then the matrix H is called a *parity-check matrix* for the code. If q equals 2 we speak of *binary* linear codes. Otherwise we speak of *q-ary* linear codes.

A linear code C is a k-dimensional subspace of an n-dimensional vector space over the field $\mathbb{F}_q$. We sometimes call a linear code of length n and dimension k an $[n, k]$ code. The matrices G and H are not unique. Given a generator matrix G for a linear code C one can easily determine a parity-check matrix H for C by linear transformations. In particular, if G has *systematic form*, i.e., $G = (I_k|Q)$ where Q is a $k \times (n-k)$ matrix, then $H = (-Q^t|I_{n-k})$ is a parity-check matrix for the code C.

Definition 4.2. The *Hamming distance* $\mathrm{dist}(x, y)$ between two words x, y in $\mathbb{F}_q^n$ is the number of coordinates where they differ. The *Hamming weight* $\mathrm{wt}(x)$ of an element $x \in \mathbb{F}_q^n$ is the number of nonzero coordinates in x. The *minimum distance* of a linear code C is the smallest Hamming distance between different codewords. In the case that C is a nontrivial linear code the minimum distance is the smallest Hamming weight of a nonzero codeword in C.

The Hamming distance is a metric on $\mathbb{F}_q^n$. Note that the Hamming weight of a word is equal to its distance to the all-zero word in $\mathbb{F}_q^n$. In many cases we will simply

write *weight* and *distance* when speaking of the Hamming weight or the Hamming distance.

Definition 4.3. A *ball* of *radius* w centered at an element $x \in \mathbb{F}_q^n$ is the set of all elements $y \in \mathbb{F}_q^n$ with $\mathrm{dist}(x, y) \leq w$.

From now on C denotes a q-ary linear code of length n and dimension k.

Definition 4.4. Let H be a parity-check matrix for C. The *syndrome* of a vector y in $\mathbb{F}_q^n$ with respect to H is the (column) vector Hy^t in $\mathbb{F}_q^{n-k}$.

The code C consists of all elements y in $\mathbb{F}_q^n$ with zero syndrome.

Remark 4.5. Let $y = c + e$ for a codeword c and a weight-w word e. By linearity one has $Hy^t = H(c+e)^t = Hc^t + He^t = He^t$ since $Hc^t = 0$. If H is a parity-check matrix for a *binary* linear code then the syndrome is the sum of the w columns of H that are indexed by the positions of 1's in e.

The algorithms in the following chapters need the notion of an information set.

Remark 4.6 (Information sets)**.** An *information set* of a linear code of length n and dimension k with generator matrix G is a size-k subset $I \subseteq \{1, \ldots, n\}$ such that the columns of G indexed by I form an invertible $k \times k$ submatrix; this submatrix is denoted by G_I. In particular, $G_I^{-1}G$ is a generator matrix for C in systematic form (up to permutation of columns). If H is a parity-check matrix with $GH^t = 0$ then the columns of H indexed by $\{1, \ldots, n\} \setminus I$ form an invertible $(n-k) \times (n-k)$ submatrix.
Let $c = mG_I^{-1}G$ for some vector m in $\mathbb{F}_q^k$. Then the I-indexed entries of c form the information vector m; in particular, these entries are often called *information symbols*. The entries of c which are not indexed by I are *redundant* information, usually called *parity-check symbols*.

Similar to the notation G_I we use y_I for the restriction of a vector y to the positions indexed by a subset $I \subset \{1, \ldots, n\}$.

4.1.2 The general decoding problem

Definition 4.7. A *(w-error-correcting) decoding algorithm* for a linear code C receives a vector y in $\mathbb{F}_q^n$ and a positive integer w as inputs. The output is the set of all elements $c \in C$ at distance at most w from y; the set is empty if there is no such c.

The linear codes that are interesting candidates for the McEliece cryptosystem are codes allowing fast error correction, i.e., fast computation of an *error vector* e of weight $\leq w$ such that $y - e$ lies in C.

Remark 4.8 (Error-correcting capability)**.** For a vector y and $w \leq \lfloor (d-1)/2 \rfloor$ a decoding algorithm uniquely outputs the closest codeword c if d is the minimum distance of C. In particular, a linear code with minimum distance d has *error-correcting capability* $\lfloor (d-1)/2 \rfloor$.

Definition 4.9. A *list-decoding algorithm* is a w-error correcting decoding algorithm with $w > \lfloor (d-1)/2 \rfloor$. Such an algorithm outputs a (possibly empty) list of closest codewords to a given vector y.

In the context of attacking the McEliece cryptosystem we are interested in decoding algorithms which correct a fixed number of errors. We define the "fixed-distance-decoding problem" in the following remark.

Remark 4.10 (Fixed-distance decoding)**.** A *fixed-distance decoder* receives a vector y in $\mathbb{F}_q^n$ and a positive integer w as inputs. The output is a set of all elements $e \in \{y - c \ : \ c \in C\}$ having weight w; the set can be empty if no such e exists.
Note that the outputs of the fixed-distance-decoding algorithms in this thesis consist of error vectors e, rather than codewords $y - e$. In the special case $y = 0$, a fixed-distance-decoding algorithm searches for codewords of weight w. Any fixed-distance-decoding algorithm can easily be adapted to the problem of finding a codeword at distance between 1 and w from y, the problem of finding a codeword of weight between 1 and w, the problem of finding a codeword at distance between 0 and w from y, etc.

Remark 4.11. Finding a good decoding algorithm for a general linear code is a difficult problem. In the case of the McEliece cryptosystem we will be given a parity-check matrix of a code with no obvious structure. The goal is to determine a low-weight word e such that $He^t = s$ for a given syndrome s. This problem is called the *syndrome-decoding problem*. The corresponding decision problem was proven NP-complete for binary linear codes in [BMvT78]. For the q-ary case see [Bar94] and [Bar98, Theorem 4.1].

There are many code families with fast decoding algorithms, e.g., Goppa codes, (generalized) Reed–Solomon codes, Gabidulin codes, Reed-Muller codes, algebraic-geometric codes, BCH codes etc. Section 4.1.3 focuses on classical Goppa codes. For a more extensive study of the aforementioned code families see [MS77] and more up-to-date [HP03].
The best method currently known to decode a random-looking code is "information-set decoding" which will be introduced in Section 5.1.1. Note that information-set decoding — though much more efficient than a brute-force search for a low-weight word — still needs exponential time in the code length. For the purposes of Section 5.4 which deals with asymptotic complexities of information-set-decoding algorithms we now introduce the Gilbert–Varshamov distance.

Definition 4.12. The *Gilbert–Varshamov distance* of a q-ary linear code of length n and dimension k is defined as the maximal integer d_0 such that

$$\sum_{i=0}^{d_0-1} \binom{n}{i} (q-1)^i \leq q^{n-k}. \tag{4.1}$$

Remark 4.13. To the best of our knowledge the name "Gilbert–Varshamov distance" is due to Barg in [Bar98, Section 1.2]. It comes from Gilbert's lower bound on the size of a linear code with fixed length n and fixed minimum distance d_0. Varshamov's bound differs only slightly from (4.1). Both bounds are asymptotically the same; see e.g. [HP03, Section 2.10.6].
If an $[n, k]$ code has Gilbert-Varshamov distance d_0 then there exists an $(n-k) \times n$ matrix H over $\mathbb{F}_q$ such that every set of $d_0 - 1$ columns of H is linearly independent.

Another notion needed for the asymptotic analyses are the rate and the error fraction:

Definition 4.14. The *information rate* or *rate* describes the amount of information a code can transmit in proportion to its length; the rate is usually denoted by R. If a code C has length n and dimension k, then one has $R = k/n$.

It is common to investigate the asymptotic cost of decoding algorithms for random linear codes with fixed rate and increasing length. Similarly, one can fix the error-correcting capability of a code family while increasing its length. This thesis denotes the *error fraction* a code is supposed to correct by W.
Our asymptotic analyses only deal with decoding methods for binary linear codes. For these matters we recall the definition of the binary entropy function.

Definition 4.15. The *binary entropy function* H_2 is defined as $H_2(x) = -x \log_2 x - (1-x) \log_2(1-x)$.

4.1.3 Classical Goppa codes

This section reviews classical Goppa codes which were introduced by Valery D. Goppa in [Gop70] and [Gop71]. For an overview including proofs of certain properties of Goppa codes and the more general family of alternant codes see [MS77, Section 12.3].
Fix a prime power q; a positive integer m; a positive integer $n \leq q^m$; an integer $t < n/m$; distinct elements $a_1, \ldots, a_n$ in $\mathbb{F}_{q^m}$; and a polynomial $g(x)$ in $\mathbb{F}_{q^m}[x]$ of degree t such that $g(a_i) \neq 0$ for all i.
The words $c = (c_1, \ldots, c_n)$ in $\mathbb{F}_{q^m}^n$ with

$$\sum_{i=1}^{n} \frac{c_i}{x - a_i} \equiv 0 \pmod{g(x)} \tag{4.2}$$

form a linear code $\Gamma_{q^m}(a_1, \ldots, a_n, g)$ of length n and dimension $n - t$ over $\mathbb{F}_{q^m}$.
The code $\Gamma_{q^m}(a_1, \ldots, a_n, g)$ is a special case of a *generalized Reed–Solomon code*; see [MS77, Chapter 10]. The restriction of a generalized Reed–Solomon code over $\mathbb{F}_{q^m}$ to a subfield $\mathbb{F}_q$ is called an *alternant code*; in general the restriction of a code to a smaller field is called a *subfield subcode*.

Definition 4.16. The *Goppa code* $\Gamma_q(a_1, \ldots, a_n, g)$ with *Goppa polynomial* $g(x)$ and *support* $a_1, \ldots, a_n$ is the restriction of $\Gamma_{q^m}(a_1, \ldots, a_n, g)$ to the field $\mathbb{F}_q$, i.e., the set of elements $(c_1, \ldots, c_n)$ in $\mathbb{F}_q^n$ that satisfy (4.2).

Beware that there is a conflicting definition of "support" elsewhere in coding theory. In order to ensure that a chosen Goppa polynomial g does not vanish at the support elements $a_1, \ldots, a_n$ it is common to choose g to be a nonlinear irreducible element of $\mathbb{F}_{q^m}[x]$. In this case $\Gamma_q(a_1, \ldots, a_n, g)$ is called an *irreducible* Goppa code.

Remark 4.17. The code $\Gamma_q(a_1, \ldots, a_n, g)$ is a subfield subcode of $\Gamma_{q^m}(a_1, \ldots, a_n, g)$. The *dimension* of $\Gamma_q(a_1, \ldots, a_n, g)$ is at least $n - mt$. The *minimum distance* of $\Gamma_q(a_1, \ldots, a_n, g)$ is $\geq t + 1$; this is a consequence of Goppa codes being part of the family of alternant codes/generalized Reed–Solomon codes. For details we refer the reader to [MS77, Section 12.3] and [HP03, Section 13.2.2].

Goppa codes can be decoded by any decoder for generalized Reed–Solomon codes. For example, Berlekamp's algorithm corrects $t/2$ errors; see e.g. [Ber84] or [HP03, Section 5.4.2]. Guruswami and Sudan in [GS99] introduced a list-decoding algorithm for correcting more errors than the minimum distance in a generalized Reed–Solomon code. The Guruswami–Sudan decoder applied to a Goppa code corrects about $n - \sqrt{n(n-t)} > t/2$ errors.

Remark 4.18 (Efficient decoding of binary Goppa codes)**.** Note that $t + 1$ is is a lower bound for the minimum distance. There are Goppa codes whose minimum distance is much larger. Binary Goppa codes have minimum distance at least $2t + 1$ ([Gop70][1]). Patterson in [Pat75] devised a fast polynomial-time decoding algorithm for correcting t errors in binary Goppa codes with monic and squarefree polynomial g.
Bernstein in [Ber08b] devised a list-decoding method for binary irreducible Goppa codes which combines the Guruswami–Sudan decoder with Patterson decoding. The decoder corrects up to $n - \sqrt{n(n-2t-2)} > t$ errors; again with the small inefficiency of possibly returning more than one codeword for some inputs. See Chapter 7 and [ABC10] for newer methods.

Remark 4.19 (Parity-check matrix)**.** Let $\Gamma_q(a_1, \ldots, a_n, g)$ be a Goppa code of length n, support $a_1, \ldots, a_n$, and Goppa polynomial g of degree t. Fix a basis

[1] Goppa's article is written in Russian. The proof can be found in many textbooks, e.g., in [MS77].

of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ and write each element of $\mathbb{F}_{q^m}$ with respect to that basis. Then a parity-check matrix for $\Gamma_q(a_1, \ldots, a_n, g)$ is given by the $mt \times n$ matrix

$$H = \begin{pmatrix} \frac{1}{g(a_1)} & \frac{1}{g(a_2)} & \cdots & \frac{1}{g(a_n)} \\ \frac{a_1}{g(a_1)} & \frac{a_2}{g(a_2)} & \cdots & \frac{a_n}{g(a_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_1^{t-1}}{g(a_1)} & \frac{a_2^{t-1}}{g(a_2)} & \cdots & \frac{a_n^{t-1}}{g(a_n)} \end{pmatrix},$$

over $\mathbb{F}_q$ where each entry is actually a column vector written in the chosen $\mathbb{F}_q$-basis of $\mathbb{F}_{q^m}$.

The code $\Gamma_q(a_1, \ldots, a_n, g)$ is often referred to as a "classical" Goppa code since it is the basic construction of a genus-0 geometric Goppa code which Goppa in [Gop77] generalized for higher-genus varieties.
Chapter 7 introduces "wild Goppa codes." For this purpose it is useful to recall the definition of $\Gamma_{q^m}(a_1, \ldots, a_n, g)$ as a generalized Reed–Solomon code which means that $\Gamma_{q^m}(a_1, \ldots, a_n, g)$ and thus Goppa codes can be constructed by evaluating certain functions at $a_1, \ldots, a_n$.

Remark 4.20 (Polynomial view of $\Gamma_{q^m}(a_1, \ldots, a_n, g)$)**.** Define $h(x) = \prod_{i=1}^{n}(x - a_i)$. Note that $g(x)$ and $h(x)$ are coprime. For each $f \in g\mathbb{F}_{q^m}[x]$ define

$$\mathrm{ev}(f) = \left(\frac{f(a_1)}{h'(a_1)}, \frac{f(a_2)}{h'(a_2)}, \ldots, \frac{f(a_n)}{h'(a_n)}\right),$$

where h' denotes the derivative of h.
If f has degree less than n then one can recover it from the entries of $\mathrm{ev}(f)$ by Lagrange interpolation: namely, $f/h = \sum_i (f(a_i)/h'(a_i))/(x - a_i)$. Consequently $\sum_i (\mathrm{ev}(f))_i/(x - a_i)$ is 0 in $\mathbb{F}_{q^m}[x]/g$, where $(\mathrm{ev}(f))_i$ denotes the ith entry of $\mathrm{ev}(f)$. Let $(c_1, \ldots, c_n)$ in $\mathbb{F}_{q^m}^n$ be such that $\sum_i c_i/(x - a_i) \equiv 0 \pmod{g(x)}$. Define $f = \sum_i c_i h/(x - a_i)$ in $\mathbb{F}_{q^m}[x]$. Then $f \in g\mathbb{F}_{q^m}[x]$. Since the polynomial $\sum_i c_i h/(x - a_i)$ has degree less than n, also f has degree less than n. Moreover, $c_j = f(a_j)/h'(a_j) = \mathrm{ev}(f)_j$.
Therefore

$$\begin{aligned} \Gamma_{q^m}(a_1, \ldots, a_n, g) &= \{\mathrm{ev}(f) : f \in g\mathbb{F}_{q^m}[x],\ \deg(f) < n\} \\ &= \{(f(a_1)/h'(a_1), \ldots, f(a_n)/h'(a_n)) : f \in g\mathbb{F}_{q^m}[x],\ \deg(f) < n\}. \end{aligned}$$

4.2 Code-based public-key cryptography

This section describes the McEliece cryptosystem and the Niederreiter cryptosystem and outlines possible attacks.

4.2.1 The McEliece cryptosystem

For the purposes of the McEliece encryption the generator matrix G of a linear code over $\mathbb{F}_q$ should be seen as a map $\mathbb{F}_q^k \to \mathbb{F}_q^n$ sending a message m of length k to an element in $\mathbb{F}_q^n$.
McEliece proposed code-based cryptography using *binary* Goppa codes. This thesis also considers the possibility of using a Goppa code over a larger alphabet; see Chapter 6 and Chapter 7.
The *McEliece public-key cryptosystem* is set up as follows:

- The *secret key* consists of a random classical Goppa code $\Gamma = \Gamma_q(a_1, \ldots, a_n, g)$ over $\mathbb{F}_q$ of length n and dimension k with an error-correction capability of w errors. A generator matrix G for the code Γ as well as an $n \times n$ permutation matrix P, and an invertible $k \times k$ matrix S are randomly generated and kept secret as part of the secret key. In particular, an efficient decoding algorithm for Γ is known.

- The parameters n, k, and w are *public system parameters*.

- The *McEliece public key* is the $k \times n$ matrix $\hat{G} = SGP$ and the error weight w.

Information needs to be embedded in a length-k word $m \in \mathbb{F}_q^k$ in order to be suitable for the encryption algorithm. Then m can be encrypted with the following algorithm.

Algorithm 4.1: McEliece encryption

Input: A message $m \in \mathbb{F}_q^k$, the public key $\hat{G}$, and the parameter w.
Output: A ciphertext $y \in \mathbb{F}_q^n$.

1: Compute $m\hat{G}$.
2: Hide the message by adding a random error vector e of length n and weight w.
3: **Return** $y = m\hat{G} + e$.

The decryption algorithm 4.2 needs to decode the ciphertext y, i.e., determine the error vector e. The legitimate receiver of y, i.e., the owner of the private key, can make use of the hidden Goppa-code structure, in particular of the decoding algorithm for Γ.

Algorithm 4.2: McEliece decryption

Input: A vector $y = m\hat{G} + e \in \mathbb{F}_q^n$, and the private key (Γ, G, P, S) corresponding to $\hat{G}$.
Output: The message m.

1: Compute $yP^{-1} = mSG + eP^{-1}$.
2: Use the decoding algorithm for Γ to find mS. Use linear algebra to retrieve m. **Return** m.

The decryption algorithm works for any *McEliece ciphertext* y, i.e., for any y which is an output of Algorithm 4.1. Indeed, in this case y is known to be at distance w from a vector mSG which is a codeword in Γ. The permutation by P^{-1} of the errors in the error vector does not change the weight of this vector, so it does not affect the decoding algorithm for Γ.

Remark 4.21. McEliece in his original article [McE78] proposed to choose a classical binary Goppa code of length 1024, dimension ≥ 524 with irreducible polynomial g of degree $t = 50$.

4.2.2 The Niederreiter cryptosystem

This section considers a variant of the McEliece cryptosystem published by Niederreiter in [Nie86]. Niederreiter's system, with the same Goppa codes used by McEliece, has the same security as McEliece's system, as shown in [LDW94].
Niederreiter's system differs from McEliece's system in public-key structure, encryption mechanism, and decryption mechanism. Beware that the specific system in [Nie86] also used different codes — Goppa codes were replaced by generalized Reed–Solomon codes — but the Niederreiter system with generalized Reed–Solomon codes was broken by Sidelnikov and Shestakov in 1992; see [SS92].
The *Niederreiter public-key cryptosystem* is set up as follows:

- The *secret key* consists of an $n \times n$ permutation matrix P; a non-singular $(n-k) \times (n-k)$ matrix M; and a parity-check matrix H for a Goppa code $\Gamma = \Gamma_q(a_1, \ldots, a_n, g)$ of dimension k and error-correcting capability w. In particular, an efficient decoding algorithm for Γ is known.

- As in the McEliece cryptosystem, the sizes n, k, w are *public system parameters*, but $a_1, \ldots, a_n$, g, P, and M are randomly generated secrets.

- The *Niederreiter public key* is the $(n-k) \times n$ matrix $\hat{H} = MHP$.

Information needs to be embedded in a length-n word $x \in \mathbb{F}_q^n$ with w nonzero entries in order to be suitable for the encryption algorithm. Then x can be encrypted with the following algorithm; the output is the syndrome of x with respect to the public matrix $\hat{H}$.

Algorithm 4.3: Niederreiter encryption

Input: A message $x \in \mathbb{F}_q^n$ of weight w and the public key $\hat{H}$.
Output: A ciphertext $s \in \mathbb{F}_q^{n-k}$.

1: Compute the syndrome $s = \hat{H}x^t$.
2: **Return** s.

In order to decrypt the message one has to find a weight-w vector x having syndrome s with respect to $\hat{H}$. As in the McEliece cryptosystem, the owner of the private key

can make use of the hidden Goppa-code structure, in particular of the decoding algorithm for Γ.

Algorithm 4.4: Niederreiter decryption

Input: A syndrome $s = \hat{H}x^t \in \mathbb{F}_q^{n-k}$, and the private key (Γ, H, M, S) corresponding to $\hat{H}$.

Output: The message x.

1: Compute z with $Hz^t = M^{-1}s$ using linear algebra.
2: Use the decoding algorithm for Γ to find the codeword $z - xP^t$. Use linear algebra to retrieve x. **Return** x.

The decryption algorithm works for any Niederreiter ciphertext s, i.e., for any s which is an output of Algorithm 4.3. The legitimate receiver knows the matrices H and M and can use linear algebra to compute a vector $z \in \mathbb{F}_q^n$ with $Hz^t = M^{-1}s = HPx^t = H(xP^t)^t$ in Step 1. By linearity $H(z - xP^t)^t = 0$. Hence, $z - xP^t$ is a codeword in Γ at distance w from z and can be found by the decoding algorithm for Γ.

Remark 4.22 (Focus on the McEliece cryptosystem)**.** Note that, in a nutshell, decryption of ciphertexts in both the McEliece and the Niederreiter public-key cryptosystem means finding a certain vector of a given weight. In the following the focus lies on attacking and improving the McEliece cryptosystem.
There are a few attacks on the McEliece cryptosystem which are easier to describe in terms of a parity-check matrix of the public code; for these cases recall that computing a parity-check matrix from a given generator matrix is done using only elementary row operations whose cost will be negligible in comparison to the cost of the attack.

4.2.3 Security of code-based cryptography

The previous two sections introduced textbook versions of code-based public-key cryptography. The original McEliece cryptosystem, like the original RSA cryptosystem, is really just a trapdoor one-way function; when used naively as a public-key cryptosystem it is trivially broken by chosen-ciphertext attacks such as Berson's attack [Ber97] and the Verheul–Doumen–van Tilborg attack [VDvT02]. I.e., McEliece's system as described in the previous sections does not achieve "IND-CCA2 security." For instance, encryption of the same message twice produces two different ciphertexts which can be compared to find out the original message since it is highly unlikely that errors were added in the same positions.
There are several suggestions to make the McEliece cryptosystem CCA2 secure. In [KI01] Kobara and Imai show that adding this protection does not significantly increase the cost of the McEliece cryptosystem. Overviews on CCA2-secure versions of McEliece's scheme can be found in [EOS06, Chapters 5–6] and [OS08]. All techniques share the idea of scrambling the message inputs. The aim is to destroy any

relations of two dependent messages which an adversary might be able to exploit.

Remark 4.23 (Key sizes)**.** In a CCA2-secure version of the McEliece cryptosystem, where the message is scrambled, one can use a systematic generator matrix as public key. This reduces the public-key size from kn bits to $k(n-k)$ bits: it is sufficient to store the non-trivial part of the matrix, i.e., the $k \times (n-k)$ matrix Q in the notation above. Similarly for Niederreiter's system it suffices to store the non-trivial part of the parity-check matrix, reducing the public-key size from $(n-k)n$ bits to $k(n-k)$ bits.

4.2.4 Attacking the McEliece cryptosystem

An attacker of the McEliece cryptosystem is faced with the problem of determining an error vector e given a random-looking generator matrix $\hat{G}$ and a vector $m\hat{G} + e$. Finding e is equivalent to finding the encrypted message m: subtracting e from $m\hat{G} + e$ produces $m\hat{G}$, and then simple linear transformations produce m.
An attacker who got hold of an encrypted message y has two possibilities in order to retrieve the original message m.

- Find out the secret code; i.e., find G given $\hat{G}$, or
- Decode y directly for the public code given by $\hat{G}$.

Attacks of the first type are called *structural attacks*. If G or an equivalently efficiently decodable representation of the underlying code can be retrieved in subexponential time, this code should not be used in the McEliece cryptosystem. Suitable codes are such that the best known attacks are decoding random codes.
McEliece in [McE78] observed that "a fast decoding algorithm exists for a general Goppa code, while no such exists for a general linear code." This is still true. Though the algorithms have been improved over the last more than 30 years the best known methods for decoding a general code all take exponential time.
Canteaut and Chabaud in [CC98, page 368] observed that one can decode a binary linear code — and thus break the McEliece system — by finding a low-weight codeword in a larger code. The following remark explains the reduction.

Remark 4.24 (Reduction of the decoding problem)**.** Let C be a binary linear code of length n, and w a positive integer which is less than half the minimum distance of C. Let $y \in \mathbb{F}_2^n$ be at distance w from C. Denote the closest codeword to y by c. Then $y - c$ is a weight-w element of $\langle C, y \rangle$, the linear code spanned by C and y.
Conversely, if C is a linear code of length n over $\mathbb{F}_2$ with minimum distance larger than $2w$, then a weight-w element $e \in \langle C, y \rangle$ cannot be in C, so it must be in $\langle C, y \rangle$; in other words, $y - e$ is an element of C with distance w from y.

The attacker knows the McEliece public key $\hat{G}$ which generates a code C. Given a McEliece ciphertext $y = c + e$ the attacker can simply append y to the rows of $\hat{G}$ to form a generator matrix for the code $\langle C, y \rangle$. Since the error vector e has weight

w which is less than the assumed minimum distance of C, it is a minimum-weight word in $\langle C, y\rangle$. The attacker then applies a low-weight-word finding algorithm to find e. Similar comments apply if the attacker is given a Niederreiter public key, i.e., a parity-check matrix for a public code C. The bottleneck in all of these attacks is finding the weight-w codeword in $\langle C, y\rangle$ which is a difficult problem as discussed in Remark 4.11.

Beware that there is a slight inefficiency in the reduction from the decoding problem to the problem of finding low-weight codewords: the code C has dimension k and $y \notin C$, thus $\langle C, y\rangle$ has larger dimension, namely $k + 1$. The user of the low-weight-codeword algorithm knows that the generator y will participate in the solution, but does not pass this information to the algorithm.

The following chapter outlines the attack in [BLP08] on the original McEliece parameters; this attack builds as many other articles before on Stern's method to find low-weight words in order to attack the McEliece cryptosystem.

Chapter 5

Collision decoding

The central idea for collision decoding is due to Stern who in [Ste89] published a method for finding low-weight codewords which makes use of the birthday paradox[1]. This chapter discusses attacks against the *binary* version of the McEliece cryptosystem. This chapter also contains an analysis of the asymptotic cost of information-set decoding and in particular of Stern's algorithm.

- Section 5.1 introduces basic information-set decoding and Stern's algorithm, essentially following [BLP08] and [BLPvT09].
- Section 5.2 presents the the break of the original McEliece parameters. The result is based on joint work with Bernstein and Lange and appeared in [BLP08].
- The main differences between the content of Section 5.2 and [BLP08] lie in Section 5.2.1 and Section 5.2.2; there the improvements to Stern's algorithm essentially follow [BLP08] and [Pet10].
- The actual attack on the McEliece cryptosystem is outlined in Section 5.2.4; the CPU timings from [BLP08, Section 6] are omitted.
- Section 5.3 discusses the improvements on collision decoding which were introduced after [BLP08]; the description of the "birthday speedup" is similar to [Pet10, Sections 6–7].
- Section 5.4 presents the analysis of the asymptotic cost of information-set decoding which is joint work with Bernstein, Lange, and van Tilborg and which appeared in [BLPvT09]. Section 5.4 presents the results of this article.
- The main differences between the content of Section 5.4 and [BLPvT09] are that the discussion on fixed-distance decoding was moved in parts to Chapter 4 and Section 5.3; also the descriptions of the algorithms in [BLPvT09, Section 2] is omitted since those are given in Section 5.1.

The Stern algorithm is used to look for a error vector e which is known to have weight w. The assumption throughout this chapter is that there are only a few such vectors e — if any at all. In most cases there is exactly one target vector e.

[1] For a detailed discussion of the birthday paradox we refer the reader to Section 9.2.

5.1 Information-set-decoding algorithms

This section discusses basic information-set-decoding algorithms. The Lee–Brickell algorithm is a decoding algorithm whereas Stern's algorithm originally was intended to find low-weight codewords in a binary linear code. The reduction of decoding to low-weight-word finding was given in Section 4.2.4.

5.1.1 Lee–Brickell's algorithm

The algorithm in this section gets as input a ciphertext y in $\mathbb{F}_2^n$, an error weight w, and a generator matrix G of a *binary* linear code C of length n and dimension k with unknown structure.[2] The closest codeword c in C has distance w from y.

Remark 5.1 (Information-set decoding)**.** Recall that an information set for C is a set I of k positions such that G contains an invertible submatrix G_I which is formed by the columns indexed by I. Any *information-set decoding algorithm* hopes that the error vector e has a certain error pattern with respect to a given information set I. *Plain information-set decoding* hopes that no error occurred among the I-indexed positions of y. Then, $y_I G_I^{-1}$ is the preimage of a codeword $c \in C$ and we can obtain c as $(y_I G_I^{-1})G$.

The extension by Lee and Brickell allows p errors in positions indexed by the information set. These p errors can be corrected by finding the p rows of G corresponding to error indices in I. Leon in [Leo88] and also Krouk in [Kro89] refined Lee–Brickell's algorithm by additionally requiring that the error pattern has weight 0 on ℓ positions outside the information set.

An overview of all error patterns investigated in this chapter is given in Figure 5.2. In the following we will discuss Lee–Brickell's algorithm.

Denote by G_a the unique row of $G_I^{-1}G$ in which the column indexed by some $a \in I$ has a 1.

Let p be an integer with $0 \leq p \leq w$. The algorithm consists of a series of independent iterations, each iteration making random choices. If the set I chosen in Step 1 does not lead to a weight-w word in Step 3 another iteration has to be performed. Each iteration consists of the steps described in Algorithm 5.1.

[2] The notation $\hat{G}$ for the McEliece public key is abandoned. From now on G generates a linear code with unknown structure. Similarly, H is used instead of $\hat{H}$.

Algorithm 5.1: Lee–Brickell algorithm

Input: A generator matrix G in $\mathbb{F}_2^{k\times n}$ of a binary linear code C, a vector $y \in \mathbb{F}_2^n$, and an integer $w \geq 0$.

Output: A weight-w word $e \in \mathbb{F}_2^n$ with $y - e \in C$ if such e exists.

1: Choose a uniform random information set I.
2: Replace y by $y - y_I G_I^{-1} G$.
3: **For each** *size-p subset $A \subseteq I$:* compute $e = y - \sum_{a\in A} G_a$. **If** *$e$ has weight w* **then return** e.
4: Go back to Step 1.

Algorithm 5.1 was published by Lee and Brickell in [LB88] who generalized plain information-set decoding, i.e., the case $p = 0$, which is due to Prange [Pra62].
The parameter p is chosen to be a small number to keep the number of size-p subsets small in Step 3. In the binary case $p = 2$ is optimal as will be shown in Section 5.4.
If e is a uniform random weight-w element of $\mathbb{F}_2^n$, and I is a size-k subset of $\{1, \ldots, n\}$, then e has probability exactly

$$\mathrm{LBPr}(n, k, w, p) = \binom{n-k}{w-p}\binom{k}{p}\binom{n}{w}^{-1}$$

of having weight exactly p on I. Consequently the Lee–Brickell algorithm, given $c+e$ as input for some codeword c, has probability exactly $\mathrm{LBPr}(n, k, w, p)$ of printing e in the first iteration.
We emphasize that these probabilities are averages over e. The iteration can have larger success chances for some vectors e and smaller success chances for others. As an extreme example, take $n = 5$, $k = 1$, $w = 1$, and $p = 0$, and consider the code whose only nonzero codeword is $(1, 1, 0, 0, 0)$. If $e = (0, 0, 1, 0, 0)$ or $(0, 0, 0, 1, 0)$, or $(0, 0, 0, 0, 1)$ then the iteration has chance 1 of printing e; but if $e = (1, 0, 0, 0, 0)$ or $(0, 1, 0, 0, 0)$ then the iteration has chance only $1/2$ of printing e. The value $\mathrm{LBPr}(5, 1, 1, 0) = 4/5$ is the average of these probabilities over all choices of e.
For the same reason, the average number of iterations used by the Lee–Brickell algorithm to find e is not necessarily $1/\mathrm{LBPr}(n, k, w, p)$. In the above example, the average number of iterations is 1 for three choices of e, and 2 for two choices of e. The overall average, if e is uniformly distributed, is $7/5$, while $1/\mathrm{LBPr}(5, 1, 1, 0) = 5/4$.
We nevertheless use $1/\mathrm{LBPr}(n, k, w, p)$ as a *model* for the average number of iterations used by the Lee–Brickell algorithm to find e. Analogous models appear throughout the literature on information-set decoding; for example, Stern says in [Ste89, Section 3] that such models are "a reasonable measure." We have carried out computer experiments for many large random codes, and in every case the average number of iterations was statistically indistinguishable from $1/\mathrm{LBPr}(n, k, w, p)$.

5.1.2 Stern's algorithm

Given a parity-check matrix H, an error weight w, and a syndrome s, Algorithm 5.2 looks for a weight-w word $e \in \mathbb{F}_2^n$ such that $He^t = s$. Stern devised the algorithm

for the case $s = 0$. Then his algorithm tries to build a codeword of weight w by identifying w columns of H which sum up to 0; the corresponding column indices indicate the position of 1's in a weight-w word.
Stern uses the idea of Lee and Brickell to allow p errors in the information set I. He also uses Leon's idea [Leo88] to restrict the number of possible candidates for the low-weight word e to those vectors having ℓ zeros in prespecified positions outside the I-indexed columns.
Stern's algorithm finds a weight-w vector e with $He^t = s$ if an information set I together with sets X, Y, and Z can be found such that e has weights p, p, 0 on the positions indexed by X, Y, and Z, respectively (see Figure 5.2). The main difference to Lee–Brickell's algorithm is that Stern's algorithm tries to build a weight-w word by first looking for collisions among sums of few columns restricted to certain positions and then checks the weight of the column sums arising from the collisions. Dumer in [Dum89] independently introduced the core idea, although in a more limited form, and in [Dum91] achieved an algorithm similar to Stern's.
For a visual interpretation of Algorithm 5.2 we refer the reader to Remark 5.2 and in particular to Figure 5.1. The algorithm has a parameter $p \in \{0, 1, \ldots, w\}$ and another parameter $\ell \in \{0, 1, \ldots, n-k\}$. The algorithm consists of a series of independent iterations, each iteration making random choices. If the set I chosen in Step 1 does not lead to a weight-w word in Step 9 another iteration has to be performed. Each iteration consists of the steps described in Algorithm 5.2.
Algorithm 5.2 was published by Stern in [Ste89] to look for low-weight codewords which is the special case $s = 0$. The algorithm here is described as a fixed-distance-decoding algorithm; for more discussion on phrasing the algorithm in this way we refer the reader to Section 5.3.1.

Remark 5.2. Figure 5.1 gives a visualization of one iteration of Stern's algorithm for the case $s = 0$. Without loss of generality assume that I indexes the first k columns of UH which then can be written as $Q = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix}$ with $Q_1 \in \mathbb{F}_2^{\ell \times k}$, $Q_2 \in \mathbb{F}_2^{(n-k-\ell)\times k}$. The set X indexes columns $1, \ldots, k/2$ and Y indexes columns $k/2 + 1, \ldots, k$. The set Z indexes columns $k + 1, \ldots, k + \ell$. Note that Z also indexes the first ℓ rows of UH; in particular, Z indexes the rows of Q_1. If a collision can be found between p columns of Q_1 indexed by X and p columns of Q_1 indexed by Y, the sum of the columns on all $n - k$ positions is computed and the weight is checked; if s is the sum of $2p$ columns having weight $w - 2p$ then those nonzero positions come from the rows corresponding to Q_2 and can be cancelled by $w - 2p$ ones in the identity matrix on the right which are not indexed by Z.

The expected number of iterations of Stern's algorithm as well as how to choose the parameters p and ℓ are discussed in Section 5.2.

Remark 5.3 (Adaptive information sets)**.** Step 1 can be performed by choosing $n-k$ indices in $\{1, \ldots, n\}$ uniformly at random and then performing Gaussian elimination on H in order to see if these columns form an invertible submatrix. Stern suggests a

Algorithm 5.2: Stern's algorithm

Input: A parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ for a binary linear code C, a column vector $s \in \mathbb{F}_2^{n-k}$, and an integer $w \geq 0$.

Output: A weight-w element $e \in \mathbb{F}_2^n$ with $He^t = s$ if such e exists.

1: Choose a uniform random information set I and bring H in systematic form with respect to I: find an invertible $U \in \mathbb{F}_2^{(n-k)\times(n-k)}$ such the submatrix of UH indexed by $\{1, \ldots, n\} \setminus I$ is the $(n-k) \times (n-k)$ identity matrix.
2: Select a uniform random size-$\lfloor k/2 \rfloor$ subset $X \subset I$.
3: Define $Y = I \setminus X$.
4: Select a size-ℓ subset Z of $\{1, \ldots, n\} \setminus I$.
5: **For each** *size-p subset $A \subseteq X$:* consider the p columns $(UH)_a$ indexed by $a \in A$ and compute $\phi(A) = s - \sum_a (UH)_a$ restricted to ℓ rows indexed by Z.
6: **For each** *size-p subset $B \subseteq Y$:* consider the p columns $(UH)_b$ indexed by $b \in B$ and compute $\psi(B) = \sum_b (UH)_b$ restricted to ℓ rows indexed by Z.
7: **For each** *pair (A, B) such that $\phi(A) = \psi(B)$:*
8: Compute $s' = s - \sum_{i \in A \cup B} (UH)_i$.
9: **If** $\mathrm{wt}(s') = w - 2p$ **then** add the corresponding $w - 2p$ columns in the $(n-k) \times (n-k)$ identity submatrix to make s' the all-zero syndrome. **Return** the vector $e \in \mathbb{F}_2^n$ indicating those columns and the columns indexed by $A \cup B$.
10: Go back to Step 1.

better way of determining an information set I, namely by choosing $n - k$ columns one by one: check for each newly selected column if it does not linearly depend on the already selected columns. In theory this adaptive choice could bias the choice of (X, Y, Z), as Stern points out, but the bias does not seem to have a noticeable effect on performance.

Remark 5.4. All state-of-the-art decoding attacks since [Ste89] have been increasingly optimized forms of Stern's algorithm; we will call any decoding algorithm using Stern's collision idea a *collision-decoding* algorithm. Other approaches to decoding, such as "gradient decoding" ([AB98]), "supercode decoding" ([BKvT99]), and "statistical decoding" (see [Jab01] and [Ove06]), have never been competitive with Stern's algorithm for attacking the McEliece cryptosystem.

5.2 A successful attack on the original McEliece parameters

This section presents the techniques we used to break the original McEliece parameters $n = 1024$, $k = 524$, and $w = 50$ for a binary linear code. The results presented here are based on the article "Attacking and defending the McEliece cryptosystem" [BLP08] which is joint work with Bernstein and Lange. Bernstein implemented

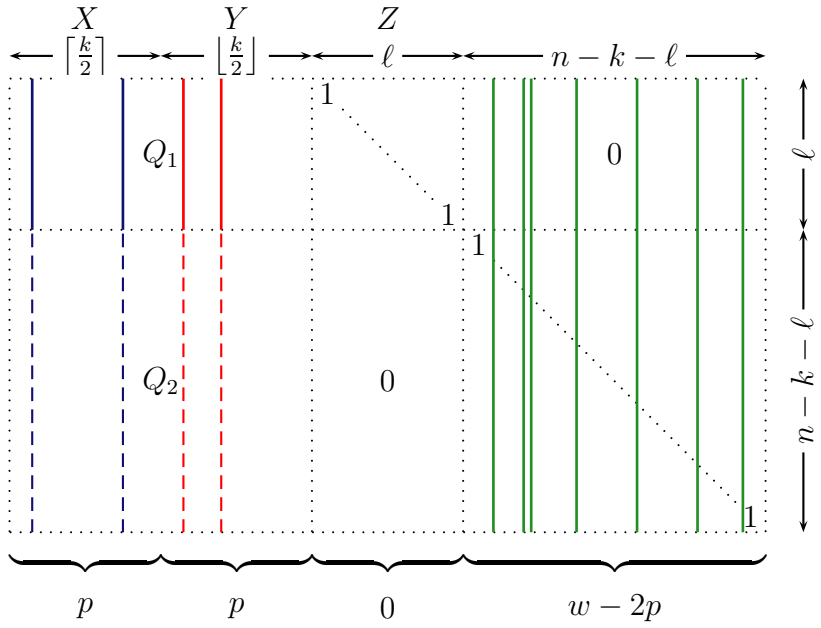

Figure 5.1: Stern's algorithm. The weights p, p, 0, and $w-2p$ indicate how many columns are taken from the respective parts.

the attack. We used his software to successfully decrypt a McEliece ciphertext which was created using McEliece's original parameter settings. The attack uses the reduction to low-weight-word finding; i.e., Stern's algorithm 5.2 with input $s = 0$. The new attack is presented as the culmination of a series of improvements that we have made to Stern's attack. As a result of these improvements, the attack speeds are considerably better than the attack speeds reported by Canteaut, Chabaud, and Sendrier in [CC98] and [CS98]. See Sections 5.2.3 and 5.2.4 for concrete results and comparisons.

Remark 5.5. For the purpose of this section recall that Stern's algorithm consists of basically three steps:

(I) (Updating the matrix) Select an information set I and compute H in systematic form with respect to I.

(II) (List building) Compute all possible sums coming from p columns indexed by a set X on ℓ positions. Similarly, compute all possible sums coming from p columns indexed by a set Y on ℓ positions. In practice this step can be implemented efficiently by sorting, by hashing, or by simple table indexing.

(III) (Collision handling) Compare the entries of the two lists. For each collision between elements of both tables compute the sums of the corresponding $2p$ columns on all $n-k$ positions and check the weight of the resulting vector.

Note that for the purpose of the operation count we assume that k is even. For uneven k without loss of generality let X have size $\lfloor k/2 \rfloor$ and Y size $\lceil k/2 \rceil$.

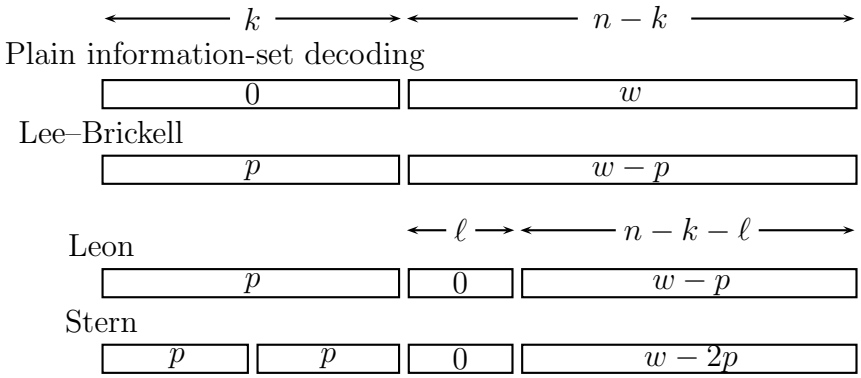

Figure 5.2: Distribution of error positions of information-set-decoding algorithms; this visual comparison is copied from Overbeck and Sendrier's survey [OS08].

5.2.1 Speeding up Gaussian elimination

The following improvements deal with the first step of Stern's algorithm, i.e., with updating the parity-check matrix at the beginning of each iteration with respect to a newly chosen information set. A selection of an information set is in fact a selection of $n-k$ columns of H as described in Remark 4.6.

Remark 5.6 (Reusing existing pivots)**.** Each iteration of Stern's algorithm selects $n-k$ columns of the parity-check matrix H and applies row operations — Gaussian elimination — to reduce those columns to the $(n-k) \times (n-k)$ identity matrix.
Any parity-check matrix for the same code will produce the same results here. In particular, instead of starting from the originally supplied parity-check matrix, we start from the parity-check matrix UH produced in the previous iteration — which, by construction, already has an $(n-k)\times(n-k)$ identity submatrix. About $(n-k)^2/n$ of the newly selected columns will match previously selected columns, and are simply permuted into identity form with minimal effort, leaving real work for only about $n-k-(n-k)^2/n = (k/n)(n-k)$ of the columns.
Stern states that reduction involves about $(1/2)(n-k)^3+k(n-k)^2$ bit operations; for example, $(3/16)n^3$ bit operations for $k = n/2$. To understand this formula, observe that the first column requires $\leq n-k$ reductions, each involving $\leq n-1$ additions (mod 2); the second column requires $\leq n-k$ reductions, each involving $\leq n-2$ additions; and so on through the $(n-k)$th column, which requires $\leq n-k$ reductions, each involving $\leq k$ additions; for a total of $(1/2)(n-k)^3 + (k-1/2)(n-k)^2$.
We improve the bit-operation count to $k^2(n-k)(n-k-1)(3n-k)/4n^2$: for example, $(5/128)n^2(n-2)$ for $k = n/2$. Part of the improvement is from eliminating the work for the first $(n-k)^2/n$ columns. The other part is the standard observation that the number of reductions in a typical column is only about $(n-k-1)/2$.

The *bit-swapping technique* is a way to reduce Gaussian-elimination cost by changing only one information-set element in each iteration. This idea was introduced by Omura, according to [CC81, Section 3.2.4]. It was applied to increasingly optimized forms of information-set decoding by van Tilburg in [vT90] and [vT94], by Chabanne and Courteau in [CC93], by Chabaud in [Cha93], by Canteaut and Chabanne in [CC94], by Canteaut and Chabaud in [CC98], and by Canteaut and Sendrier in [CS98].
This section considers a generalization of bit swapping devised together with Bernstein and Lange in [BLP08] which improves the balance between the cost of Gaussian elimination and the cost of error-searching as will be explained in the following.

Remark 5.7 (Force more existing pivots)**.** Start with the matrix UH from the previous iteration which contains an $(n-k) \times (n-k)$ identity matrix. Instead of finding a new set of $n-k$ columns which need to be row-reduced — or equivalently finding a new invertible matrix U — one can artificially reuse exactly $n-k-c$ column selections, and select the remaining c new columns randomly from among the other k columns, where c is a new algorithm parameter.[3] Then only c columns need to be newly pivoted. Reducing c below $(k/n)(n-k)$ saves time correspondingly.
Beware, however, that smaller values of c introduce a dependence between iterations and require more iterations before the algorithm finds the desired weight-w word. See Section 5.2.3 for a detailed discussion of this effect.

Illustrative example from the literature: Canteaut and Sendrier report in [CS98, Table 2] that they need $9.85 \cdot 10^{11}$ iterations to handle $n = 1024$, $k = 525$, $w = 50$ with their best parameters $(p, \ell) = (2, 18)$. Stern's algorithm, with the same $(p, \ell) = (2, 18)$, needs only $5.78 \cdot 10^{11}$ iterations. Note that these are not the best parameters for Stern's algorithm; the parameters $p = 3$ and $\ell = 28$ achieve better results, needing only $1.85 \cdot 10^{10}$ iterations. The total cost for attacking $n = 1024$, $k = 525$, $w = 50$ with $(p, \ell) = (3, 28)$ — crudely estimated as in Stern's article — without any improvements from [BLP08] amounts to $2^{62.81}$ bit operations; already considerably less than Canteaut et al.'s $2^{64.1}$ bit operations with $(p, \ell) = (2, 18)$.
Another illustrative example: Canteaut and Chabaud recommend $(p, \ell) = (2, 20)$ for $n = 2048$, $k = 1025$, $w = 112$ in [CC98, Table 2], These parameters use $5.067 \cdot 10^{29}$ iterations, whereas Stern's algorithm with the same parameters uses $3.754 \cdot 10^{29}$ iterations.
Canteaut and Chabaud claim that Gaussian elimination is the "most expensive step" in previous attacks, justifying the switch to $c = 1$. We point out, however, that this switch often loses speed compared to Stern's original attack; in particular when aiming at parameter sizes which are interesting for cryptography. For example, Stern's original attack (without reuse of existing pivots) uses only $2^{124.06}$ bit operations for $n = 2048$, $k = 1025$, $w = 112$ with $(p, \ell) = (3, 31)$, beating the algorithm by Canteaut et al.; in this case Gaussian elimination is only 22% of the cost of each iteration.

[3] This section uses the letter c to denote an integer. In this section c does not denote a codeword as in other parts of the thesis.

Both $c = 1$, as used by Canteaut et al., and $c = (k/n)(n-k)$, as used (essentially) by Stern, are beaten by intermediate values of c. See Section 5.2.3 for some examples of optimized choices of c.
Adding the first selected row to various other rows cancels all remaining 1's in the first selected column. Adding the second selected row to various other rows then cancels all remaining 1's in the second selected column.
It has frequently been observed — see, e.g., [Bar06] — that there is an overlap of work in these additions: about 25% of the rows will have *both* the first row and the second row added. One can save half of the work in these rows by simply precomputing the sum of the first row and the second row. The precomputation involves at most one vector addition (and is free if the first selected column originally began $1, 1$). This observation leads to the following speedup for Stern's algorithm.

Remark 5.8 (Faster pivoting)**.** Suppose that we defer additions of r rows; here r is another algorithm parameter which needs to be tuned with respect to n and in particular to the number of rows $n - k$ and also $r \leq c$. After precomputing all $2^r - 1$ sums of nonempty subsets of these rows, we can handle each remaining row with, on average, $1 - 1/2^r$ vector additions, rather than $r/2$ vector additions. For example, after precomputing 15 sums of nonempty subsets of 4 rows, we can handle each remaining row with, on average, 0.9375 vector additions, rather than 2 vector additions; the precomputation in this case uses at most 11 vector additions. The optimal choice of r is roughly $\log_2(n-k) - \log_2 \log_2(n-k)$ but interacts with the optimal choice of c.

See [Pip79] for a much more thorough optimization of subset-sum computations.

Remark 5.9 (Multiple choices of Z)**.** Recall that one iteration of Stern's algorithm finds a particular weight-w word if that word has exactly $p, p, 0$ errors in the column sets X, Y, Z respectively. We generalize Stern's algorithm to allow m disjoint sets $Z_1, Z_2, \ldots, Z_m$ with the same X, Y, each of $Z_1, Z_2, \ldots, Z_m$ having cardinality ℓ; here $m \geq 1$ is another algorithm parameter.
The cost of this generalization is an m-fold increase in the time spent in the second and third steps of the algorithm — but the first step, the initial Gaussian elimination, depends only on X, Y and is done only once. The benefit of this generalization is that the chance of finding any particular weight-w word grows by a factor of nearly m.
For example, if $(n, k, w) = (1024, 525, 50)$ and $(p, \ell) = (3, 29)$, then one set Z_1 works with probability approximately 6.336%, while two disjoint sets Z_1, Z_2 work with probability approximately 12.338%. Switching from one set to two produces a $1.947\times$ increase in effectiveness at the expense of replacing steps (I), (II), (III) by steps (I), (II), (III), (II), (III). This is worthwhile if step (I), Gaussian elimination, is more than about 5% of the original computation. Note that the Roman numerals denote the steps in the high-level description at the beginning of this section.

The improvements in this section and the next section led to the break of the original McEliece parameters which will be discussed in Section 5.2.3. For attacking larger

parameter sets, in particular, as the code length increases the cost for Gaussian elimination becomes negligible; this will be shown in the asymptotic analysis of information-set decoding in Section 5.4.
The following section discusses two more improvements we suggested in [BLP08] which prove to be more significant on the long term.

5.2.2 Speeding up list building and collision handling

The "list-building step" of Stern's algorithm considers all p-element subsets A of X and all p-element subsets B of Y, and computes ℓ-bit sums $\phi(A), \psi(B)$. Stern [Ste89] states that this takes $2\ell p\binom{k/2}{p}$ bit operations for average-size X, Y. Similarly, Canteaut et al. [CC98, CS98] mention that there are $\binom{k/2}{p}$ choices of A and $\binom{k/2}{p}$ choices of B, each using $p\ell$ bit operations.
We comment that, although computing $\phi(A)$ means $p-1$ additions of ℓ-bit vectors, usually $p-2$ of those additions were carried out before. Simple caching, i.e., using "intermediate sums", reduces the average cost of computing $\phi(A)$ to only marginally more than ℓ bit operations for each A. In more detail:

Remark 5.10 (Reusing additions of the ℓ-bit vectors)**.** Computing a vector $\phi(A)$ which is the sum of p columns indexed by A restricted to ℓ positions can be done by adding the specified p columns in $p-1$ additions in $\mathbb{F}_2^\ell$. Computing vectors $\phi(A)$ for *all* the $\binom{k/2}{p}$ possible size-p subsets can be done more efficiently than repeating this process for each of them. Start by computing all $\binom{k/2}{2}$ sums of 2 columns indexed by I; each sum costs one addition in $\mathbb{F}_2^\ell$. Then compute all $\binom{k/2}{3}$ sums of 3 columns by adding one extra column to the previous results; again restricted to ℓ positions indexed by Z. Proceed in the same way until all $\binom{k/2}{p}$ sums of p columns on ℓ positions are computed. This produces all required sums in only marginally more than one $\mathbb{F}_2^\ell$ addition per sum.

The total cost amounts to

$$2\ell\left(\binom{k/2}{2}+\binom{k/2}{3}+\cdots\binom{k/2}{p}\right).$$

These costs can be written as $2\ell\,(L(k/2,p)-k/2)$, using $L(k,p)=\sum_{i=1}^{p}\binom{k}{i}$ as a shorthand.
This improvement becomes increasingly important as p grows as it saves a factor p in comparison to Stern's as well as to Canteaut et al.'s methods.
The last improvement concerns the "collision handling" in Stern's algorithm. For the pairs (A,B) with $\phi(A)=\psi(B)$ one has to add all the columns indexed by $A\cup B$ and then check their weight. If the vectors $\phi(A)$ and $\psi(B)$ were uniformly distributed among the 2^ℓ possible values then on average $2^{-\ell}\binom{k/2}{p}^2$ checks would need to be done. The expected number of checks is extremely close to this for almost all matrices H; the extremely rare codes with different behavior are disregarded for the purpose of this analysis.

Remark 5.11 (Faster additions after collisions)**.** Naive computation of the sum of $2p$ columns in the third step would take $2p-1$ additions on $n-k-\ell$ positions; ℓ positions are known to be zero as a consequence of the collision. However, as pointed out in [BLP08, Section 4], many of these additions overlap. The algorithm hopes that the sum of $2p$ columns has weight $w-2p$, i.e., that exactly $w-2p$ of $n-k$ entries equal 1. Each entry has a chance of $1/2$ to be 1. In order to save operations, one computes the result in a row-by-row fashion and uses an early abort: after about $2(w-2p+1)$ rows are handled it is very likely that the resulting column vector has more than the allowed $w-2p$ nonzero entries and can be discarded. This means that partial collisions that do not lead to a full collision consume only $2(w-2p+1)$ operations.

Remark 5.12. As mentioned before the "list-building step" and "collision handling" are the most costly parts of Stern's algorithm. The bottleneck of the algorithm is the number of possible p-sums coming from columns indexed by X and Y, respectively. In order to keep this number manageable p is chosen to be quite small. In order to balance the cost of "list-building step" and "collision handling" one can choose ℓ as $\log_2 \binom{k/2}{p}$. This is good as a rule of thumb but we nevertheless recommend using the parameter optimization which will be presented in the next section and which depends on the expected number of iterations.

5.2.3 Analysis of the number of iterations and comparison

Canteaut, Chabaud, and Sendrier [CC98, CS98] announced thirteen years ago that the original parameters for McEliece's cryptosystem were not acceptably secure: specifically, an attacker can decode 50 errors in a $[1024, 524]$ code over $\mathbb{F}_2$ in $2^{64.1}$ bit operations.

Using the improvements from the previous two sections with parameters $p = 2$, $m = 2$, $\ell = 20$, $c = 7$, and $r = 7$ the same computation can be done in only $2^{60.55}$ bit operations, almost a $12\times$ improvement over Canteaut et al. The number of iterations drops from $9.85 \cdot 10^{11}$ to $4.21 \cdot 10^{11}$, and the number of bit operations per iteration drops from $20 \cdot 10^6$ to $4 \cdot 10^6$. As discussed in Section 5.2.4, Bernstein has achieved even larger speedups in software.

The rest of this section explains how the number of iterations used by the attack is computed, and then presents similar results for many more sizes $[n, k]$.

The parameter optimization in [BLP08] relies on being able to quickly and accurately compute the average number of iterations required for the attack.

It is easy to understand the success chance of *one* iteration of the attack:

- Let I be an information set chosen uniformly at random. A random weight-w word e has weight $2p$ among the columns indexed by I with probability $\binom{k}{2p}\binom{n-k}{w-2p} \Big/ \binom{n}{w}$. The actual selection of columns is adaptive and thus not exactly uniform, but as mentioned before this bias appears to be negligible; we have tried many attacks with small w and found no significant deviation from uniformity.

- Let X, Y be disjoint size-$(k/2)$ subsets of I chosen uniformly at random. The conditional probability of the $2p$ errors of I-indexed positions in e appearing as p errors among the positions indexed by X and p errors among the positions indexed by Y is given by $\binom{k/2}{p}^2 \Big/ \binom{k}{2p}$.

- Let Z be a size-ℓ subset in $\{1, \ldots, n\} \setminus I$ chosen uniformly at random. The conditional probability of $w - 2p$ errors avoiding Z, a uniform random selection of ℓ out of the remaining $n - k$ columns, is $\binom{n-k-(w-2p)}{\ell} / \binom{n-k}{\ell}$. As discussed in Section 5.2.1, we increase this chance by allowing disjoint sets $Z_1, Z_2, \ldots, Z_m$; the conditional probability of $w - 2p$ errors avoiding at least one of $Z_1, Z_2, \ldots, Z_m$ is

$$m\frac{\binom{n-k-(w-2p)}{\ell}}{\binom{n-k}{\ell}} - \binom{m}{2}\frac{\binom{n-k-(w-2p)}{2\ell}}{\binom{n-k}{2\ell}} + \binom{m}{3}\frac{\binom{n-k-(w-2p)}{3\ell}}{\binom{n-k}{3\ell}} - \cdots$$

  by the inclusion-exclusion principle.

The product of these probabilities is the chance that Stern's algorithm finds e after the *first* round.
For $m = 1$ the chance to find e in the first round equals

$$\mathrm{STPr}(n, k, w, \ell, p) = \frac{\binom{k/2}{p}^2 \binom{n-k-\ell}{w-2p}}{\binom{n}{w}}.$$

Remark 5.13. If iterations were independent, as in Stern's original attack, then the average number of iterations would be simply the reciprocal of the product of the probabilities. But iterations are not, in fact, independent. The difficulty is that the number of errors in the selected $n - k$ columns is correlated with the number of errors in the columns selected in the next iteration. This is most obvious with the bit-swapping technique as used by van Tilburg for the Lee–Brickell algorithm [vT90, vT94] and by Canteaut et al. [CC98, CS98] for collision decoding (here the case $c = 1$). Swapping one selected column for one deselected column is quite likely to preserve the number of errors in the selected columns and in any case cannot change it by more than one. The effect decreases in magnitude as c increases, but iterations also become slower as c increases; optimal selection of c requires understanding how c affects the number of iterations.

To analyze the impact of c we compute a Markov chain for the number of errors. generalizing the analysis of Canteaut et al. from $c = 1$ to arbitrary c. Here are the states of the chain:

- 0: There are 0 errors in the deselected k columns.

- 1: There is 1 error in the deselected k columns.

- ...
- w: There are w errors in the deselected k columns.
- Done: The attack has succeeded.

An iteration of the attack moves between states as follows. Starting from state u, the attack replaces c selected columns, moving to states $u-c, \ldots, u-2, u-1, u, u+1, u+2, \ldots, u+c$ with various probabilities discussed below. The attack then checks for success, moving from state $2p$ to state Done with probability

$$\frac{\binom{\lfloor k/2 \rfloor}{p}\binom{\lceil k/2 \rceil}{p}}{\binom{k}{2p}}\left(m\frac{\binom{n-k-(w-2p)}{\ell}}{\binom{n-k}{\ell}} - \binom{m}{2}\frac{\binom{n-k-(w-2p)}{2\ell}}{\binom{n-k}{2\ell}} + \cdots\right)$$

and otherwise staying in the same state.
For $c = 1$, the column-replacement transition probabilities are mentioned by Canteaut et al.:

- state u moves to state $u-1$ with probability $u(n-k-(w-u))/(k(n-k))$;
- state u moves to state $u+1$ with probability $(k-u)(w-u)/(k(n-k))$;
- state u stays in state u otherwise.

For $c > 1$, there are at least three different interpretations of "select c new columns":

- "Type 1": Choose a selected column; choose a non-selected column; swap. Continue in this way for a total of c swaps.
- "Type 2": Choose c distinct selected columns. Swap the first of these with a random non-selected column. Swap the second with a random non-selected column. Etc.
- "Type 3": Choose c distinct selected columns and c distinct non-selected columns. Swap the first selected column with the first non-selected column. Swap the second with the second. Etc.

Type 1 is the closest to Canteaut et al.: its transition matrix among states $0, 1, \ldots, w$ is simply the cth power of the matrix for $c = 1$. On the other hand, type 1 has the highest chance of re-selecting a column and thus ending up with fewer than c new columns; this effectively decreases c. Type 2 reduces this chance, and type 3 eliminates this chance.
The type-3 transition matrix has a simple description: state u moves to state $u+d$ with probability

$$\binom{n-k}{c}^{-1}\binom{k}{c}^{-1}\sum_i \binom{w-u}{i}\binom{n-k-w+u}{c-i}\binom{u}{d+i}\binom{k-u}{c-d-i}.$$

For $c = 1$ this matrix matches the Canteaut-et-al. matrix.
We have implemented[4] the type-1 Markov analysis and the type-3 Markov analysis. To save time we use floating-point computations with a few hundred bits of precision rather than exact rational computations. We use the MPFI library [RR] (on top of the MPFR library [HLP+] on top of GMP [Gra]) to compute intervals around each floating-point number, guaranteeing that rounding errors do not affect our final results.
As a check we have also performed millions of type-1, type-2, and type-3 simulations and millions of real experiments decoding small numbers of errors. The simulation results are consistent with the experimental results. The type-1 and type-3 simulation results are consistent with the predictions from our Markov-chain software. Type 1 is slightly slower than type 3, and type 2 is intermediate. Bernstein's attack software uses type 3. The graphs below also use type 3.

Remark 5.14 (Choosing attack parameters)**.** For each (n, t) in a wide range, we have explored parameters for the new attack and set new records for the number of bit operations needed to decode t errors in an $[n, n-t\lceil\log_2 n\rceil]$ code. Figure 5.3 shows the new records. Note that the optimal attack parameters (p, m, ℓ, c, r) depend on n, and depend on t for fixed n. In particular, the Markov-chain implementation is useful to find optimal attack parameters (p, m, ℓ, c, r) for any input (n, k).

5.2.4 Breaking the original McEliece parameters

Bernstein implemented an attack based on the improvements described in the sections before. Using his software we extracted a plaintext from a ciphertext by decoding 50 errors in a code of length 1024 and dimension 524 over $\mathbb{F}_2$.
If we were running the attack software on a single computer with a 2.4GHz Intel Core 2 Quad Q6600 CPU then we would need, on average, approximately 1400 days (2^{58} CPU cycles) to complete the attack. Running the software on 200 such computers — a moderate-size cluster costing under $200000 — would reduce the average time to one week. Since iterations are independent no communication is needed between the computers.
These attack speeds are much faster than the best speeds reported in the previous literature. Specifically, Canteaut, Chabaud, and Sendrier in [CC98] and [CS98] report implementation results for a 433MHz DEC Alpha CPU and conclude that one such computer would need approximately 7400000 days (2^{68} CPU cycles): "decrypting one message out of 10,000 requires 2 months and 14 days with 10 such computers."
Of course, the dramatic reduction from 7400000 days to 1400 days can be partially explained by hardware improvements — the Intel Core 2 Quad runs at 5.54× the clock speed of the Alpha 21164, has four parallel cores (compared to one), and can perform three arithmetic instructions per cycle in each core (compared to two). But these hardware improvements alone would only reduce 7400000 days to 220000 days.

[4]The software can be found at `http://www.win.tue.nl/~cpeters/mceliece.html`

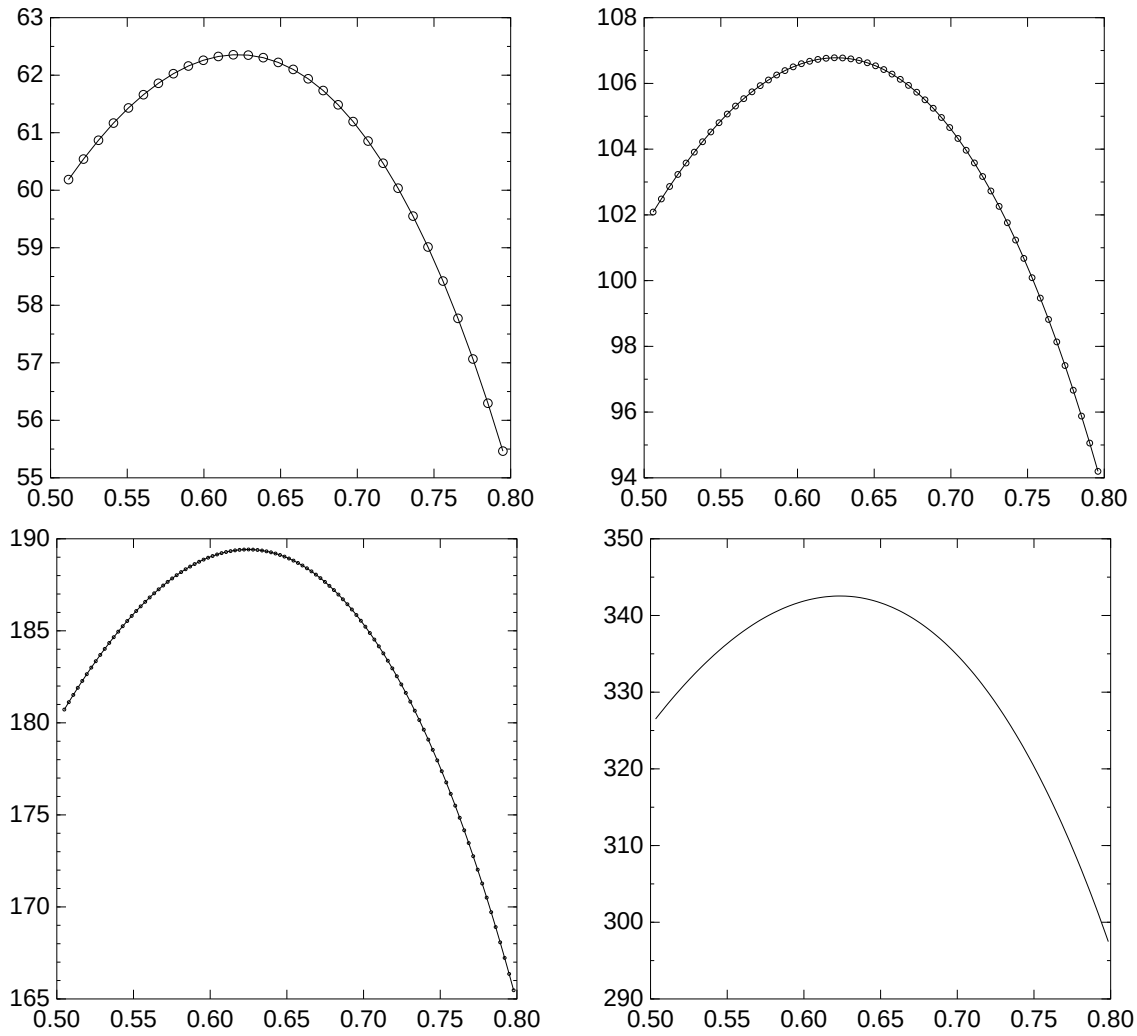

Figure 5.3: Attack costs for $n = 1024$, $n = 2048$, $n = 4096$, $n = 8192$. Horizontal axis is the code rate $(n - t\lceil \log_2 n \rceil)/n$. Vertical axis is $\log_2$(bit operations).

The remaining speedup factor of 150, allowing us to carry out the first successful attack on the original McEliece parameters, comes from the improvements of the attack itself.

We ran a distributed attack which involved about 200 computers, with about 300 cores. We gratefully acknowledge contributions of CPU time from the following sources.

- The Coding and Cryptography Computer Cluster at Technische Universiteit Eindhoven (TU/e);
- the FACS cluster at the Centrum Wiskunde & Informatica (CWI) in Amsterdam;
- the Walton cluster at SFI/HEA Irish Centre for High-End Computing;

- the Department of Electrical Engineering at National Taiwan University;
- the Courbes, Algèbre, Calculs, Arithmétique des Ordinateurs (CACAO) cluster at Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA)[5];
- the System Architecture and Networking Distributed and Parallel Integrated Terminal (sandpit) at TU/e;
- the Argo cluster at the Academic Computing and Communications Center at the University of Illinois at Chicago (UIC);
- the Center for Research and Instruction in Technologies for Electronic Security (RITES) at UIC;
- and private cores owned by Daniel J. Bernstein and Tanja Lange.

The computation finished in under 90 days (starting in July 2008, ending at the beginning of October 2008) and used about 8000 core-days. Most of the cores put in far fewer than 90 days of work; some of the CPUs were considerably slower than a Core 2. The error vector was found by the Walton cluster at SFI/HEA Irish Centre of High-End Computing (ICHEC).

Remark 5.15 (Number of iterations)**.** The Canteaut-et-al. attack uses $9.85 \cdot 10^{11}$ iterations on average, with (in the notation of this section) $p = 2$, $\ell = 18$, $m = 1$, and $c = 1$.
To avoid excessive time spent handling collisions in the main loop, we increased ℓ from 18 to 20. This increased the number of iterations to $11.14 \cdot 10^{11}$.
We then increased m from 1 to 5: for each selection of column sets X, Y we try five sets Z_1, Z_2, Z_3, Z_4, Z_5. We further increased c from 1 to 32: each iteration replaces 32 columns from the previous iteration. These choices increased various parts of the per-iteration time by factors of 5 and (almost) 32 respectively; the combined changes reduced the number of iterations by a factor of more than 6, down to $1.85 \cdot 10^{11}$.
In particular, the new parameters $m = 2$, $c = 12$ should take only 5000 core-days on average. But having reached feasibility we decided to proceed with the attack.

5.2.5 Defending the McEliece cryptosystem

This section proposes new parameters for the McEliece cryptosystem with binary Goppa codes.

Remark 5.16 (Increasing n)**.** The most obvious way to defend McEliece's cryptosystem is to increase n, the length of the code used in the cryptosystem. McEliece proposed in [McE78] to use binary Goppa codes whose length is a power of 2, making the length as large as possible. Like Sendrier [Sen02, page 151] we recommend values

[5]Since January 2010 CACAO has been superseded by the CARAMEL project.

of n between powers of 2 which allows considerably better optimization of (e.g.) the McEliece/Niederreiter public-key size. See below for examples. Aside from a mild growth in decoding time, there is no obstacle to the key generator using a Goppa code defined via a field $\mathbb{F}_{2^m}$ of size *much* larger than n.

Remark 5.17 (Using list decoding to increase w)**.** As discussed in Remark 4.18 Bernstein devised a list-decoding algorithm for classical irreducible binary Goppa codes, exactly the codes used in McEliece's cryptosystem. This algorithm allows the receiver to efficiently decode approximately $n-\sqrt{n(n-2t-2)} \geq t+1$ errors instead of t errors. The sender, knowing this, can introduce correspondingly more errors; the attacker is then faced with a more difficult problem of decoding the additional errors.
List decoding can, and occasionally does, return more than one codeword within the specified distance. In CCA2-secure variants of McEliece's system there is no difficulty in identifying which codeword is a valid message. Our attack can, in exactly the same way, easily discard codewords that do not correspond to valid messages.

We propose concrete parameters $[n, k]$ for various security levels in CCA2-secure variants of the McEliece cryptosystem. Recall that public keys in these variants are systematic generator matrices occupying $k(n-k)$ bits.

Remark 5.18 (80-bit security)**.** For (just barely!) 80-bit security against our attack we propose $[1632, 1269]$ Goppa codes (degree $t = 33$), with 34 errors added by the sender. The public-key size here is $1269(1632 - 1269) = 460647$ bits.
Without list decoding, and with the traditional restriction that the code length n is a power of 2, the best possibility is $[2048, 1751]$ Goppa codes ($t = 27$). The public key here is considerably larger, namely 520047 bits.

Remark 5.19 (128-bit security)**.** For 128-bit security we propose $[2960, 2288]$ Goppa codes ($t = 56$), with 57 errors added by the sender. The public-key size here is 1537536 bits.

Remark 5.20 (256-bit security)**.** For 256-bit security we propose $[6624, 5129]$ Goppa codes ($t = 115$), with 117 errors added by the sender. The public-key size here is 7667855 bits.

Remark 5.21 (Small keys)**.** For keys limited to 2^{16}, 2^{17}, 2^{18}, 2^{19}, 2^{20} bytes, we propose Goppa codes of lengths 1744, 2480, 3408, 4624, 6960 and degrees 35, 45, 67, 95, 119 respectively, with 36, 46, 68, 97, 121 errors added by the sender. These codes achieve security levels 84.88, 107.41, 147.94, 191.18, 266.94 against our attack. In general, for any particular limit on public-key size, codes of rate approximately 0.75 appear to maximize the difficulty of our attack.

5.3 Collision decoding: recent developments

Article [BLP08] triggered a series of papers on improved versions of Stern's algorithm.

5.3.1 Fixed-distance decoding

To the best of our knowledge all collision-decoding algorithms used the reduction to low-weight-word finding which was explained in Section 4.2.4, in particular in Remark 4.24. The article "Explicit bounds for generic decoding algorithms for code-based cryptography" [BLPvT09] which is joint work with Bernstein, Lange, and van Tilborg gives an *explicit* description of Stern's algorithm as a fixed-distance-decoding algorithm. Since then many improvements on information-set decoding have been described in this way; see e.g., [FS09], [Pet10], [BLP10].
Section 5.1.2 already described Stern's algorithm as a fixed-distance-decoding algorithm. However, the analysis of the improvements to Stern's algorithm did not take this change into account; Chapter 6 will point out the subtle differences between Stern's algorithm with input $s = 0$ and with arbitrary s, respectively.
Chapter 6 describes Stern's algorithm as a fixed-distance-decoding algorithm for decoding w errors in a linear code over $\mathbb{F}_q$; moreover, the generalized Stern algorithm is described in terms of the generator matrix. For $q = 2$ the algorithm is the same as the one in [BLPvT09, Section 2]. Chapter 8 describes "ball-collision decoding" as a fixed-distance-decoding algorithm; it contains collision decoding as a special case and uses the parity-check matrix.

5.3.2 The birthday speedup

Finiasz and Sendrier in [FS09] presented a further improvement to Stern's algorithm, the "birthday speedup".
Stern splits an information set I into two disjoint sets X and Y, each of size $\binom{k/2}{p}$ (assuming k is even) and searches for collisions among size-p subsets taken from X and Y. Finiasz and Sendrier propose not to split the information set I into two disjoint sets but to look more generally for collisions. The split of I into two disjoint size-$(k/2)$ sets is omitted at the benefit of creating more possible words having weight $2p$ among the information set.
The Stern algorithm changes as follows: introduce a new parameter N with $0 \leq N \leq \binom{k}{p}$, *remove* Steps 2–3, and *replace* Steps 5–6 in Algorithm 5.2 by the following two steps.

5: **Repeat** N times: choose a uniform random size-p set $A \subseteq I$ and consider the p columns $(UH)_a$ indexed by $a \in A$; compute $\phi(A) = s - \sum_a (UH)_a$ restricted to ℓ rows indexed by Z.
6: **Repeat** N times: choose a uniform random size-p set $B \subseteq I$ and consider the p columns $(UH)_b$ indexed by $b \in B$ and compute $\psi(B) = \sum_b (UH)_b$ restricted to ℓ rows indexed by Z.

Note that Steps 1, 4, and 7–10 stay the same.

Remark 5.22. The Stern algorithm with "birthday speedup" finds a weight-w vector e with $He^t = s$ if it finds an information set I and size-p subsets $A, B \subseteq I$ such

that e has weight $2p$ on positions indexed by $A \cup B$ and weight 0 on the positions indexed by Z.

It is possible that a subset chosen in Step 5 is also chosen in Step 6. This case is allowed in order to benefit from a larger set of possible sums of p rows.
In [FS09] Finiasz and Sendrier give a *bound* for the cost of collision decoding with birthday speedup:

$$\min_p \frac{2\ell \min\left(\binom{n}{w}, 2^{n-k}\right)}{\lambda \binom{n-k-\ell}{w-p} \sqrt{\binom{k+\ell}{p}}}, \tag{5.1}$$

where $\lambda = 1 - \exp(-1)$. They conclude that the asymptotic speedup factor compared to classical collision decoding with parameter p is $\Theta(p^{1/4})$; see [FS09, page 95]. Finiasz and Sendrier did not analyze the complexity of the birthday speedup together with all speedups discussed in Section 5.2. The q-ary version of collision decoding in Chapter 6 contains a detailed analysis; it covers $q = 2$ as a special case.
In practice the size of subsets N will be bigger than $\binom{k/2}{p}$ so that the sets X and Y overlap. As there are more possible p-sums arising from X and Y Steps 5–6 are a little more expensive. However, also here one can apply the improvement from Section 5.2.2 (speeding up "list building" by intermediate sums).
Note that for simplicity of the analysis this section speaks of two sets X and Y indexing possible p-sums. Another way of describing the birthday speedup is to construct one single set of p-sums indexed by entries in the information set and search for collisions within the set. The important point is that one should not try to build all possible sums coming from p columns in the information set.

Remark 5.23 (Success probability of the first iteration)**.** There are $\binom{2p}{p}$ different possibilities of splitting $2p$ errors into two disjoint subsets of cardinality p each. The probability of not finding an error vector e, which has $2p$ errors in I, by a fixed size-p set A and a fixed size-p set B is $1 - \binom{2p}{p} / \binom{k}{p}^2$. If one chooses N sets A and N sets B uniformly at random the probability of e not being found by any pair (A, B) equals $\left(1 - \binom{2p}{p} \Big/ \binom{k}{p}^2\right)^{N^2} \approx \exp\left(-N^2 \binom{2p}{p} \Big/ \binom{k}{p}^2\right)$.
The probability of the first iteration to succeed is thus

$$\binom{n}{w}^{-1} \binom{k}{2p} \binom{n-k-\ell}{w-2p} \left(1 - \left(1 - \binom{2p}{p} \binom{k}{p}^{-2}\right)^{N^2}\right).$$

Remark 5.24 (Choice of parameters)**.** As in Stern's algorithm the parameter p is chosen to be a small number. The parameter ℓ is chosen to balance $2N$, the number of all computed length-ℓ vectors $\phi(A)$ and $\psi(B)$, with the number of expected collisions on ℓ positions which is $N^2/2^\ell$. Similarly to Remark 5.12 let $\ell = \log_2 N$.
This algorithm works for any numbers N less than or equal to $\binom{k}{p}$, the number of all possible size-p subsets taken from an information set I. There is no point in choosing N larger than this number since otherwise all possible combinations of p elements out of I could be deterministically tested. A sensible choice for N is $\binom{k}{p} \Big/ \sqrt{\binom{2p}{p}}$.

5.4 Decoding complexity comparison

The break of the original McEliece parameters described in Section 5.2 was quite costly, and it is not clear how well the modifications to Stern's algorithm presented in Section 5.2 scale to other sizes. Extrapolation from examples is dangerous: for example, the Canteaut–Chabaud decoding algorithm of [CC98] was claimed to be an improvement over [Ste89], but was shown in Section 5.2 to be worse than [Ste89] for large code lengths, including 1024.

One mathematically pleasing way to measure the scalability of an algorithm is to compute its asymptotic cost exponent as a function of the code rate R and the error fraction W. This section analyzes the asymptotic cost of information-set decoding. In particular, the cost of the Lee–Brickell algorithm and collision decoding are analyzed.

The results presented here are based on the article "Explicit bounds for generic decoding algorithms for code-based cryptography" [BLPvT09] which is joint work with Bernstein, Lange, and van Tilborg.

5.4.1 Motivation and background

We consider binary linear codes of length n, code rate $R = k/n$, and error fraction $W = w/n$. In order to compare the asymptotic cost of information-set-decoding algorithms define the exponent $\alpha(R, W)$ as follows.

Remark 5.25. Let

$$\alpha(R,W) = (1-R-W)\log_2(1-R-W) - (1-R)\log_2(1-R) - (1-W)\log_2(1-W).$$

Then the simplest form of information-set decoding takes time $2^{(\alpha(R,W)+o(1))n}$ to find Wn errors in a dimension-Rn length-n binary code if R and W are fixed while $n \to \infty$. Unfortunately, the asymptotic formula $2^{(\alpha(R,W)+o(1))n}$ is too imprecise to see typical algorithm improvements. An $n\times$ speedup, for example, is quite valuable in practice but does not change the cost exponent.

Van Tilburg in his Ph.D. thesis [vT94, Section 7.4], discusses the "bounded-hard-decision-decoding" case $1 - R = H_2(2W)$ with $\alpha(R, W) = H_2(W) - (1 - R)H_2(\frac{W}{1-R})$ and the "bounded-soft-decision-decoding" case $1 - R = H_2(W)$ with $\alpha(R, W) = (1 - R)(1 - H_2(\frac{W}{1-R}))$; where H_2 is the binary entropy function. An algorithm that achieves a smaller cost exponent is obviously an improvement for sufficiently large n, if R and W are fixed.

The main objective of this section is to build a much more precise framework for the systematic analysis and comparison of information-set-decoding algorithms. We use the Θ-notation for giving results of the cost analysis and note that "$g(n) = \Theta(f(n))$" means that g is asymptotically bounded by a constant times f from above and also by a (different) constant times f from below.

The notation $\alpha(R, W)$ defined above is reused throughout this section, along with the notation $\beta(R, W) = \sqrt{(1 - R - W)/((1 - R)(1 - W))}$.

Recall that information-set-decoding algorithms search for certain error patterns; see Figure 5.2 in Section 5.1. The success probability of the first iteration of Lee–Brickell's algorithm and Stern's algorithm can be computed using combinatorics. As a model for the success probabilities take

- $\mathrm{LBPr}(n,k,w,p) = \binom{n}{w}^{-1}\binom{n-k}{w-p}\binom{k}{p}$ for the first iteration of the Lee–Brickell algorithm to succeed (see Section 5.1.1); and
- $\mathrm{STPr}(n,k,w,\ell,p) = \binom{n}{w}^{-1}\binom{k/2}{p}^2\binom{n-k-\ell}{w-2p}$ for the first iteration of the plain Stern algorithm to succeed (see Section 5.2.3).

We will put bounds on these probabilities by using the following bounds on the binomial coefficients.

Remark 5.26. Define $\epsilon : \{1,2,3,\ldots\} \to \mathbb{R}$ by the formula

$$m! = \sqrt{2\pi}\ m^{m+1/2}\ e^{-m+\epsilon(m)}. \tag{5.2}$$

The classic Stirling approximation is $\epsilon(m) \approx 0$. We will use a tighter bound on ϵ, namely

$$\frac{1}{12m+1} < \epsilon(m) < \frac{1}{12m}$$

which was proven by Robbins in [Rob55].

5.4.2 Asymptotic cost of the Lee–Brickell algorithm

This section introduces and analyzes $\mathrm{LBCost}(n,k,w,p)$, a model of the average time used by the Lee–Brickell algorithm. Define

$$\mathrm{LBCost}(n,k,w,p) = \left(\frac{1}{2}(n-k)^2(n+k) + \binom{k}{p}p(n-k)\right)\cdot \mathrm{LBPr}(n,k,w,p)^{-1},$$

where LBPr is the success probability of one iteration of the Lee–Brickell algorithm. The term $\frac{1}{2}(n-k)^2(n+k)$ is a model of row-reduction time, exactly as in [Ste89]; $\binom{k}{p}$ is the number of size-p subsets A of $\{1,2,\ldots,k\}$; and $p(n-k)$ is a model of the cost of computing $y - \sum_{a\in A} G_a$. Each vector G_a has n bits, but the k bits in columns corresponding to I can be skipped, since the sum of those columns is known to have weight p.

There can be many codewords at distance w from y if w is not below half the minimum distance of the code. In some applications, any of those codewords are acceptable, and a decoding algorithm can be stopped as soon as it finds a single codeword. One can use $\mathrm{LBCost}(n,k,w,p)/\#\{\text{distance-}w\text{ codewords}\}$ as a model of the cost of the Lee–Brickell algorithm in those applications.

We write the cost exponent of the Lee–Brickell algorithm as $2^{(\alpha(R,W)+o(1))n}$ where the $o(1)$ part is specified by means of an "error term" $\mathrm{LBErr}(n,k,w,p)$ which is defined as

$$\mathrm{LBErr}(n,k,w,p) = \frac{k!}{(k-p)!k^p}\frac{w!}{(w-p)!w^p}\frac{(n-k-w)!(n-k-w)^p}{(n-k-w+p)!}\frac{e^{\epsilon(n-k)+\epsilon(n-w)}}{e^{\epsilon(n-k-w)+\epsilon(n)}}.$$

The following analysis will show that this error term is close to 1 as n goes to infinity. The following lemma puts upper and lower bounds on $\mathrm{LBPr}(n,k,w,p)$. We assume that the code rate $R = k/n$ and error fraction $W = w/n$ satisfy $0 < W < 1-R < 1$.

Lemma 5.27. $\mathrm{LBPr}(n,k,w,p)$ *equals*

$$2^{-\alpha(R,W)n}\,\frac{1}{p!}\left(\frac{RWn}{1-R-W}\right)^p\frac{1}{\beta(R,W)}\,\mathrm{LBErr}(n,k,w,p).$$

Furthermore

$$\frac{(1-\frac{p}{k})^p(1-\frac{p}{w})^p}{(1+\frac{p}{n-k-w})^p}e^{-\frac{1}{12n}\left(1+\frac{1}{1-R-W}\right)} < \mathrm{LBErr}(n,k,w,p) < e^{\frac{1}{12n}\left(\frac{1}{1-R}+\frac{1}{1-W}\right)}.$$

Proof. Replace binomial coefficients by factorials, factor out the $p=0$ case, use the refined Stirling formula (5.2), and factor out LBErr:

$$\begin{aligned}
\mathrm{LBPr}(n,k,w,p) &= \frac{\binom{n-k}{w-p}\binom{k}{p}}{\binom{n}{w}} = \frac{(n-k)!}{(w-p)!(n-k-w+p)!}\frac{k!}{p!(k-p)!}\frac{w!(n-w)!}{n!}\\
&= \frac{(n-k)!(n-w)!}{(n-k-w)!n!}\cdot\frac{1}{p!}\frac{k!}{(k-p)!}\frac{w!}{(w-p)!}\frac{(n-k-w)!}{(n-k-w+p)!}\\
&= \frac{(n-k)^{n-k+1/2}(n-w)^{n-w+1/2}}{(n-k-w)^{n-k-w+1/2}n^{n+1/2}}\cdot\frac{e^{\epsilon(n-k)+\epsilon(n-w)}}{e^{\epsilon(n-k-w)+\epsilon(n)}}\\
&\quad\cdot\frac{1}{p!}\frac{k!}{(k-p)!}\frac{w!}{(w-p)!}\frac{(n-k-w)!}{(n-k-w+p)!}\\
&= \frac{(n-k)^{n-k+1/2}(n-w)^{n-w+1/2}}{(n-k-w)^{n-k-w+1/2}n^{n+1/2}}\frac{k^pw^p\,\mathrm{LBErr}(n,k,w,p)}{p!(n-k-w)^p}.
\end{aligned}$$

Substitute $k = Rn$, $w = Wn$, $(1-R)^{1-R}(1-W)^{1-W}/(1-R-W)^{1-R-W} = 2^{-\alpha(R,W)}$, and $(1-R-W)^{1/2}(1-R)^{-1/2}(1-W)^{-1/2} = \beta(R,W)$:

$$\begin{aligned}
\mathrm{LBPr}(n,k,w,p) &= \frac{(1-R)^{n-Rn+1/2}(1-W)^{n-Wn+1/2}}{(1-R-W)^{n-Rn-Wn+1/2}}\frac{(RWn^2)^p\,\mathrm{LBErr}(n,k,w,p)}{p!((1-R-W)n)^p}\\
&= 2^{-\alpha(R,W)n}\frac{1}{\beta(R,W)}\frac{(RWn)^p\,\mathrm{LBErr}(n,k,w,p)}{p!(1-R-W)^p}.
\end{aligned}$$

This is the desired equation for LBPr. Robbins's bounds

$$\begin{aligned}
&\epsilon(n-k-w) > 0,\\
&\epsilon(n) > 0,\\
&\epsilon(n-k) < 1/(12(n-k)) = 1/(12n(1-R)),\ \text{and}\\
&\epsilon(n-w) < 1/(12(n-w)) = 1/(12n(1-W)),
\end{aligned}$$

together with elementary bounds such as

$$k!/((k-p)!k^p) = k(k-1)\cdots(k-p+1)/k^p \le 1,$$

produce the stated upper bound on $\mathrm{LBErr}(n,k,w,p)$. The lower bound is derived in a similar way, using

$$k!/((k-p)!k^p) = k(k-1)\cdots(k-p+1)/k^p \ge (k-p)^p/k^p = (1-(p/k))^p.$$

□

Note that for fixed rate R, fixed error fraction W, and fixed p the error factor $\mathrm{LBErr}(n, Rn, Wn, p)$ is close to 1 as n tends to infinity.
The following corollaries analyze the asymptotics of $\mathrm{LBCost}(n,k,w,p)$ for p equal to 0, 1, 2, and 3. Recall that $p=0$ is the plain information-set decoding algorithm due to Prange [Pra62]. It turns out that $p=2$ is optimal.

Corollary 5.28. $\mathrm{LBCost}(n, Rn, Wn, 0) = (c_0 + O(1/n))2^{\alpha(R,W)n}n^3$ *as* $n \to \infty$ *where* $c_0 = (1/2)(1-R)(1-R^2)\beta(R,W)$.

Proof. $\mathrm{LBCost}(n, Rn, Wn, 0) = (1/2)(n-Rn)^2(n+Rn)/\mathrm{LBPr}(n, Rn, Wn, 0) = (1/2)(1-R)(1-R^2)n^3/\mathrm{LBPr}(n, Rn, Wn, 0)$.
By Lemma 5.27, $\mathrm{LBPr}(n, Rn, Wn, 0) = 2^{-\alpha(R,W)n}(1/\beta(R,W))\,\mathrm{LBErr}(n, Rn, Wn, 0)$, and the LBErr factor is $1 + O(1/n)$. □

Corollary 5.29. $\mathrm{LBCost}(n, Rn, Wn, 1) = (c_1 + O(1/n))2^{\alpha(R,W)n}n^2$ *as* $n \to \infty$ *where* $c_1 = (1/2)(1-R)(1-R^2)(1-R-W)(1/RW)\beta(R,W)$.

Proof. The numerator of $\mathrm{LBCost}(n, Rn, Wn, 1)$ is $(1/2)(n-Rn)^2(n+Rn) + Rn(n-Rn) = ((1/2)(1-R)(1-R^2) + R(1-R)/n)n^3$. The denominator $\mathrm{LBPr}(n, Rn, Wn, 1)$ is similar to the denominator $\mathrm{LBPr}(n, Rn, Wn, 0)$ analyzed above but has an extra factor $RWn/(1-R-W)$. □

Corollary 5.30. $\mathrm{LBCost}(n, Rn, Wn, 2) = (c_2 + O(1/n))2^{\alpha(R,W)n}n$ *as* $n \to \infty$ *where* $c_2 = (1-R)(1+R^2)(1-R-W)^2(1/RW)^2\beta(R,W)$.

Proof. The numerator $(1/2)(n-Rn)^2(n+Rn) + Rn(Rn-1)(n-Rn)$ has leading coefficient $(1/2)(1-R)^2(1+R) + R^2(1-R) = (1/2)(1-R)(1+R^2)$. The denominator $\mathrm{LBPr}(n, Rn, Wn, 2)$ is similar to $\mathrm{LBPr}(n, Rn, Wn, 1)$ but has an extra factor $RWn/(2(1-R-W))$. □

Corollary 5.31. $\mathrm{LBCost}(n, Rn, Wn, 3) = (c_3 + O(1/n))2^{\alpha(R,W)n}n$ *as* $n \to \infty$ *where* $c_3 = 3(1-R)(1-R-W)^3(1/W)^3\beta(R,W)$.

Proof. The numerator of $\mathrm{LBCost}(n, Rn, Wn, 2)$ is dominated by $(3/6)k^3(n-k) = (1/2)R^3(1-R)n^4$. The denominator is similar to $\mathrm{LBPr}(n, Rn, Wn, 2)$ but has an extra factor $RWn/(3(1-R-W))$. □

Comparing Corollary 5.28, Corollary 5.29, Corollary 5.30, and Corollary 5.31 shows that the cost ratio $\mathrm{LBCost}(n, Rn, Wn, p+1)/\mathrm{LBCost}(n, Rn, Wn, p)$ is approximately $(1-R-W)/(RWn)$ for $p=0$; $2(1+R^2)(1-R-W)/((1-R^2)RWn)$ for $p=1$; and $3R^2(1-R-W)/((1+R^2)W)$ for $p=2$. Note that for $p=0$ and $p=1$ the costs are dominated by row reduction, while for larger p the costs are dominated by the work of handling all size-p subsets of I. This is why the costs do not further decrease by factors of n for $p=2$ and beyond. Whether $p=3$ is better than $p=2$ depends the sizes of R and W; in particular, $p=2$ is best in the common case where W is small.
Note that the results are stated for arbitrary R and W. In the context of the McEliece cryptosystem W is a function of R, namely $W=(1-R)/\log_2 n$.
With this particular W one gets $\alpha(R,W)$ and $\beta(R,W)$ as

$$\begin{aligned}\alpha(R,(1-R)/\log_2 n) &= (1-R)(\log_2 n-1)/\log_2 n\big(\log_2(1-R)+\log_2(\log_2 n-1)\\ &\quad -\log_2\log_2 n\big)-(1-R)\log_2(1-R)\\ &\quad -(1-(1-R)/\log_2 n)\log_2(1-(1-R)/\log_2 n),\\ \beta(R,(1-R)/\log_2 n) &= \sqrt{(\log_2 n-1)/(\log_2 n-1+R)}.\end{aligned}$$

For an analysis of $\alpha(R,(1-R)/\log_2 n)$ and in particular LBCost for typical McEliece rates and error fractions see Section 5.4.4.

5.4.3 Complexity of Stern's algorithm

This section shows that p has a much larger impact on the performance in Stern's algorithm and in particular it shows that p grows with n. The optimal ℓ is approximately $\log_2 \binom{k/2}{p}$.
We introduce and analyze $\mathrm{STCost}(n,k,w,\ell,p)$, a model of the average time used by Stern's algorithm. Define

$$\mathrm{STCost}(n,k,w,\ell,p)=\frac{\frac{1}{2}(n-k)^2(n+k)+2\binom{k/2}{p}p\ell+2\binom{k/2}{p}^2p(n-k)/2^\ell}{\mathrm{STPr}(n,k,w,\ell,p)},$$

where STPr is the success probability of Stern's algorithm as defined above. As in LBCost the term $\frac{1}{2}(n-k)^2(n+k)$ is a model of row-reduction time; $\binom{k/2}{p}$ is the number of size-p subsets A of X; $p\ell$ is a model of the cost of computing $\phi(A)$; $\binom{k/2}{p}$ is the number of size-p subsets B of Y; $p\ell$ is a model of the cost of computing $\psi(B)$. As in [Ste89] et al., $\binom{k/2}{p}^2/2^\ell$ is used as a model for the number of colliding pairs (A,B), i.e., the number of pairs (A,B) such that $\phi(A)=\psi(B)$. For each collision $2p(n-k)$ is a model of the cost of computing $s-\sum_{i\in A\cup B}(UH)_i$. Note that this cost model does not take the improvements from [BLP08] (Section 5.2) into account. Variants such as bit swapping will be discussed in Remark 5.33. We emphasize that Stern's algorithm is seen here as a fixed-distance-decoding algorithm in order to make a fair comparison to Lee–Brickell's algorithm possible. To the best of our knowledge

article [BLPvT09] was the first to give an explicit version of Stern's algorithm as fixed-distance-decoding algorithm; as opposed to an algorithm for finding low-weight words — see Algorithm 5.2 and Algorithm 6.1 for $q = 2$.
We are going to write the cost exponent of Stern's algorithm as $2^{(\alpha(R,W)+o(1))n}$ where we specify the $o(1)$ part by means of an "error term" $\mathrm{STErr}(n, k, w, \ell, p)$ which is defined as

$$\mathrm{STErr}(n,k,w,\ell,p) = \left(\frac{(k/2)!}{(k/2-p)!(k/2)^p}\right)^2 \cdot \frac{w!}{(w-2p)!w^{2p}} \cdot \frac{(n-k-\ell)!(n-k)^\ell}{(n-k)!} \cdot \frac{(n-k-w)!}{(n-k-\ell-w+2p)!(n-k-w)^{\ell-2p}} \cdot \frac{e^{\epsilon(n-k)+\epsilon(n-w)}}{e^{\epsilon(n-k-w)+\epsilon(n)}},$$

again using $\epsilon(i)$ from equation (5.2).
We will now prove bounds on $\mathrm{STPr}(n, k, w, \ell, p)$. We assume that the code rate $R = k/n$ and error fraction $W = w/n$ satisfy $0 < W < 1 - R < 1$.

Lemma 5.32. *If* $2p < \ell$ *then* $\mathrm{STPr}(n, k, w, \ell, p)$ *equals*

$$2^{-\alpha(R,W)n} \frac{1}{(p!)^2} \left(\frac{RWn}{2(1-R-W)}\right)^{2p} \left(\frac{1-R-W}{1-R}\right)^{\ell} \frac{1}{\beta(R,W)} \mathrm{STErr}(n,k,w,\ell,p).$$

Furthermore
$(1-\frac{2p}{k})^{2p}(1-\frac{2p}{w})^{2p}(1-\frac{n-k-\ell-w+2p}{n-k-w})^{\ell-2p} e^{-\frac{1}{12n}\left(1+\frac{1}{1-R-W}\right)} < \mathrm{STErr}(n,k,w,\ell,p) < (1+\frac{\ell-1}{n-k-\ell+1})^{\ell} e^{\frac{1}{12n}\left(\frac{1}{1-R}+\frac{1}{1-W}\right)}$.

Proof. Replace binomial coefficients by factorials, factor out the $(p, \ell) = (0, 0)$ case, use the refined Stirling formula (5.2), and factor out STErr:

$$\begin{aligned}
\mathrm{STPr}(n,k,w,\ell,p) &= \binom{k/2}{p}^2 \binom{n-k-\ell}{w-2p} \binom{n}{w}^{-1} \\
&= \frac{(n-k-\ell)!}{(w-2p)!(n-k-\ell-w+2p)!} \left(\frac{(k/2)!}{p!(k/2-p)!}\right)^2 \frac{w!(n-w)!}{n!} \\
&= \frac{(n-k)!(n-w)!}{(n-k-w)!n!} \cdot \left(\frac{1}{p!}\frac{(k/2)!}{(k/2-p)!}\right)^2 \frac{w!}{(w-2p)!} \frac{(n-k-w)!}{(n-k-\ell-w+2p)!} \frac{(n-k-\ell)!}{(n-k)!} \\
&= \frac{(n-k)^{n-k+1/2}(n-w)^{n-w+1/2}}{(n-k-w)^{n-k-w+1/2}n^{n+1/2}} \cdot \frac{e^{\epsilon(n-k)+\epsilon(n-w)}}{e^{\epsilon(n-k-w)+\epsilon(n)}} \\
&\quad \cdot \left(\frac{1}{p!}\frac{(k/2)!}{(k/2-p)!}\right)^2 \frac{w!}{(w-2p)!} \frac{(n-k-\ell)!}{(n-k)!} \frac{(n-k-w)!}{(n-k-\ell-w)!} \frac{(n-k-\ell-w)!}{(n-k-\ell-w+2p)!} \\
&= \frac{(n-k)^{n-k+1/2}(n-w)^{n-w+1/2}}{(n-k-w)^{n-k-w+1/2}n^{n+1/2}} \frac{(n-k-w)^\ell}{(n-k)^\ell} \frac{(k/2)^{2p}w^{2p}}{(p!)^2(n-k-w)^{2p}} \mathrm{STErr}(n,k,w,\ell,p).
\end{aligned}$$

Substitute $k = Rn$, $w = Wn$, $\alpha(R, W)$, and $\beta(R, W)$ to obtain:

$$\begin{aligned}
&\mathrm{STPr}(n, k, w, \ell, p) \\
&= \frac{(1-R)^{n-Rn+1/2}(1-W)^{n-Wn+1/2}}{(1-R-W)^{n-Rn-Wn+1/2}} \frac{(n(1-R-W))^{\ell}}{(n(1-R))^{\ell}} \frac{(RWn^2)^{2p}\,\mathrm{STErr}(n, k, w, \ell, p)}{4^p (p!)^2 ((1-R-W)n)^{2p}} \\
&= 2^{-\alpha(R,W)n} \frac{1}{\beta(R, W)} \left(\frac{1-R-W}{1-R}\right)^{\ell} \frac{(RWn)^{2p}\,\mathrm{STErr}(n, k, w, \ell, p)}{4^p (p!)^2 (1-R-W)^{2p}}.
\end{aligned}$$

This is the desired equation for STPr. Similar to the proof of Lemma 5.27 we get upper and lower bounds on STErr by applying Robbins's bounds on $\epsilon(n-k-w)$, $\epsilon(n)$, $\epsilon(n-k)$, and $\epsilon(n-w)$ as well as elementary bounds such as

$$(k/2)!/\left((k/2-p)!\,(k/2)^p\right) = k/2\,(k/2-1)\cdots(k/2-p+1)/\,(k/2)^p \le 1$$

and

$$\begin{aligned}
(k/2)!/\left((k/2-p)!\,(k/2)^p\right) &= k/2\,(k/2-1)\cdots(k/2-p+1)/\,(k/2)^p \\
&\ge (k/2-p)^p/\,(k/2)^p = (1-2p/k)^p\,.
\end{aligned}$$

□

Note that the bounds for STErr do not converge for large p. It is common folklore—see, for example, [OS08]—that ℓ is optimally chosen as $\log_2 \binom{k/2}{p}$, balancing $\binom{k/2}{p}$ with $\binom{k/2}{p}^2/2^{\ell}$. However, starting from this balance, increasing ℓ by 1 produces better results: it chops $2\binom{k/2}{p}^2 p(n-k)/2^{\ell}$ in half without seriously affecting $2\binom{k/2}{p} p\ell$ or $\mathrm{STPr}(n, k, w, \ell, p)$. Choosing ℓ close to $\log_2 \binom{k/2}{p} + \log_2(n-k)$ would ensure that $2\binom{k/2}{p} p\ell$ dominates but would also significantly hurt STPr.
With any reasonable choice of ℓ, increasing p by 1 means that the dominating term $2\binom{k/2}{p} p\ell$ increases by a factor of approximately $k/(2p)$ while the denominator $\mathrm{STPr}(n, k, w, \ell, p)$ increases by a factor of approximately $(k/2p)^2 w^2/(n-k-w)^2$. Overall $\mathrm{STCost}(n, k, w, \ell, p)$ decreases by a factor of approximately $(k/2p)w^2/(n-k-w)^2 = (R/2)(W/(1-R-W))^2(n/p)$. The improvement from Lee–Brickell to Stern is therefore, for fixed R and W, more than any constant power of n.

5.4.4 Implications for code-based cryptography

The standard choices $w = t$ and $k = n - t\lceil \log_2 n \rceil$ for the McEliece parameters imply that the code rate $R = k/n$ and the error fraction $W = w/n$ are related by $W = (1-R)/\lceil \log_2 n \rceil$. For example, if $R = 1/2$, then $W = 1/(2\lceil \log_2 n \rceil)$. Consequently $W \to 0$ as $n \to \infty$.
The function

$$\alpha(R, W) = (1-R-W)\log_2(1-R-W) - (1-R)\log_2(1-R) - (1-W)\log_2(1-W)$$

has series expansion $-W\log_2(1-R)+W^2R/(2(1-R)\log 2)+\cdots$ around $W=0$. In particular, $\alpha(R,(1-R)/\lceil\log_2 n\rceil)=(-(1-R)\log_2(1-R)+o(1))/\log_2 n$ as $n\to\infty$. For example, $\alpha(1/2,1/(2\lceil\log_2 n\rceil))=((1/2)+o(1))/\log_2 n$, so

$$\mathrm{LBCost}(n,(1/2)n,(1/2)n/\lceil\log_2 n\rceil,0)=2^{(1/2+o(1))n/\log_2 n}.$$

Note the distinction between the sublinear exponent in attacks against the McEliece cryptosystem and the linear exponents often quoted for asymptotic information-set decoding. The critical difference is that the error fraction in the McEliece cryptosystem is $\Theta(1/\log_2 n)$ for a fixed rate R while the Gilbert–Varshamov bound is $\Theta(1)$.

Taking more terms in the $\alpha(R,W)$ series shows that

$$2^{\alpha(R,W)n}=(1-R)^{-Wn}e^{W^2Rn/(2(1-R))}\cdots=\left(\frac{1}{1-R}\right)^{w}e^{RWw/(2(1-R))}\cdots;$$

e.g., $2^{\alpha(R,W)n}=2^w e^{Ww/2}\cdots$ for $R=1/2$. For example, for McEliece's original $R=524/1024$ and $W=50/1024$, the leading factor $(1/(1-R))^w$ is $2^{51.71\ldots}$, and the next factor $e^{RWw/(2(1-R))}$ is $2^{1.84\ldots}$, with product $2^{53.55\ldots}$, while $\alpha(R,W)=53.65\ldots$.

The detailed analyses in this section, such as Lemma 5.27, allow much more precise comparisons between various decoding algorithms. For example, increasing p from 0 to 2 saves a factor $(R^2(1-R^2)/(1+R^2)+o(1))n^2/(\log_2 n)^2$ in $\mathrm{LBCost}(n,Rn,(1-R)n/\lceil\log_2 n\rceil,p)$, and increasing p from 2 to 3 loses a factor $\Theta(\log_2 n)$.

Adding an extra error, while leaving R constant, increases the cost by a factor of approximately $1/(1-R)$; more precisely, approximately $e^{R/(2\log_2 n)}/(1-R)$. Adding an extra code dimension, while leaving W constant, has a different effect: it increases the cost by a factor of approximately $(1-R)/(1-R-W)\approx 1+1/\log_2 n$.

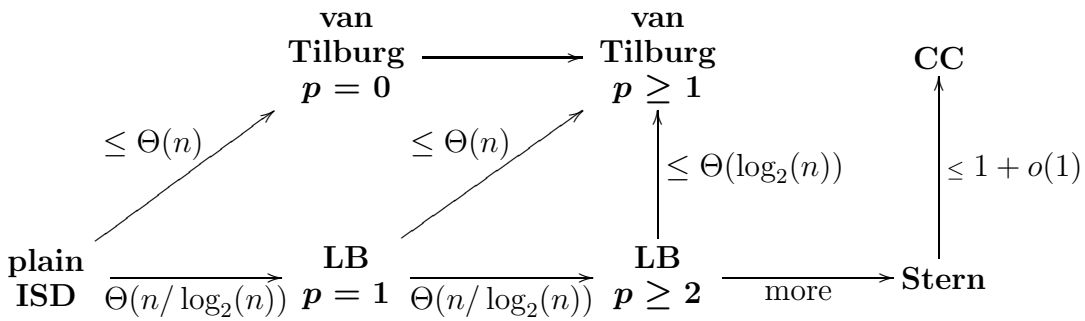

Figure 5.4: Relations between variants of information-set-decoding algorithms designed to reduce time spent on row reduction. By **plain ISD** we mean for the simplest case of information-set decoding (Lee–Brickell (**LB**) with $p=0$); **CC** stands for the Canteaut–Chabaud tweak.

Remark 5.33. There are several variants of information-set decoding designed to reduce the cost of row reduction, sometimes at the expense of success probability. E.g., van Tilburg's version of the Lee–Brickell algorithm [vT90] and the Canteaut–Chabaud version of Stern's algorithm [CC98]. Both variants use the *bit swapping technique* outlined in Section 5.2.1; van Tilburg saves a non-constant factor for Lee–Brickell but the gain of the Canteaut–Chabaud algorithm is at most a factor $1+o(1)$ over Stern. The critical point is that for large n row reduction takes negligible time inside Stern's algorithm, since p is large.
What does save a non-constant factor compared to Stern's algorithm is the suggestion in [BLP08] to reuse additions among sets of p columns. The savings here is a factor of approximately p in Stern's $p\ell$ bottleneck.

Chapter 6

Information-set decoding over $\mathbb{F}_q$

Several articles have suggested to use base fields other than $\mathbb{F}_2$ for the McEliece cryptosystem, e.g. [JM96], [BL05], and more recently [BCGO09] and [MB09]. This idea is interesting as it has the potential to reduce the public-key size. The analysis in Section 5.2 showed that in order to achieve 128-bit security the McEliece private key should be a binary Goppa code of length 2960 and dimension 2288 with a degree-56 Goppa polynomial and 57 added errors. Using an equal-size code over $\mathbb{F}_q$ would save a factor of $\log_2 q$: row and column dimension of the generator matrix both shrink by a factor of $\log_2 q$ at the cost of the matrix entries having size $\log_2 q$. However, information-set-decoding algorithms do not scale purely with the code size. It is important to understand the implications of changing from $\mathbb{F}_2$ to $\mathbb{F}_q$ for arbitrary prime powers q on the attacks. Note that [FOPT10] and [GL10] claim structural attacks against [MB09] but they assume that the codes are dyadic and do not attack the general principle of using larger base fields.
This chapter generalizes Lee–Brickell's algorithm and Stern's algorithm to decoding algorithms for codes over arbitrary fields and extends the improvements from [BLP08] and [FS09] which were described in Chapter 5 of this thesis. This chapter gives a precise analysis of these improved and generalized algorithms. For $q = 31$, Goppa code parameters (length n, dimension k, and degree t of the Goppa polynomial) are presented that require 2^{128} bit operations to compute the closest codeword, i.e., to break McEliece's system using a code of the form $\Gamma_{31}(a_1, \ldots, a_n, g)$.
The results presented in this chapter appeared in [Pet10]. The main differences between the content of this chapter and [Pet10] are the following.

- The descriptions and analyses of the generalized Stern algorithm with classical split and the generalized Stern algorithm with birthday speedup (Sections 3,4,6,7 in [Pet10]) are merged.
- Speedups for the first step of the generalized Stern algorithm are only briefly mentioned as they were discussed in Chapter 5 and also due to their minor impact on performance.
- Section 6.2.3 discusses potential speedups for the "list-building step" in the generalized Stern algorithm.
- Section 6.3 describes the results of the experiments with the MPFI implementation of the Markov-chain iteration count. Moreover, key sizes for a 128-bit

secure McEliece cryptosystem with Goppa codes over fields other than binary fields are investigated. The discussion of key sizes for CCA2-secure variants of the McEliece cryptosystem is taken from [BLP11] which is joint work with Bernstein and Lange.

6.1 Generalization of information-set decoding

Chapter 5 showed that Stern's algorithm is more efficient and supersedes the Lee–Brickell algorithm but the latter is easier to understand and the generalization of it can be used as a stepping stone to the generalization of Stern's.
The algorithms in this chapter get as input a ciphertext y in $\mathbb{F}_q^n$, an error weight $w \geq 0$, and a generator matrix G of a q-*ary* linear code C of length n and dimension k with unknown structure. The closest codeword c in C has distance w from y.

6.1.1 The generalized Lee–Brickell algorithm

Let p be an integer with $0 \leq p \leq w$. The generalization of the Lee–Brickell algorithm allows p errors in positions indexed by the information set similarly to the binary case. The p errors can be corrected by finding the p *weighted* rows of G corresponding to error indices in I. We keep the notation as in Section 5.1.1: the restriction of the generator matrix G to the columns indexed by an information set I is denoted by G_I and the unique row of $G_I^{-1}G$ where the column indexed by some $a \in I$ has a 1 is denoted by G_a.
The generalized Lee–Brickell algorithm changes as follows: replace Step 3 in Algorithm 5.1 by the following step.

3: **For each** *size-p subset $A = \{a_1, \ldots, a_p\} \subset I$ and for each $m = (m_1, \ldots, m_p)$ in $(\mathbb{F}_q^*)^p$:* compute $e = y - \sum_{i=1}^{p} m_i G_{a_i}$. **If** *$e$ has weight w* **then return** e. **Else** go back to Step 1.

Remark 6.1. If $p = 0$ Step 3 only consists of checking whether $y - y_I G_I^{-1} G$ has weight w. If $p > 0$ the loop requires going through all possible weighted sums of p rows of G which need to be subtracted from $y - y_I G_I^{-1} G$ in order to make up for the p errors permitted in I. Section 6.2 explains how to generate all vectors needed using exactly one row addition for each combination.

The parameter p is chosen to be a small number to keep the number of size-p subsets small in Step 3.

6.1.2 The generalized Stern algorithm

Recall that Stern's algorithm was originally designed to look for a codeword of a given weight in a binary linear code.

The generalized Stern algorithm uses a generator matrix instead of a parity-check matrix. At the beginning an information set I is chosen. Then the algorithm tries to build the weight-w vector e with $y - e \in C$ by subtracting *weighted* rows of the systematic generator matrix $G_I^{-1}G$ of y in order to correct $2p$ errors which are assumed to lie in I. The algorithm finds e if an information set I together with sets X, Y, and Z can be found such that e has weights p, p, 0 on the positions indexed by X, Y, and Z, respectively. This is done by testing for collisions on ℓ positions outside the information set: the candidates for collisions are sums of p weighted rows of G and vectors generated by subtracting p weighted rows from $y - G_I^{-1}G$, all restricted to ℓ positions. Each collision vector has by construction weight 0 on ℓ positions and weight $2p$ on positions indexed by I; if it has weight w in total it is the desired error vector.
The algorithm has two integer parameters: a parameter p with $0 \leq p \leq w$ and ℓ with $0 \leq \ell \leq n-k$. For the sake of simplicity assume that k is even. The algorithm consists of a series of independent iterations, each iteration making random choices. If the set I chosen in Step 1 does not lead to a weight-w word in Step 10 another iteration has to be performed. Each iteration consists of the steps described in Algorithm 6.1.

Algorithm 6.1: Generalized Stern algorithm

Input: A generator matrix $G \in \mathbb{F}_2^{k \times n}$ for a q-ary linear code C, a vector $y \in \mathbb{F}_q^n$, and an integer $w \geq 0$.
Output: A weight-w element $e \in \mathbb{F}_q^n$ with $y - e \in C$ if such e exists.

1: Choose an information set I. Compute $G_I^{-1}G$ and replace y by $y - y_I G_I^{-1} G$.
2: Select a uniform random size-$\lfloor k/2 \rfloor$ subset $X \subset I$.
3: Define $Y = I \setminus X$.
4: Select a size-ℓ subset Z of $\{1, \ldots, n\} \setminus I$.
5: **For each** *uniform random size-p subset $A = \{a_1, \ldots, a_p\} \subset X$ and* **for each** $m = (m_1, \ldots, m_p) \in (\mathbb{F}_q^*)^p$*:* consider the p rows $r_i = m_i G_{a_i}$ and compute $\phi_m(A) = y - \sum_i r_i$ restricted to ℓ columns indexed by Z.
6: **For each** *uniform random size-p subset $B = \{b_1, \ldots, b_p\} \subset Y$ and* **for each** $m' = (m'_1, \ldots, m'_p) \in (\mathbb{F}_q^*)^p$*:* consider the p rows $r'_j = m'_j G_{b_j}$ and compute $\psi_{m'}(B) = \sum_j r'_j$ restricted to ℓ columns indexed by Z.
7: **For each** *pair (A, B) and coefficient vectors m, m' where there is a pair of colliding vectors $\phi_m(A) = \psi_{m'}(B)$*
8: Compute $e = y - (\sum_i m_i G_{a_i} + \sum_j m'_j G_{b_j})$.
9: **If** *e has weight w* **then return** e.
10: Go back to Step 1.

For $q = 2$ and $y = 0$ this is Stern's algorithm using the generator matrix of a linear code.

Remark 6.2. The "birthday speedup" which was discussed in Section 5.3.2 can easily be adapted to work for the generalized Stern algorithm: in Algorithm 6.1

remove Steps 2–3, and *replace* Steps 5–6 by the following two steps.

5: **Repeat** N times: choose a uniform random size-p set $A = \{a_1, \ldots, a_p\} \subseteq I$ and consider **for each** $m = (m_1, \ldots, m_p) \in (\mathbb{F}_q^*)^p$: the p rows $r_i = m_i G_{a_i}$ and compute $\phi_m(A) = y - \sum_i r_i$ restricted to ℓ columns indexed by Z.
6: **Repeat** N times: choose a uniform random size-p set $B = \{b_1, \ldots, b_p\} \subseteq I$ and consider **for each** $m' = (m'_1, \ldots, m'_p) \in (\mathbb{F}_q^*)^p$: the p rows $r'_j = m'_j G_{b_j}$ and compute $\psi_{m'}(B) = \sum_j r'_j$ restricted to ℓ columns indexed by Z.

Note that Steps 1, 4, and 7–10 stay the same.

In the following whenever I is split into two disjoint sets X and Y as in the original version then N equals $\binom{k/2}{p}$. If one uses the birthday speedup then N is another algorithm parameter which needs to be optimized. A rough estimate is $N \approx \binom{k}{p} \Big/ \sqrt{\binom{2p}{p}}$ as explained in Section 5.3.2.

6.2 Fast generalized Stern algorithm

This section analyzes the cost for the generalization of Stern's algorithm as presented in Section 6.1.2. The analysis takes the techniques from [BLP08] for the binary case into account which were discussed in Section 5.2. First, the cost of the algorithm of linear codes over prime fields is discussed, then the cost for extension fields.

6.2.1 Analysis for prime fields

Let G be the generator matrix of a linear code over $\mathbb{F}_q$ where q is prime. As in the binary case the algorithm consists of basically three steps: "updating the matrix G with respect to a new information set I", "list building", and "collision handling."

Remark 6.3 (Reusing additions)**.** The "list-building step" computes $N(q-1)^p$ vectors $y - \sum_{i=1}^{p} m_i G_{a_i}$ on ℓ positions coming from size-p subsets A in X. Computing those vectors naively one by one would require $p\ell$ multiplications and $p\ell$ additions in $\mathbb{F}_q$ per vector.
Instead, compute each sum $\sum_{i=1}^{p} m_i G_{a_i}$ using intermediate sums; this takes about one row operation per sum (compare to Section 5.2.2). Note that the sums in Step 5 additionally involve adding y. The naive approach is to subtract each $\sum_i m_i G_{a_i}$ from y. Recall that for $i \in I$ the vector G_i is the unique row of $G_I^{-1} G$ where the column indexed by i has a 1. Here we need to distinguish two cases. If the sets X and Y are disjoint, both of size $k/2$, then each combination $y - \sum_i m_i G_{a_i}$ includes at least one vector $y - G_i$ for $k/2 - p + 1$ rows G_i with $i \in X$. So, it is sufficient to carry out only $k/2 - p + 1$ additions for y. If we use the "birthday speedup" then we can build all possible vectors with indices in X and Y at the same time. We can start by first computing $k - p + 1$ vectors $y - G_i$ with $i \in I$.

Remark 6.4 (Fast collision handling)**.** The expected number of colliding vectors $\phi_m(A)$, $\psi_{m'}(B)$ in Step 7 is about $N^2(q-1)^{2p}/q^\ell$, under the assumption that all vectors $\phi_m(A)$, $\psi_{m'}(B)$ are uniformly distributed among all q^ℓ possible values. For each collision one computes y minus the sum of $2p$ weighted rows on all positions outside I and Z. Naive computation of one such a vector would take $2p$ multiplications and $2p$ additions on $n-k-\ell$ positions. However, first of all one can discard multiplications by 1, leaving $2p$ additions and $(2p)(q-2)/(q-1)$ multiplications. Looking more carefully one observes that each entry has a chance of $(q-1)/q$ of being a nonzero entry. In order to save operations, one computes the result in a column-by-column fashion and uses an early abort: after about $(q/(q-1))(w-2p+1)$ columns are handled it is very likely that the resulting row vector has more than the allowed $w-2p$ nonzero entries and can be discarded. This means that partial collisions that do not lead to a full collision consume only $(q/(q-1))(w-2p+1)$ operations.
If y is the all-zero codeword the algorithm looks for a weight-w codeword. Then the cost for adding y to weighted rows G_a coming from sets A can be neglected.

Remark 6.5 (Reusing existing pivots)**.** Section 4 in [Pet10] contains a detailed analysis of all techniques in Section 5.2.1 to speed up Gaussian elimination for the q-ary case; only the trick of choosing multiple sets Z of size ℓ is omitted. Also here we leave out the details since they do not differ from the binary case. Note that the idea of reusing all but c entries of the previous information set (see 5.7) proves useful for small fields since in this case only c new columns have to be pivoted at the beginning of each iteration.

However, Gaussian elimination is even less of a bottleneck than in binary fields. For large fields the cost for Step 1 compared to "list building" and "collision handling" turns out to be negligible. We assume that each iteration chooses a new information set. So, we crudely estimate the cost and assume Gaussian elimination to take $(n-k)^2(n+k)$ operations in $\mathbb{F}_q$.

Remark 6.6. The choice of the field does not influence the success probability of one iteration of the algorithm since all possible weights in $\mathbb{F}_q^*$ are tested. Without birthday trick the probability equals $\mathrm{STPr}(n,k,w,\ell,p)$ which was defined in Section 5.2.3. The analysis of the birthday-speedup success probability is the same as in Remark 5.23.

Remark 6.7 (Number of iterations)**.** If each iteration chooses a new information set uniformly at random then the iterations are independent. The expected number of iterations is the reciprocal of the success probability of the first iteration, which is given by $\mathrm{STPr}(n,k,w,\ell,p)$.
Reusing all but c elements of the previous information set results in dependent iterations. The expected number of iterations can be computed using the same Markov chain as in Section 5.2.3; see also Section 6.3.1.

Remark 6.8 (Cost measure)**.** To estimate the cost per iteration we need to express the cost in one measure, namely in additions in $\mathbb{F}_q$. We described Steps 5–7 using

multiplications. Since we consider only quite small fields $\mathbb{F}_q$ multiplications can be implemented as table lookups and thus cost the same as one addition.

Remark 6.9 (Cost for one iteration of Stern's algorithm)**.** The cost of one iteration of Algorithm 6.1 is as follows:

$$\begin{aligned}&(n-k)^2(n+k) + ((\kappa - p + 1) + 2N(q-1)^p)\,\ell \\ &+ \frac{q}{q-1}(w-2p+1)2p\left(1+\frac{q-2}{q-1}\right)\frac{N^2(q-1)^{2p}}{q^\ell},\end{aligned}$$

where $\kappa = k/2$ and $N = \binom{k/2}{p}$ if I is split into disjoint sets of size $k/2$. If one uses the birthday speedup then $\kappa = k$ and N is another algorithm parameter which needs to be optimized as discussed in Section 5.3.2.

Remark 6.10 (Choice of parameters for Stern's algorithm)**.** The parameter p is chosen quite small in order to minimize the cost of going though all subsets A, B of X and Y. The parameter ℓ is chosen to balance the number of all possible length-ℓ vectors $\phi_m(A)$ and $\psi_{m'}(B)$ with the number of expected collisions on ℓ positions. A reasonable choice is $\ell = \log_q N + p\log_q(q-1)$.

6.2.2 Analysis for extension fields

We presented a generalization for information-set decoding over arbitrary finite fields $\mathbb{F}_q$. However, the cost analysis in the previous section was restricted to prime values of q. Here we point out the differences in handling arbitrary finite fields.
The main difference in handling arbitrary finite fields is in Steps 5–6 of the generalized Stern algorithm when computing sums of p rows coming from subsets A of X and sums of p rows coming from subsets B of Y. In prime fields all elements are reached by repeated addition since 1 generates the additive group. If q is a prime power 1 does not generate the additive group.
Let $\mathbb{F}_q$ be represented over its prime field via a sparse irreducible polynomial $h(x)$. To reach all elements we also need to compute x times a field element, which is essentially the cost of reducing modulo h. In turn this means several additions of the prime field elements. Even though these operations technically are not additions in $\mathbb{F}_q$, the costs are essentially the same. This means that the costs of these steps are the same as before.
In the analysis of Step 7 we need to account for multiplications with the coefficient vectors $(m_1, \ldots, m_p)$ and $(m'_1, \ldots, m'_p)$. This is the same problem that we faced in the previous section and thus we use the same assumption, namely that one multiplication in $\mathbb{F}_q$ has about the same cost as one addition in $\mathbb{F}_q$.

6.2.3 Discussion

Note that generating all possible p-sums in the "list-building step" generates an overhead. Similarly, Step 3 in the generalized Lee–Brickell algorithm goes through

more vectors than needed. In particular, there are many linearly dependent vectors in $\mathbb{F}_q^\ell$ which can be built as sums of p weighted rows G_a restricted to ℓ positions. Minder and Sinclair in [MS10] present a generalization of Wagner's generalized-birthday algorithm [Wag02a] (see also Chapter 9) which constructs sums of elements in a vector space over some arbitrary field $\mathbb{F}_r$. Minder and Sinclair say that their version "suffers at most only a factor 4 increase in space and a factor $\sqrt{r}$ in time compared to the binary version." We briefly comment how the technique proposed by Minder and Sinclair in [MS10, Section 5.4] can be adapted to our generalization of Lee–Brickell's and Stern's algorithm and discuss its uses. Note that other than Minder and Sinclair we use q instead of r to denote the field size.

Remark 6.11. Minder and Sinclair propose to generate only normalized vectors which in our case are sums of p rows; "normalized" means that the first entry of each sum of p rows is 1. This is done by first computing the sum v of p rows and then multiplying by the inverse of the first entry of v. We question that one can produce those sums efficiently. Also note that, in the collision-handling step, one still has to find the right multiple of each collision vector in order to produce an error vector having weight $w - 2p$ on the remaining $n - k - \ell$ positions. In theory the Minder–Sinclair idea could save a factor of $\sqrt{q-1}$ since both lists coming from indices in X and Y contain vectors in $\mathbb{F}_q^\ell$ having one entry fixed to 1. We comment that using intermediate sums each p-sum costs only one addition in $\mathbb{F}_q^\ell$ anyway. Also no inversions are needed here. Of course, for small fields inversions also could be implemented as table lookups. One could try to build the first combinations of p sums in the Minder–Sinclair version using intermediate sums. However, this is not likely to gain a lot in practice; the speedup for "list building" described in Remarks 5.10 and 6.3 achieves the cost of almost one addition per sum in $\mathbb{F}_q^\ell$ only because of the abundance of vectors which need to be generated. For small fields the idea does not have much impact on the overall cost. For larger q it has even less potential as the cost for "list building" and "collision handling" increases. We therefore stick to the method of generating more vectors than needed.

The Minder–Sinclair idea was claimed by [NCBB10] to save a factor of $\sqrt{q-1}$ for q-ary information-set decoding. But as many other points in [NCBB10] the idea was not carefully analyzed by a proper operation count.

6.3 Parameters

This section proposes new parameters for the McEliece cryptosystem using Goppa codes $\Gamma_q(a_1, \ldots, a_n, g)$ over $\mathbb{F}_q$. Recall that for $q > 2$ the error-correcting capability of $\Gamma_q(a_1, \ldots, a_n, g)$ is only $(\deg g)/2$. See, however, Chapter 7 for a new special class of Goppa codes which allow correction of more errors.

6.3.1 MPFI implementation

We adapted the Markov chain implementation from [BLP08] to look for parameters for the McEliece cryptosystem using codes over arbitrary fields $\mathbb{F}_q$. The implementation computes the cost of Algorithm 6.1 including all improvements discussed in Section 6.2.1, in particular, the reusing of existing pivots; see Remark 6.5.
We took the parameters proposed in [BCGO09] and [MB09], respectively, and investigated their security against our attack. The code can be found at `http://www.win.tue.nl/~cpeters/isdfq.html`.
Note that some instances of the setups in [BCGO09] and [MB09] are broken by the structural attacks in [FOPT10] and [GL10]. Our analysis showed how crude many of the original security estimates against non-structural attacks are. The results are summarized in Table 6.1.

6.3.2 Codes for 128-bit security

The public key in Kobara and Imai's CCA2-secure variant [KI01] of the McEliece cryptosystem can be stored in systematic form as $(n-k)k$ entries in $\mathbb{F}_q$. The same is true for the Niederreiter variant; see, e.g., [OS08, Algorithm 2.3]. The simplest representation of an element of $\mathbb{F}_q$ takes $\lceil \log_2 q \rceil$ bits (e.g., 3 bits for $q = 5$), but a sequence of elements can be compressed: one stores a batch of b elements of $\mathbb{F}_q$ in $\lceil \log_2 q^b \rceil$ bits, at the expense of some easy computation to recover the individual elements. As b grows the storage per field element drops to approximately $\log_2 q$ bits, so $(n-k)k$ elements can be stored using about $\lceil (n-k)k \log_2 q \rceil$ bits.
We carried out experiments to find small key sizes for a 128-bit secure McEliece setup using Goppa codes over non-binary fields. These experiments estimate the cost of Algorithm 6.1 for decrypting a McEliece ciphertext using the operation count summarized in Remark 6.9. For finite-fields sizes $2 \leq q \leq 32$ the code length n, and the degree t for the Goppa polynomial need to be chosen. Then k is set to $n - \lceil \log_q n \rceil t$. The parameter t determines the public error weight w. We distinguish two cases: if $q = 2$ Patterson's algorithm can correct $w = t$ errors whereas if $q > 2$ only $(t+1)/2$ errors can be corrected.
Figure 6.1 shows the graph of the resulting key sizes $\lceil (n-k)k \log_2 q \rceil$ for $3 \leq q \leq 32$ compared to the key size we get when using a binary Goppa code. Note that we recomputed the binary case to find 128-bit secure binary McEliece parameters $(n, k, t) = (3009, 2325, 57)$ such that classical decoding (and not list decoding as for the $[2960, 2288]$ code in Remark 5.19) can be used.
Moreover, Figure 6.1 shows that Goppa codes over small fields of size 3, 4, 5, 7, 8, 9, 11 cannot compete with binary Goppa codes in terms of key size. However, increasing the field size further yields astonishingly good results: we can halve the key size when using a Goppa code over $\mathbb{F}_{31}$ instead of the binary $[2960, 2288]$ code mentioned above.
Our experiments with the MPFI implementation showed that a code of length n and dimension k over $\mathbb{F}_{31}$ with $n = 961$, $k = 771$, and $w = 48$ introduced errors achieves

code parameters				claimed security level	disjoint split						birthday trick						
q	n	k	w		$\log_2$ bit ops	$\log_2$ #it	p	ℓ	c	r	$\log_2$ bit ops	$\log_2$ #it	p	ℓ	c	r	μ
256	459	255	50	80	77.02	55.27	1	3	1	1	77.10	54.37	1	3	2	1	1.3
256	510	306	50	90	84.79	62.75	1	3	1	1	84.87	61.85	1	3	2	1	1.3
256	612	408	50	100	98.19	75.68	1	3	1	1	98.29	74.78	1	3	2	1	1.3
256	765	510	50	120	97.45	74.49	1	3	1	1	97.52	73.61	1	3	2	1	1.2
1024	450	225	56	80	76.84	53.26	1	3	1	1	76.88	52.37	1	3	2	1	1.3
1024	558	279	63	90	83.97	60.05	1	3	1	1	84.00	59.20	1	3	2	1	1.2
1024	744	372	54	110	73.65	48.71	1	3	2	1	70.54	44.90	1	3	3	1	1.3
4	2560	1536	128	128	181.86	154.41	2	15	4	4	187.46	153.71	2	10	10	5	0.6
16	1408	896	128	128	210.62	179.75	2	8	8	2	210.76	179.26	2	9	10	2	1.1
256	640	512	64	102	181.67	158.82	1	3	1	1	181.62	158.27	1	3	1	1	1.2
256	768	512	128	136	253.02	229.94	1	3	1	1	253.01	229.38	1	3	1	1	1.2
256	1024	512	256	168	329.00	305.55	1	3	1	1	329.04	305.09	1	3	1	1	1.1
2	2304	1281	64	80	83.56	59.17	2	21	8	8	83.39	58.66	2	22	9	9	0.9
2	3584	1537	128	112	112.31	87.69	2	24	8	8	112.18	87.12	2	26	8	8	1.0
2	4096	2048	128	128	136.48	111.10	2	25	8	8	136.47	110.50	2	26	9	9	1.0
2	7168	3073	256	192	216.07	180.45	3	37	32	8	215.91	179.44	3	38	36	9	1.0
2	8192	4097	256	256	265.16	228.23	3	38	16	8	265.01	227.23	3	39	17	9	1.0

Table 6.1: Cost of Algorithm 6.1 against the McEliece cryptosystem with codes using parameters proposed in [BCGO09] (rows 1–7) and [MB09] (rows 8–17). The algorithm parameters p, ℓ are the well-known Stern parameters; parameters c and r were discussed in Section 5.2.1 (note that c and r larger than 1 means that it makes sense to speed up Gaussian elimination). If the birthday trick is used the number of subsets A is chosen to be $\mu \cdot \binom{k}{p}\binom{2p}{p}^{-\frac{1}{2}}$ where μ is a scaling factor. Similarly for the number of subsets B.

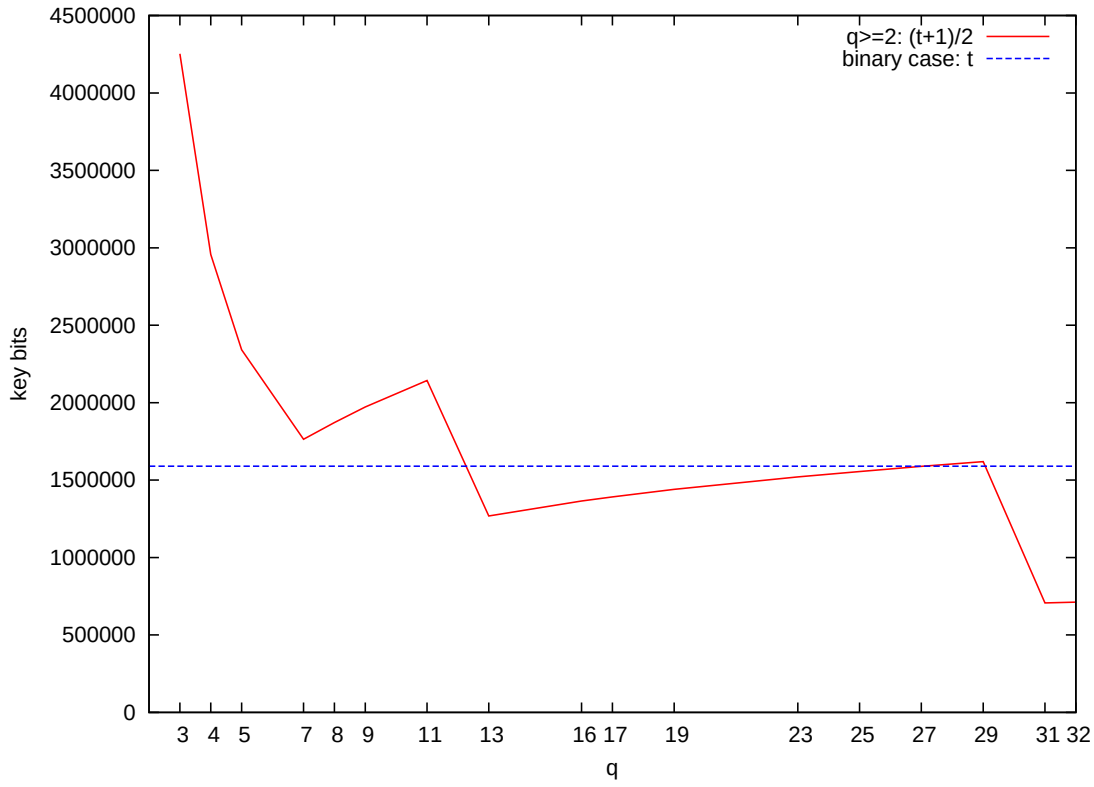

Figure 6.1: Decrease in key sizes when going to larger fields (128-bit security). Horizontal axis is the field size q. The dashed line is the best key size achieved for 128-bit security when using a binary Goppa code. The solid line is the best key size achieved for 128-bit security when using a q-ary Goppa code with classical decoding, i.e., $\lfloor (t+1)/2 \rfloor$ added errors.

128-bit security. Using a Goppa code $\Gamma = \Gamma_{31}(a_1, \ldots, a_n, g)$ with distinct elements $a_1, \ldots, a_n$ in $\mathbb{F}_{31^2}$ and an irreducible polynomial $g(x)$ in $\mathbb{F}_{31^2}[x]$ of degree 97; in this case Γ is the subfield subcode of a code over $\mathbb{F}_{31^2}$. A public key for such a $[961, 771]$ code over $\mathbb{F}_{31}$ would consist of $k(n-k)\log_2 31 = 725741$ bits.
A successful attack needs about $2^{96.815}$ iterations with about $2^{32.207}$ bit operations per iteration. A good choice of parameters are $p = 2$, $\ell = 7$, and $c = 12$. Using the "birthday speedup" costs about the same, namely $2^{129.0290}$ bit operations. In comparison to the classical disjoint split of the information set one can afford to spend more time on Gaussian elimination and consider $c = 17$ new columns in each iteration. Increasing the standard choice $\binom{k}{p}/\sqrt{\binom{2p}{p}}$ of the number of subsets A and B by a factor of 1.1 to $N = 133300$ yields the best result. The expected number of iterations is $2^{95.913}$, each taking about $2^{33.116}$ bit operations.

Chapter 7

Wild McEliece

McEliece's original system uses binary Goppa codes. Several smaller-key variants have been proposed using other codes, such as Reed–Solomon codes [Nie86], generalized Reed–Solomon codes [SS92], quasi-dyadic codes [MB09] or geometric Goppa codes [JM96]. Unfortunately, many specific proposals turned out to be breakable. The most confidence-inspiring proposal for code-based public-key cryptography is still McEliece's original proposal to use binary Goppa codes. For these only information-set-decoding attacks apply. The best defense against this type of attack is to use codes with a larger error-correcting capability. The disadvantage of binary Goppa codes is that they have a comparably large key size. Chapter 6 discussed how using Goppa codes over non-binary fields decreases the key size at the same security level against information-set decoding. However, this effect appears only with sufficiently big base fields such as $\mathbb{F}_{31}$; codes over $\mathbb{F}_3$ and $\mathbb{F}_4$ look worse than those over $\mathbb{F}_2$. The main reason of $\mathbb{F}_2$ being better is that for binary Goppa codes we can use Patterson's decoding algorithm which corrects twice as many errors as decoders for the non-binary case.

This chapter discusses the results of the article "Wild McEliece" [BLP11] which is joint work with Bernstein and Lange. The article proposed using "wild Goppa codes". These are subfield codes over small $\mathbb{F}_q$ that have an increase in error-correcting capability by a factor of about $q/(q-1)$. McEliece's construction using binary Goppa codes is the special case $q = 2$ of our construction.

These codes were analyzed in 1976 by Sugiyama, Kasahara, Hirasawa, and Namekawa [SKHN76] but have not been used in code-based cryptography so far. This chapter explains how to use these codes in the McEliece cryptosystem and how to correct $\lfloor qt/2 \rfloor$ errors where previous proposals corrected only $\lfloor (q-1)t/2 \rfloor$ errors.

The main differences between the content of this chapter and [BLP11] are the following.

- The introduction to classical Goppa codes was moved to Chapter 4.
- The discussion of key sizes for CCA2-secure variants of the McEliece cryptosystem was moved to Chapter 6.

7.1 The wild McEliece cryptosystem

The reader is assumed to be familiar with Goppa codes, and in particular the notation introduced in Section 4.1.3.

7.1.1 Wild Goppa codes

The *wild McEliece cryptosystem* is set up as follows.

- We propose using the McEliece cryptosystem (or equivalently the Niederreiter cryptosystem) with Goppa codes of the form $\Gamma_q(a_1, \ldots, a_n, g^{q-1})$ where g is an irreducible monic polynomial in $\mathbb{F}_{q^m}[x]$ of degree t. Note the exponent $q-1$ in g^{q-1}. We refer to these codes as *wild Goppa codes* for reasons explained later.

- We further propose to use error vectors of weight $\lfloor qt/2 \rfloor$. The advantage of wild Goppa codes is that they allow us to efficiently correct $\lfloor qt/2 \rfloor$ errors (or slightly more with the help of list decoding); see the next sections. For $q \in \{3, 4, \ldots\}$ this is strikingly better than the performance of an irreducible polynomial of the same degree $(q-1)t$, namely correcting $\lfloor (q-1)t/2 \rfloor$ errors. This change does not hurt the code dimension: polynomials of the form g^{q-1} produce codes of dimension at least $n - m(q-1)t$ (and usually exactly $n - m(q-1)t$), just like irreducible polynomials of degree $(q-1)t$.

For $q = 2$ this proposal is not new: it is exactly McEliece's original proposal to use a binary Goppa code $\Gamma_2(a_1, \ldots, a_n, g)$, where g is an irreducible polynomial of degree t, and to use error vectors of weight t. McEliece used Patterson's algorithm to efficiently decode t errors.
In Chapter 6 we considered Goppa codes over slightly larger fields. In particular, switching from binary Goppa codes to codes of the form $\Gamma_{31}(a_1, \ldots, a_n, g)$ yields good results as shown in Section 6.3.2.
What is new in the wild McEliece cryptosystem is the use of Goppa polynomials of the form g^{q-1} for $q \geq 3$, allowing us to correct more errors for the same field size, the same code length, and the same code dimension.

7.1.2 Minimum distance of wild Goppa codes

The following theorem is the main theorem of the article [SKHN76] by Sugiyama, Kasahara, Hirasawa, and Namekawa. What the theorem states is that, for any monic squarefree polynomial g in $\mathbb{F}_{q^m}[x]$, the code $\Gamma_q(a_1, \ldots, a_n, g^{q-1})$ is the same as $\Gamma_q(a_1, \ldots, a_n, g^q)$. The code therefore has *minimum distance* at least $qt+1$. Efficient decoding of $\lfloor qt/2 \rfloor$ errors requires more effort and is discussed in the next section.
The case $q = 2$ of this theorem is due to Goppa, using a different proof that can be found in many textbooks; for example, in [MS77, pp. 341–342]. The case $q \geq 3$ has received less attention. We include a streamlined proof to keep this chapter self-contained.

The proof immediately generalizes from the pair (g^{q-1}, g^q) to the pair (g^{rq-1}, g^{rq}), and to coprime products of such pairs. These generalizations also occur in [SKHN76]. Wirtz in [Wir88], and independently Katsman and Tsfasman in [KT89], further generalized the results of [SKHN76] to geometric Goppa codes. See Janwa and Moreno [JM96] for a discussion of the possibility of using geometric Goppa codes in the McEliece cryptosystem but also Minder's thesis [Min07] and the article by Faure and Minder [FM08] for attacks on the elliptic-curve version and the genus-2 version. We do not consider this possibility further in this chapter.

Theorem 7.1. *Let q be a prime power. Let m be a positive integer. Let n be an integer with $1 \leq n \leq q^m$. Let $a_1, a_2, \ldots, a_n$ be distinct elements of $\mathbb{F}_{q^m}$. Let g be a monic squarefree polynomial in $\mathbb{F}_{q^m}[x]$ coprime to $(x - a_1) \cdots (x - a_n)$. Then $\Gamma_q(a_1, a_2, \ldots, a_n, g^{q-1}) = \Gamma_q(a_1, a_2, \ldots, a_n, g^q)$.*

Proof. If $\sum_i c_i/(x - a_i) = 0$ in $\mathbb{F}_{q^m}[x]/g^q$ then certainly $\sum_i c_i/(x - a_i) = 0$ in $\mathbb{F}_{q^m}[x]/g^{q-1}$.
Conversely, consider any $(c_1, c_2, \ldots, c_n) \in \mathbb{F}_q^n$ such that $\sum_i c_i/(x - a_i) = 0$ in $\mathbb{F}_{q^m}[x]/g^{q-1}$. Find an extension k of $\mathbb{F}_{q^m}$ so that g splits into linear factors in $k[x]$. Then $\sum_i c_i/(x - a_i) = 0$ in $k[x]/g^{q-1}$, so $\sum_i c_i/(x - a_i) = 0$ in $k[x]/(x - r)^{q-1}$ for each factor $x - r$ of g. The elementary series expansion

$$\frac{1}{x - a_i} = -\frac{1}{a_i - r} - \frac{x - r}{(a_i - r)^2} - \frac{(x - r)^2}{(a_i - r)^3} - \cdots$$

then implies

$$\sum_i \frac{c_i}{a_i - r} + (x - r) \sum_i \frac{c_i}{(a_i - r)^2} + (x - r)^2 \sum_i \frac{c_i}{(a_i - r)^3} + \cdots = 0$$

in $k[x]/(x-r)^{q-1}$; i.e., $\sum_i c_i/(a_i-r) = 0$, $\sum_i c_i/(a_i-r)^2 = 0, \ldots, \sum_i c_i/(a_i-r)^{q-1} = 0$. Now take the qth power of the equation $\sum_i c_i/(a_i - r) = 0$, and use the fact that $c_i \in \mathbb{F}_q$, to obtain $\sum_i c_i/(a_i-r)^q = 0$. Work backwards to see that $\sum_i c_i/(x-a_i) = 0$ in $k[x]/(x - r)^q$.
By hypothesis g is the product of its distinct linear factors $x - r$. Therefore g^q is the product of the coprime polynomials $(x - r)^q$, and $\sum_i c_i/(x - a_i) = 0$ in $k[x]/g^q$; i.e., $\sum_i c_i/(x - a_i) = 0$ in $\mathbb{F}_{q^m}[x]/g^q$. □

Remark 7.2 (The "wild" terminology)**.** To explain the name "wild Goppa codes" we briefly review the standard concept of wild ramification. A prime p "ramifies" in a number field L if the unique factorization $p\mathcal{O}_L = Q_1^{e_1} Q_2^{e_2} \cdots$ has an exponent e_i larger than 1, where $\mathcal{O}_L$ is the ring of integers of L and $Q_1, Q_2, \ldots$ are distinct maximal ideals of $\mathcal{O}_L$. Each Q_i with $e_i > 1$ is "ramified over p"; this ramification is "wild" if e_i is divisible by p.
If $\mathcal{O}_L/p$ has the form $\mathbb{F}_p[x]/f$, where f is a monic polynomial in $\mathbb{F}_p[x]$, then the maximal ideals $Q_1, Q_2, \ldots$ correspond naturally to the irreducible factors of f, and

the exponents $e_1, e_2, \ldots$ correspond naturally to the exponents in the factorization of f. In particular, the ramification corresponding to an irreducible factor of f is wild if and only if the exponent is divisible by p.
Similar comments apply to more general extensions of global fields. Ramification corresponding to an irreducible factor φ of a monic polynomial f in $\mathbb{F}_{p^m}[x]$ is wild if and only if the exponent is divisible by p, i.e., the local component of f is a power of φ^p. We take the small step of referring to φ^p as being "wild", and referring to the corresponding Goppa codes as "wild Goppa codes". Of course, if the Goppa code for φ^p is wild, then the Goppa code for φ^{p-1} must also be wild, since (by Theorem 7.1) it is the same code.
The traditional concept of wild ramification is defined by the characteristic of the base field. We find it more useful to allow a change of base from $\mathbb{F}_p$ to $\mathbb{F}_q$, generalizing the definition of wildness to use the size of $\mathbb{F}_q$ rather than just the characteristic of $\mathbb{F}_q$.

7.2 Decrypting wild-McEliece ciphertexts

The main problem faced by a wild-McEliece receiver is to decode $\lfloor qt/2 \rfloor$ errors in the code $\Gamma = \Gamma_q(a_1, \ldots, a_n, g^{q-1})$: i.e., to find a codeword $c = (c_1, \ldots, c_n) \in \Gamma$, given a received word $y = (y_1, \ldots, y_n) \in \mathbb{F}_q^n$ at Hamming distance $\leq \lfloor qt/2 \rfloor$ from c. This section presents an asymptotically fast algorithm that decodes up to $\lfloor qt/2 \rfloor$ errors, and then a "list decoding" algorithm that decodes even more errors.

7.2.1 Classical decoding

Recall the representation of $\Gamma_{q^m}(a_1, \ldots, a_n, g)$ as a generalized Reed–Solomon code; specifically the polynomial view of generalized Reed–Solomon codes summarized in Remark 4.20. It follows from Theorem 7.1 that

$$\begin{aligned}\Gamma &= \Gamma_q(a_1, \ldots, a_n, g^q) \\ &\subseteq \Gamma_{q^m}(a_1, \ldots, a_n, g^q) \\ &= \left\{ \left(\frac{f(a_1)}{h'(a_1)}, \ldots, \frac{f(a_n)}{h'(a_n)} \right) : f \in g^q \mathbb{F}_{q^m}[x], \deg f < n \right\}\end{aligned}$$

where $h = (x - a_1) \cdots (x - a_n)$. We thus view the target codeword $c = (c_1, \ldots, c_n) \in \Gamma$ as a sequence $(f(a_1)/h'(a_1), \ldots, f(a_n)/h'(a_n))$ of function values, where f is a multiple of g^q of degree below n. Recall that the *norm* $|\psi|$ of a polynomial $\psi \in \mathbb{F}_{q^m}[x]$ is defined as $q^{\deg \psi}$ if $\psi \neq 0$ and 0 if $\psi = 0$.
We are given y, the same sequence with $\lfloor qt/2 \rfloor$ errors, or more generally with $\leq \lfloor qt/2 \rfloor$ errors. We first state the decoding algorithm for $\Gamma_{q^m}(a_1, \ldots, a_n, g^q)$ and then show that it reconstructs c from y. We emphasize that the algorithm decodes the bigger code $\Gamma_{q^m}(a_1, \ldots, a_n, g^q)$ as well as the wild Goppa code $\Gamma = \Gamma_q(a_1, \ldots, a_n, g^q)$.

Algorithm 7.1: Alternant decoder

Input: Given a code $\Gamma = \Gamma_{q^m}(a_1, \ldots, a_n, g^q)$, in particular, the polynomials $g(x)$ and $h(x) = (x - a_1) \cdots (x - a_n)$ in $\mathbb{F}_{q^m}[x]$, and a vector $y \in \mathbb{F}_{q^m}^n$ at distance $\lfloor qt/2 \rfloor$ from a codeword $c \in \Gamma$.

Output: The codeword $c \in \mathbb{F}_q^n$.

1: Interpolate $y_1 h'(a_1)/g(a_1)^q, \ldots, y_n h'(a_n)/g(a_n)^q$ into a polynomial φ: i.e., construct the unique $\varphi \in \mathbb{F}_{q^m}[x]$ such that $\varphi(a_i) = y_i h'(a_i)/g(a_i)^q$ and $\deg \varphi < n$.

2: Compute the continued fraction of φ/h to degree $\lfloor qt/2 \rfloor$: i.e., apply the Euclidean algorithm to h and φ, stopping with the first remainder $v_0 h - v_1 \varphi$ of degree $< n - \lfloor qt/2 \rfloor$.

3: Compute $f = (\varphi - v_0 h/v_1) g^q$.

4: **Return** $c = (f(a_1)/h'(a_1), \ldots, f(a_n)/h'(a_n))$.

This algorithm uses $n^{1+o(1)}$ operations in $\mathbb{F}_{q^m}$ if multiplication, evaluation, interpolation, and continued-fraction computation are carried out by standard FFT-based subroutines; see [Ber08a] for a survey of those subroutines.

Remark 7.3 (Correctness of the algorithm)**.** To see that this algorithm works, observe that φ has many values in common with the target polynomial f/g^q: specifically, $\varphi(a_i) = f(a_i)/g(a_i)^q$ for all but $\lfloor qt/2 \rfloor$ values of i. In other words, the error-locator polynomial

$$\epsilon = \prod_{i : \frac{f(a_i)}{g(a_i)^q} \neq \varphi(a_i)} (x - a_i)$$

has degree at most $\lfloor qt/2 \rfloor$. The difference $\varphi - f/g^q$ is a multiple of h/ϵ, say $\delta h/\epsilon$; i.e., $\delta/\epsilon - \varphi/h = -(f/g^q)/h$. The idea is to construct δ and ϵ by examining continued fractions of φ/h.

The difference $\delta/\epsilon - \varphi/h = -(f/g^q)/h$ has norm smaller than $|1/x^{qt}|$ and is therefore smaller than $1/\epsilon^2$, so δ/ϵ is a "best approximation" to φ/h, so δ/ϵ must appear as a convergent to the continued fraction of φ/h, specifically the convergent at degree $\lfloor qt/2 \rfloor$. Consequently $\delta/\epsilon = v_0/v_1$; i.e., $f/g^q = \varphi - v_0 h/v_1$.

More generally, one can use any Reed–Solomon decoder to reconstruct f/g^q from the values $f(a_1)/g(a_1)^q, \ldots, f(a_n)/g(a_n)^q$ with $\lfloor qt/2 \rfloor$ errors. This is an illustration of the following sequence of standard transformations:

Reed–Solomon decoder $\Rightarrow$ generalized Reed–Solomon decoder
$\Rightarrow$ alternant decoder $\Rightarrow$ Goppa decoder.

The resulting decoder corrects $\lfloor (\deg g)/2 \rfloor$ errors for Goppa codes $\Gamma_q(a_1, \ldots, a_n, g)$; in particular, $\lfloor q(\deg g)/2 \rfloor$ errors for $\Gamma_q(a_1, \ldots, a_n, g^q)$; and so $\lfloor q(\deg g)/2 \rfloor$ errors for $\Gamma_q(a_1, \ldots, a_n, g^{q-1})$, by Theorem 7.1.

We do not claim that the particular algorithm stated above is the fastest possible decoder, and in particular we do not claim that it is quite as fast as Patterson's

algorithm [Pat75] for $q = 2$. However, it has essentially the same scalability in n as Patterson's algorithm, works for general q, and is obviously fast enough to be usable.
An example implementation of a wild-Goppa-code decoder in the Sage computer-algebra system [S+10] can be found at `http://pqcrypto.org/users/christiane/wild.html`.

7.2.2 List decoding

By switching from a classical Reed–Solomon decoding algorithm to the Guruswami–Sudan list-decoding algorithm [GS99] we can efficiently correct $n - \sqrt{n(n-qt)} > \lfloor qt/2 \rfloor$ errors in the function values $f(a_1)/g(a_1)^q, \ldots, f(a_n)/g(a_n)^q$. This algorithm is not as fast as a classical decoder but still takes polynomial time. Consequently we can handle $n - \sqrt{n(n-qt)}$ errors in the wild Goppa code $\Gamma_q(a_1, \ldots, a_n, g^{q-1})$.
This algorithm can, at least in theory, produce several possible codewords c. This does not pose a problem for the CCA2-secure variants of the McEliece cryptosystem introduced by Kobara and Imai in [KI01]: those variants automatically reject all codewords that do not include proper labels cryptographically protected by an "all-or-nothing transform".
As above, we do not claim that this algorithm is the fastest possible decoder. In particular, for $q = 2$ the same error-correcting capacity was obtained by Bernstein in [Ber08b] using a more complicated algorithm, analogous to Patterson's algorithm; we do not claim that the $\Gamma(a_1, \ldots, a_n, g^2)$ approach is as fast as that algorithm.
With more decoding effort we can handle a few additional errors by the standard idea of combinatorially guessing those errors. Each additional error produces a noticeable reduction of key size, as will be shown in Section 7.4. In many applications, the McEliece decoding time is unnoticeable while the McEliece key size is a problem, so allowing extra errors at the expense of decoding time is a good tradeoff.

7.3 Attacks

This section discusses several attacks against the wild McEliece cryptosystem. All of the attacks scale poorly to large key sizes; Section 7.4 presents parameters that are safe against all of these attacks. We do not claim novelty for any of the attack ideas.
We emphasize that the wild McEliece cryptosystem includes, as a special case, the original McEliece cryptosystem. A complete break of the wild McEliece cryptosystem would therefore imply a complete break of the original McEliece cryptosystem, a system that has survived scrutiny for 32 years. It is of course possible that there is a magical dividing line between $q = 2$ and $q = 3$, an attack that breaks every new case of our proposal while leaving the original cryptosystem untouched, but we have not found any such line.
We focus on inversion attacks, i.e., attacks against the one-wayness of wild McEliece

encryption. There are several well-known chosen-ciphertext attacks that break semantic security without breaking one-wayness, but all of these attacks are stopped by standard conversions; see [KI01].

7.3.1 Generic decoding methods

The top threat against the original McEliece cryptosystem, the attack algorithm that has always dictated key-size recommendations, is information-set decoding; as a generic decoding method information-set decoding does not rely on any particular code structure. The same attack also appears to be the top threat against the wild McEliece cryptosystem for $\mathbb{F}_3$, $\mathbb{F}_4$, etc.
The exact complexity of information-set decoding is not easy to state concisely. We rely on, and refer the reader to, the analysis of state-of-the-art $\mathbb{F}_q$ information-set decoding from [Pet10] presented in Chapter 6. To find the parameters in Section 7.4 we searched various (n, k, t) and applied the complexity formulas from Chapter 6 to evaluate the security level of each (n, k, t).
Wagner's "generalized birthday attacks" [Wag02b] can also be used as a generic decoding method. The Courtois–Finiasz–Sendrier signature system [CFS01] was attacked by Bleichenbacher using this method. However, information-set decoding is always more efficient than generalized birthday attacks as an attack against code-based encryption. See [FS09] for further discussion; the analysis is essentially independent of q.

7.3.2 Structural and algebraic attacks

The following method aims at detecting the hidden Goppa code in the wild McEliece cryptosystem.

Remark 7.4 (Polynomial-searching attacks)**.** There are approximately q^{mt}/t monic irreducible polynomials g of degree t in $\mathbb{F}_{q^m}[x]$, and therefore approximately q^{mt}/t choices of g^{q-1}. One can marginally expand the space of polynomials by considering more general squarefree polynomials g, but we focus on irreducible polynomials to avoid any unnecessary security questions.
An attacker can try to guess the Goppa polynomial g^{q-1} and then apply Sendrier's "support-splitting algorithm" [Sen00] to compute the support $(a_1, \ldots, a_n)$. We combine two defenses against this attack:

- We keep q^{mt}/t extremely large, so that guessing g^{q-1} has negligible chance of success. Parameters with q^{mt}/t smaller than 2^{128} are marked with the international biohazard symbol ☣ in Section 7.4.

- We keep n noticeably lower than q^m, so that there are many possible subsets $\{a_1, \ldots, a_n\}$ of $\mathbb{F}_{q^m}$. The support-splitting algorithm takes $\{a_1, \ldots, a_n\}$ as an input along with g.

The second defense is unusual: it is traditional, although not universal, to take $n = 2^m$ and $q = 2$, so that the only possible set $\{a_1, \ldots, a_n\}$ is $\mathbb{F}_{2^m}$. The strength of the second defense is unclear: we might be the first to ask whether the support-splitting idea can be generalized to handle many sets $\{a_1, \ldots, a_n\}$ simultaneously, and we would not be surprised if the answer turns out to be yes. However, the first defense is well known for $q = 2$ and appears to be strong.

The second attack is an *algebraic attack*. In a recent article [FOPT10], Faugére, Otmani, Perret, and Tillich broke many (but not all) of the "quasi-cyclic" and "quasi-dyadic" variants of the McEliece cryptosystem that had been proposed in the articles [BCGO09] and [MB09] in 2009. Gauthier Umana and Leander in [GL10] independently broke some of the same systems.
These variants have highly regular support structures allowing very short public keys. The attacks set up systems of low-degree algebraic equations for the code support, taking advantage of the fact that there are not many variables in the support.
The article [FOPT10] indicates that the same attack strategy is of no use against the original McEliece cryptosystem because there are "much more unknowns" than in the broken proposals: for example, 1024 variables in $\mathbb{F}_{1024}$, totalling 10240 bits. Our recommended parameters also have very large supports, with no particular structure, so algebraic attacks do not appear to pose any threat.

7.4 Parameters

Table 7.1 gives parameters (n, k, t) for the McEliece cryptosystem using a code $\Gamma = \Gamma_q(a_1, \ldots, a_n, g^{q-1})$ that provides 128-bit security against our attack presented in Chapter 6. We chose the code length n, the degree t of g and the dimension $k = n - \lceil \log_q n \rceil t(q-1)$ of Γ to minimize the key size $\lceil (n-k)k \log_2 q \rceil$ for 128-bit security when w errors are added. We compare four cases:

- $w = \lfloor (q-1)t/2 \rfloor$ added errors using classical decoding techniques,
- $w = \lfloor qt/2 \rfloor$ added errors using Theorem 7.1,
- $w = \lfloor qt/2 \rfloor + 1$ added errors, and
- $w = \lfloor qt/2 \rfloor + 2$ added errors,

where the last two cases use Theorem 7.1 together with list decoding as in Section 7.2. See Figure 7.1 for a graph of the resulting key sizes; also compare it to Figure 6.1 in Section 6.3.2.
Section 5.2.5 proposed to use a Goppa code $\Gamma_2(a_1, \ldots, a_n, g)$ with length 2960, dimension 2288, and g of degree $t = 56$ for 128-bit security when 57 errors are added by the sender. A key in this setting has 1537536 bits. This is consistent with our table entry for $q = 2$ with $w = \lfloor qt/2 \rfloor + 1$ added errors.

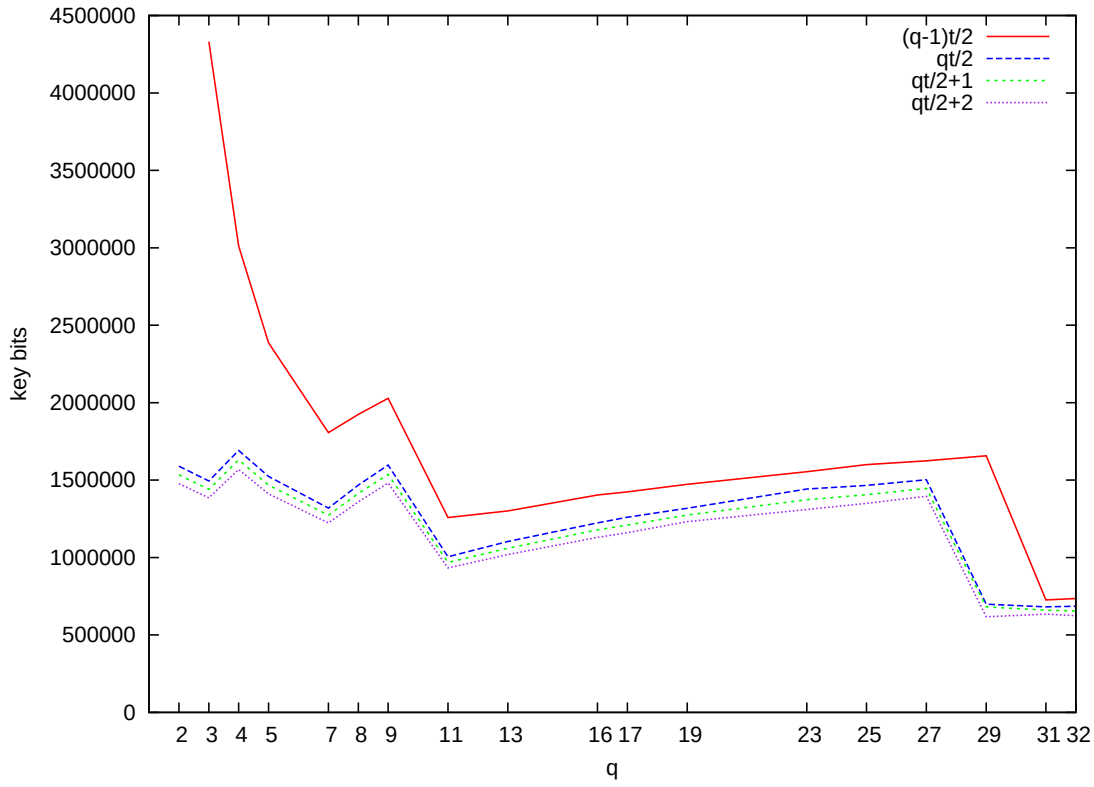

Figure 7.1: Decrease in key sizes when correcting more errors (128-bit security). Horizontal axis is the field size q. Graphs from top to bottom show the best key size when $\lfloor (q-1)t/2 \rfloor$ errors are added; $\lfloor qt/2 \rfloor$ errors are added; $\lfloor qt/2 \rfloor + 1$ errors are added; $\lfloor qt/2 \rfloor + 2$ errors are added. See Table 7.1.

Small q's larger than 2 provide astonishingly good results. For larger q's one has to be careful: parameters optimized against information-set decoding have q^{mt}/t dropping as q grows, reducing the number of suitable polynomials g in $\mathbb{F}_{q^m}[x]$ significantly. For example, there are only about 2^{28} monic irreducible polynomials g of degree 3 over $\mathbb{F}_{31^2}[x]$, while there are about 2^{227} monic irreducible polynomials g of degree 20 in $\mathbb{F}_{5^5}[x]$. The smallest q for which the g possibilities can be enumerated in less time than information-set decoding is $q = 11$: the parameters $(n, k, t) = (1199, 899, 10)$ satisfy $q^{\lceil \log_q n \rceil t}/t \approx 2^{100}$, so there are about 2^{100} monic irreducible polynomials g in $\mathbb{F}_{11^3}[x]$ of degree $t = 10$. This is one of the cases marked by ☣ in Table 7.1. The security of these cases depends on the strength of the second defense discussed in Section 7.3.

The ☣ symbol is omitted from the $\lfloor (q-1)t/2 \rfloor$ column because relatively low error-correcting capability and relatively high key size can be achieved by non-wild codes with many more choices of g.

Table 7.1: Decrease in key sizes when correcting more errors (128-bit security). Each entry in the first column states q. Each entry in the subsequent columns states key size, (n, k, t) and the number of errors.

q	$\lfloor (q-1)t/2 \rfloor$	$\lfloor qt/2 \rfloor$	$\lfloor qt/2 \rfloor + 1$	$\lfloor qt/2 \rfloor + 2$
2	—	1590300 bits: (3009, 2325, 57) 57 errors	1533840 bits: (2984, 2324, 55) 56 errors	1477008 bits: (2991, 2367, 52) 54 errors
3	4331386 bits: (3946, 3050, 56) 56 errors	1493796 bits: (2146, 1530, 44) 66 errors	1439876 bits: (2133, 1545, 42) 64 errors	1385511 bits: (2121, 1561, 40) 62 errors
4	3012336 bits: (2886, 2202, 38) errors 57	1691424 bits: (2182, 1678, 28) errors 56	1630044 bits: (2163, 1677, 27) 55 errors	1568700 bits: (2193, 1743, 25) 52 errors
5	2386014 bits: (2395, 1835, 28) 56 errors	1523278 bits: (1931, 1491, 22) 55 errors	1468109 bits: (1877, 1437, 22) 56 errors	1410804 bits: (1919, 1519, 20) 52 errors
7	1806298 bits: (1867, 1411, 19) 57 errors	1319502 bits: (1608, 1224, 16) 56 errors	1273147 bits: (1565, 1181, 16) 57 errors	1223423 bits: (1633, 1297, 14) 51 errors
8	1924608 bits: (1880, 1432, 16) 56 errors	1467648 bits: (1640, 1248, 14) 56 errors	1414140 bits: (1659, 1295, 13) 53 errors	1359540 bits: (1609, 1245, 13) 54 errors
9	2027941 bits: (1876, 1428, 14) 56 errors	1597034 bits: (1696, 1312, 12) 54 errors	1537389 bits: (1647, 1263, 12) 55 errors	1481395 bits: (1601, 1217, 12) 56 errors
11	1258265 bits: (1286, 866, 14) 70 errors	1004619 bits: (1268, 968, 10)☣ 55 errors	968295 bits: (1233, 933, 10)☣ 56 errors	933009 bits: (1199, 899, 10)☣ 57 errors
13	1300853 bits: (1409, 1085, 9) 54 errors	1104093 bits: (1324, 1036, 8)☣ 52 errors	1060399 bits: (1283, 995, 8)☣ 53 errors	1018835 bits: (1244, 956, 8)☣ 54 errors
16	1404000 bits: (1335, 975, 8) 60 errors	1223460 bits: (1286, 971, 7)☣ 56 errors	1179360 bits: (1251, 936, 7)☣ 57 errors	1129680 bits: (1316, 1046, 6)☣ 50 errors
17	1424203 bits: (1373, 1037, 7) 56 errors	1260770 bits: (1359, 1071, 6)☣ 51 errors	1208974 bits: (1315, 1027, 6)☣ 52 errors	1160709 bits: (1274, 986, 6)☣ 53 errors
19	1472672 bits: (1394, 1070, 6) 54 errors	1318523 bits: (1282, 958, 6)☣ 57 errors	1274481 bits: (1250, 926, 6)☣ 58 errors	1231815 bits: (1219, 895, 6)☣ 59 errors

Continued on next page

Table 7.1 – continued from previous page

q	$\lfloor (q-1)t/2 \rfloor$	$\lfloor qt/2 \rfloor$	$\lfloor qt/2 \rfloor + 1$	$\lfloor qt/2 \rfloor + 2$
23	1553980 bits: (1371, 1041, 5) 55 errors	1442619 bits: (1472, 1208, 4)☣ 46 errors	1373354 bits: (1414, 1150, 4)☣ 47 errors	1310060 bits: (1361, 1097, 4)☣ 48 errors
25	1599902 bits: (1317, 957, 5) 60 errors	1465824 bits: (1384, 1096, 4)☣ 50 errors	1405640 bits: (1339, 1051, 4)☣ 51 errors	1349468 bits: (1297, 1009, 4)☣ 52 errors
27	1624460 bits: (1407, 1095, 4) 52 errors	1502811 bits: (1325, 1013, 4)☣ 54 errors	1446437 bits: (1287, 975, 4)☣ 55 errors	1395997 bits: (1253, 941, 4)☣ 56 errors
29	1656766 bits: (1351, 1015, 4) 56 errors	699161 bits: (794, 514, 5)☣ 72 errors	681478 bits: (781, 501, 5)☣ 73 errors	617003 bits: (791, 567, 4)☣ 60 errors
31	726484 bits: (851, 611, 4) 60 errors	681302 bits: (813, 573, 4)☣ 62 errors	659899 bits: (795, 555, 4)☣ 63 errors	634930 bits: (892, 712, 3)☣ 48 errors
32	735320 bits: (841, 593, 4) 62 errors	685410 bits: (923, 737, 3)☣ 48 errors	654720 bits: (890, 704, 3)☣ 49 errors	624960 bits: (858, 672, 3)☣ 50 errors

Chapter 8

Ball-collision decoding

The generic decoding algorithms described in Chapter 5 take exponential time for any constant asymptotic code rate R and constant asymptotic error fraction W: i.e., time $2^{(\alpha(R,W)+o(1))n}$ for some positive real number $\alpha(R,W)$ if $k/n \to R$ and $w/n \to W$ as $n \to \infty$. As before, n denotes the code length, k the code dimension, and w the number of errors. Section 5.4 introduced $\alpha(R,W)$ and in particular how $o(1)$ looks like in the case of information-set decoding as described by Lee–Brickell and Stern.

The attack described in Section 5.2 established an upper bound on the optimal decoding exponent $\alpha(R,W)$. The upper bound is the exponent of Stern's collision-decoding algorithm 5.2. This upper bound arises from an asymptotic binomial-coefficient optimization and does not have a simple formula, but it can be straightforwardly computed to high precision for any particular (R,W). For example, for $W = 0.04$ and $R = 1 + W\log_2 W + (1-W)\log_2(1-W) = 0.7577\ldots$, Stern's algorithm shows that $\alpha(R,W) \leq 0.0809\ldots$.

There have also been many polynomial-factor speedups in generic decoding algorithms; there are dozens of articles on this topic, both inside and outside cryptography. Here is an illustration of the cumulative impact of many of the speedups. McEliece's original parameter suggestions ("$n = 1024, k = 524, t = 50$") take about $524^3\binom{1024}{50}/\binom{500}{50} \approx 2^{81}$ operations to break by the simple information-set-decoding attack explained in McEliece's original article [McE78, Section 3]. (McEliece estimated the attack cost as $524^3(1-50/1024)^{-524} \approx 2^{65}$; this underestimate was corrected by Adams and Meijer in [AM87, Section 3].) The attack presented in Section 5.2, thirty years after McEliece's article, builds on several improvements and takes only about $2^{60.5}$ operations for the same parameters. This thesis also discussed more recent improvements such as the bound in (5.1) and q-ary information-set decoding in Chapter 6. However, most improvements give only polynomial factors which have no relevance to the exponent $\alpha(R,W)$ in $2^{(\alpha(R,W)+o(1))n}$. The best known upper bound on $\alpha(R,W)$ has been unchanged since 1989.

This chapter presents smaller upper bounds on the decoding exponent $\alpha(R,W)$. Specifically, this chapter introduces a generic decoding algorithm and demonstrates that this algorithm is, for every (R,W), faster than Stern's algorithm by a factor exponential in n. This algorithm is called "ball-collision decoding" because of a geometric interpretation explained in Section 8.1.1.

The change in the exponent is not very large — for example, this chapter uses ball-

collision decoding to demonstrate that $\alpha(R,W) \leq 0.0807\ldots$ for the (R,W) given above — but it is the first exponential improvement in decoding complexity in more than twenty years.

Constant rates R and constant error fractions W are traditional in the study of coding-theory asymptotics. Note that they are not exactly right in the study of code-based cryptography. Typical code-based cryptosystems limit the error fraction W to $(1-R)/\log_2 n$, so W decreases slowly to 0 as $n \to \infty$. (In theory one could replace the Goppa codes in the McEliece system with explicit "asymptotically good" families of codes for which k/n and w/n converge to nonzero constants R and W; however, the security of these families has never been evaluated.) It seems reasonable to conjecture that the best possible algorithms take time

$$(1-R)^{-(1-R)n/\log_2 n+(\gamma(R)+o(1))n/(\log_2 n)^2}$$

if $k/n \to R$ and $w/(n/\log_2 n) \to 1-R$ as $n \to \infty$; this was discusses in Section 5.4. Ball-collision decoding produces better upper bounds on $\gamma(R)$ for the same reason that it produces better upper bounds on $\alpha(R,W)$. This chapter considers the two-variable function $\alpha(R,W)$ so that the results can be applied to a wider range of choices of W and compared to a wider range of previous articles, including articles that make the traditional coding-theory assumption $R = 1 + W\log_2 W + (1-W)\log_2(1-W)$.

This chapter also evaluates the exact cost of ball-collision decoding, using the same bit-operation-counting rules as in the previous literature, and uses this evaluation to illustrate the impact of ball-collision decoding upon cryptographic applications.

The results of this chapter are joint work with Bernstein and Lange and appeared in [BLP10]. The main differences between the content of this chapter and [BLP10] are the following.

- The ball-collision-decoding algorithm is presented in a simplified way compared to [BLP10] in order to unify all information-set-decoding algorithms in this thesis.
- The complexity analysis is shortened in comparison to Sections 3 and 5 in [BLP10] as several methods have been used already in Chapter 5 and 6.
- This chapter omits the discussion of long-term cryptography given in Appendix A of [BLP10].
- Section 8.3 combines Section 7 and the Appendix C of [BLP10], giving the asymptotic analysis of the cost of ball-collision decoding and the proof of its asymptotic superiority to collision decoding.

8.1 The ball-collision algorithm

Given a parity-check matrix H and a syndrome s, Algorithm 8.1 looks for a weight-w word $e \in \mathbb{F}_2^n$ such that $He^t = s$.

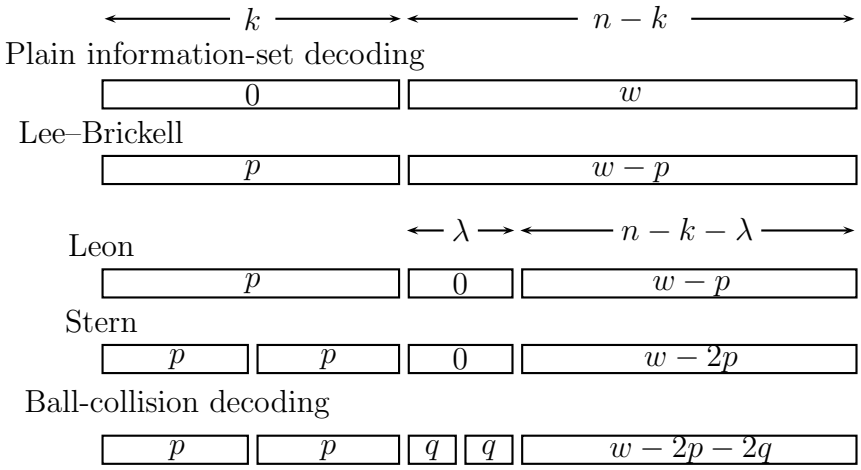

Figure 8.1: Extension of Figure 5.2 to include ball-collision decoding. Note that $\lambda = \lambda_l + \lambda_r$ for ball-collision decoding and $\lambda = \ell$ for Leon's algorithm and for Stern's algorithm.

8.1.1 The algorithm

This section presents the ball-collision decoding algorithm in a simplified way; the next section discusses various optimizations.

Ball-collision decoding builds on collision decoding, i.e., Stern's algorithm, which searches for collisions between vectors in $\mathbb{F}_2^\ell$. Any such vector $v \in \mathbb{F}_2^\ell$ is the sum of p columns of H restricted to ℓ rows. The idea of ball-collision decoding is to look for collisions among vectors in $\mathbb{F}_2^\ell$ at distance q from such a vector v. These vectors lie in the ball around v with radius q. The algorithm could easily be extended to search for collisions between all elements within these balls. For simplicity the algorithm is restricted to the vectors on the boundary.

Similar to collision decoding a randomly chosen information set I is divided into two disjoint subsets X and Y. The ball-collision algorithm finds a weight-w vector e with $He^t = s$ if an information set I together with sets X, Y, Z_l, and Z_r can be found such that e has weights p, p, q, q on the positions indexed by X, Y, Z_l, Z_r, respectively. Recall the visual comparison by Overbeck and Sendrier [OS08] of error patterns in various information-set-decoding algorithms (Figure 5.2 in Section 5.1.1). Figure 8.1 extends their picture to include ball-collision decoding. It shows that the new algorithm allows errors in an interval that had to be error-free in Leon's and Stern's algorithms.

For a visual interpretation of Algorithm 8.1 we refer the reader to Remark 8.1 and in particular to Figure 8.2. The algorithm has a parameter $p \in \{0, 1, \ldots, w\}$, a parameter $q \in \{0, 1, \ldots, w - 2p\}$, and parameters $\lambda_l, \lambda_r \in \{0, 1, \ldots, n - k\}$. The algorithm consists of a series of independent iterations, each iteration making random choices. If the set I chosen in Step 1 does not lead to a weight-w word in Step 10 another iteration has to be performed. Each iteration consists of the steps described

in Algorithm 8.1.

Algorithm 8.1: Ball-collision decoding

Input: An integer $w \geq 0$, a column vector $s \in \mathbb{F}_2^{n-k}$, and an $(n-k) \times n$ parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$.

Output: A weight-w element $e \in \mathbb{F}_2^n$ with $He^t = s$.

1: Choose an information set I and bring H into systematic form with respect to I: find an invertible $U \in \mathbb{F}_2^{(n-k)\times(n-k)}$ such the submatrix of UH indexed by $\{1,\ldots,n\} \setminus I$ is the $(n-k) \times (n-k)$ identity matrix.
2: Select a uniform random size-$\lfloor k/2 \rfloor$ subset $X \subset I$.
3: Define $Y = I \setminus X$.
4: Select a size-λ_l subset Z_l of $\{1,\ldots,n\} \setminus I$.
5: Select a size-λ_r subset Z_r of $\{1,\ldots,n\} \setminus (I \cup Z_l)$.
6: **For each** *size-p subset* $A_0 \subseteq X$*:* compute $\phi(A_0) = s + \sum_{a_0}(UH)_{a_0}$ restricted to the $\lambda_l + \lambda_r$ rows indexed by $Z_l \cup Z_r$. **For each** *size-q subset* $A_1 \subseteq Z_l$*:* compute $\phi'(A_0, A_1) = s + \sum_{i \in A_0 \cup A_1}(UH)_i$ restricted to the $\lambda_l + \lambda_r$ rows indexed by $Z_l \cup Z_r$; i.e., expand $\phi(A_0)$ into a ball of radius q by computing $\phi'(A_0, A_1) \in \mathbb{F}_2^{\lambda_l+\lambda_r}$ which is the sum of $\phi(A_0)$ and the q columns of UH indexed by A_1, restricted to $\lambda_l + \lambda_r$ rows.
7: **For each** *size-p subset* $B_0 \subseteq Y$*:* compute $\psi(B_0) = \sum_{b_0}(UH)_{b_0}$ restricted to $\lambda_l + \lambda_r$ rows indexed by $Z_l \cup Z_r$. **For each** *size-q subset* $B_1 \subseteq Z_r$*:* compute $\psi'(B_0, B_1) = \sum_{i \in B_0 \cup B_1}(UH)_i$ restricted to the $\lambda_l + \lambda_r$ rows indexed by $Z_l \cup Z_r$; i.e., expand $\psi(B_0)$ into a ball of radius q by computing $\psi'(B_0, B_1) \in \mathbb{F}_2^{\lambda_l+\lambda_r}$ which is the sum of $\phi(B_0)$ and q columns of UH indexed by B_1, restricted to $\lambda_l + \lambda_r$ rows.
8: **For each** 4*-tuple* (A_0, A_1, B_0, B_1) *such that* $\phi'(A_0, A_1) = \psi'(B_0, B_1)$*:*
9: Compute $s' = s + \sum_{i \in A_0 \cup A_1 \cup B_0 \cup B_1}(UH)_i$.
10: **If** $\mathrm{wt}(s') = w - 2p - 2q$ **then** add the corresponding $w - 2p - 2q$ columns in the $(n-k) \times (n-k)$ identity submatrix to s to make s' the all-zero syndrome. **Return** the vector $e \in \mathbb{F}_2^n$ indicating those columns and the columns indexed by $A_0 \cup A_1 \cup B_0 \cup B_1$.
11: Go back to Step 1.

Algorithm 8.1 was developed in joint work with Bernstein and Lange and published as [BLP10].

Remark 8.1. Figure 8.2 gives a visualization of one iteration of ball-collision decoding for the case $s = 0$. Without loss of generality assume that I indexes the first k columns of UH which then can be written as $Q = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix}$ with $Q_1 \in \mathbb{F}_2^{(\lambda_l+\lambda_r)\times k}$, $Q_2 \in \mathbb{F}_2^{(n-k-(\lambda_l+\lambda_r))\times k}$. The set X indexes columns $1,\ldots,k/2$ and Y indexes columns $k/2+1,\ldots,k$. The set Z_l indexes columns $k+1,\ldots,k+\lambda_l$ and Z_r indexes columns $k+\lambda_l+1,\ldots,k+\lambda_l+\lambda_r$. Note that $Z_l \cup Z_r$ also indexes the first $\lambda_l+\lambda_r$ rows of UH;

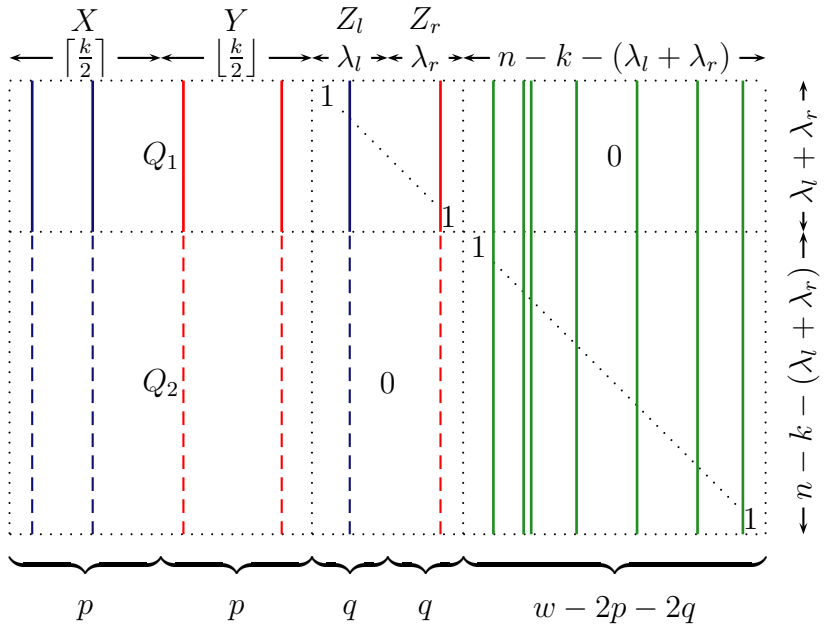

Figure 8.2: The ball-collision-decoding algorithm. The weights p, p, q, q, and $w-2p-2q$ indicate how many columns are taken from the respective parts. Also compare to Figure 5.1 in Section 5.1.2.

in particular, $Z_l \cup Z_r$ indexes the rows of Q_1. The split into Z_l and Z_r is important for the column selection, not when considering rows. We refer to columns restricted to positions indexed by $Z_l \cup Z_r$ as *partial columns* and similarly call the syndrome restricted to $\lambda_l + \lambda_r$ positions *partial syndrome*.
The algorithm first builds sums of partial columns coming from p columns of Q_1 indexed by X and q partial columns coming from columns indexed by Z_l. Then the algorithm builds sums of partial columns coming from p columns of Q_1 indexed by Y and q partial columns coming from columns indexed by Z_l. If a collision occurs between those sums, then the sum of the $2p+2q$ columns is computed on all $n-k$ positions and the weight is checked; if s is the sum of $2p+2q$ columns having weight $w-2p-2q$ then those nonzero positions come from the rows corresponding to Q_2 and can be cancelled by $w-2p-2q$ ones in the identity matrix on the right which have indices outside $Z_l \cup Z_r$.

8.1.2 Relationship to previous algorithms

Stern's algorithm, which is called *collision decoding* in this context, is the special case $q = 0$ of ball-collision decoding; Stern's parameter ℓ corresponds to $\lambda_l + \lambda_r$ in Algorithm 8.1. The name "collision decoding" was introduced together with the ball-collision-decoding algorithm in [BLP10]. Algorithm 8.1 was inspired by one of the steps of "supercode decoding" which was introduced by Barg, Krouk, and van Tilborg in [BKvT99].

From the perspective of ball-collision decoding, the fundamental disadvantage of collision decoding is that errors are required to avoid an asymptotically quite large stretch of ℓ positions ($\lambda_l + \lambda_r$ in the ball-collision setting). Ball-collision decoding makes a much more reasonable hypothesis, namely that there are asymptotically increasingly many errors in those positions. It requires extra work to enumerate the points in each ball, but the extra work is only about the square root of the improvement in success probability. The cost ratio is asymptotically superpolynomial as will be discussed in Section 8.3.

Collision decoding also has a more superficial disadvantage compared to ball-collision decoding: each iteration is slower, since computing $\phi(A_0)$ for a new size-p subset A_0 is considerably more expensive than just adding columns indexed by Z_l. Note that the columns indexed by Z_l contain only a single nonzero entry; adding such a column means only flipping a bit. The cost ratio here is only polynomial, and is not relevant to the asymptotic analysis (see Section 8.3), but is accounted for in the bit-operation count (see Section 8.1.3).

The idea of allowing errors everywhere can be extracted, with considerable effort, from [BKvT99]. After a detailed analysis it appeared that the algorithm in [BKvT99] is much slower than collision decoding. The same algorithm is claimed in [BKvT99] to have smaller exponents than collision decoding (with astonishing gaps, often 15% or more), but this claim is based on a chain of exponentially large inaccuracies in the algorithm analysis in [BKvT99]. The starting point of the chain is [BKvT99, "Corollary 12"], which claims size $\binom{k}{e_1}\binom{y}{e_2}/2^{by}$ for lists that actually have size $\binom{k}{e_1}\binom{y}{e_2}^b/2^{by}$.

The idea of allowing errors everywhere can also be found in the much more recent article [FS09], along with a polynomial-factor "birthday" speedup which was discussed in Section 5.3.2. The algorithm analysis by Finiasz and Sendrier in [FS09] concludes that the overall "gain compared with Stern's algorithm" is a constant times "$\sqrt[4]{\pi p/2}$", which is bounded by a polynomial in n. This is the improvement coming from the birthday speedup. However, if parameters had been chosen more carefully then the algorithm of [FS09] would have led to an exponential improvement over collision decoding, contrary to the conclusions in [FS09]. This algorithm would still have retained the secondary disadvantage described above, and therefore would not have been competitive with ball-collision decoding.

Note that the birthday speedup as described in Section 5.3.2 can be adapted to ball-collision decoding but would complicate the algorithm statement and analysis without changing the exponent of binary decoding. These modifications are therefore skipped for reasons of simplicity.

The algorithm as described in [BLP10, Section 3] has more flexible parameters. The next remark discusses the parameter setup given in [BLP10, Section 3] and discusses advantages and disadvantages.

Remark 8.2 (Parameter balance)**.** The algorithm in [BLP10, Section 3] splits an information set I into two disjoint subsets X and Y of cardinality k_1 and k_2 where $k = k_1 + k_2$. The algorithm looks for a word e having weight p_1, p_2, q_1, q_2 on the positions indexed by X, Y, Z_l, and Z_r, respectively. In the asymptotic analysis of the

cost of ball-collision decoding (see Section 8.3) balanced parameters always turned out to be asymptotically optimal (as one would expect). Of course, asymptotic optimality of $p_1 = p_2$ does not imply the concrete optimality of $p_1 = p_2$. There are examples with small input sizes where $(p_1, p_2) = (2, 1)$ appears to be optimal as shown in [BLP10]. For more interesting examples the operation counts in [BLP10] always produced balanced parameters, i.e., $k_1 = k_2$ (assuming that k is even), $\lambda_l = \lambda_r$, $p_1 = p_2$, and $q_1 = q_2$; see Section 8.1.4.

8.1.3 Complexity analysis

Many of the techniques for fast collision decoding can be adapted for ball-collision decoding. This section analyzes the complexity of fast ball-collision decoding. In particular, this section analyzes the success probability of each iteration and gives the total number of bit operations needed for each iteration. For the sake of simplicity assume that k is even.

The ball-collision algorithm consists like collision decoding of three steps; "updating the matrix G with respect to a new information set I", "list building", and "collision handling." The speedups are the ones described in Sections 5.2.1 and 5.2.2. The difference lies in the "list-building step" because many more vectors are built here than in the corresponding step for collision decoding.

In this chapter a quite naive form of Gaussian elimination is sufficient, taking $(1/2)(n-k)^2(n+k)$ bit operations; the interest is in large input sizes, and Gaussian elimination takes negligible time for those sizes.

Ball-collision decoding can also make use of Finiasz and Sendrier's birthday speedup described in Section 5.3.2. For reasons of simplicity the information set is split into disjoint sets X and Y.

Remark 8.3 (Reusing intermediate sums)**.** The vectors $\phi(A_0) \in \mathbb{F}_2^{\lambda_l+\lambda_r}$ are sums of a partial syndrome s and p partial columns of UH indexed by a size-p subset $A_0 \subset X$. Using intermediate sums the cost of summing up p columns on $\lambda_l + \lambda_r$ positions amounts to $(\lambda_l + \lambda_r)\,(L(k/2, p) - k/2)$ where $L(k, p) = \sum_{i=1}^{p} \binom{k}{i}$ is used again; compare to Remark 5.10. Adding $\lambda_l + \lambda_r$ bits of the syndrome s is done as described in Remark 6.3; each $s + \sum_i (UH)_{a_0}$ (restricted to $\lambda_l + \lambda_r$ positions) includes at least one vector $s + (UH)_i$ for $(n-k)/2 - p + 1$ columns $(UH)_i$ with $i \in X$. The total cost for each vector $\phi(A_0)$ amounts to $(\lambda_l + \lambda_r)\,(L(k/2, p) - k + n/2 - p + 1)$ bit operations.

Then, for each $\phi(A_0)$ all $\binom{\lambda_l}{q}$ possible q columns restricted to positions indexed by $Z_l \cup Z_r$ are added to compute vectors $\phi'(A_0, A_1) \in \mathbb{F}_2^{\lambda_l+\lambda_r}$. These extra columns are in fact part of the identity submatrix and thus contain only a single 1 each. Again intermediate sums can be used, so this step takes $\min\{1, q\}\binom{k/2}{p} L(\lambda_l, q)$ bit operations for all $\phi'(A_0, A_1)$ together. Note that for $q = 0$ the cost of this step is indeed 0. Each choice of (A_0, A_1) adds one element to the hash table built in Step 6. Hence, the number of elements in this table equals exactly the number of choices for A_0 and A_1, i.e. $\binom{k/2}{p}\binom{\lambda_l}{q}$. The vectors $\psi'(B_0, B_1) \in \mathbb{F}_2^{\lambda_l+\lambda_r}$ are built similarly.

Note that Step 8 is a standard "join" operation between the set containing vectors $\phi'(A_0, A_1)$ and the set containing vectors $\psi'(B_0, B_1)$; this join can be implemented efficiently by sorting or by list building as done in the collision decoding attack described in Section 5.2.

Remark 8.4 (Checking collisions)**.** The last step does one check for each 4-tuple (A_0, A_1, B_0, B_1) such that $\phi'(A_0, A_1) = \psi'(B_0, B_1)$. There are $\binom{k/2}{p}^2\binom{\lambda_l}{q}\binom{\lambda_r}{q}$ choices of (A_0, A_1, B_0, B_1). If the vectors $\phi'(A_0, A_1)$ and $\psi'(B_0, B_1)$ were uniformly distributed among the $2^{\lambda_l+\lambda_r}$ possible values then on average $2^{-\lambda_l-\lambda_r}\binom{k/2}{p}^2\binom{\lambda_l}{q}\binom{\lambda_r}{q}$ checks would be done. The expected number of checks is extremely close to this for almost all H; as above we disregard the extremely unusual codes with different behavior. Each check consists of computing the weight of $s + \sum_{i\in A_0\cup A_1\cup B_0\cup B_1}(UH)_i$ and testing whether it equals $w - 2p - 2q$. Note that $\lambda_l + \lambda_r$ positions are known to be zero as a result of the collision. The columns indexed by A_1 and B_1 are zero on all bits in $\{1, \ldots, n-k\} \setminus (Z_l \cup Z_r)$. So, in fact, only the sum of $2p$ columns indexed by A_0 and B_0 and the syndrome need to be computed. When using the early-abort weight calculation (Section 5.2.2), on average only $2(w - 2p - 2q + 1)$ rows of the resulting sum are computed before the weight is found too high.

Remark 8.5 (Success probability)**.** Assume that e is a uniform random vector of weight w. One iteration of ball-collision decoding finds e exactly if it has the right weight distribution, namely weight p on $k/2$ positions indexed by a set X, weight p on $k/2$ positions specified by Y, weight q on λ_l positions outside the information set $I = X \cup Y$, and weight q on another set of λ_r positions not indexed by I. The probability that e has this particular weight distribution is, by a simple counting argument, exactly

$$b(p, q, \lambda_l, \lambda_r) = \binom{n}{w}^{-1}\binom{k/2}{p}^2\binom{\lambda_l}{q}\binom{\lambda_r}{q}\binom{n-k-(\lambda_l+\lambda_r)}{w-2p-2q}.$$

The expected number of iterations is, for almost all H, very close to the reciprocal of the success probability of a single iteration. We explicitly disregard, without further comment, the extremely unusual codes for which the average number of iterations is significantly different from the reciprocal of the success probability of a single iteration. For further discussion of this issue and how unusual it is see, e.g., [CGF91] and [BLPvT09].

The *total cost* per iteration of one iteration with parameters p, q, λ_l, and λ_r amounts to

$$\begin{aligned} c(p, q, \lambda_l, \lambda_r) =& \frac{1}{2}(n-k)^2(n+k) \\ &+ \big(2L(k/2, p) + n/2 - (3/2)k - p + 1\big)(\lambda_l + \lambda_r) \\ &+ \min\{1, q\}\binom{k/2}{p}(L(\lambda_l, q) + L(\lambda_r, q)) \\ &+ 2(w - 2p - 2q + 1)(2p)\binom{k/2}{p}^2\binom{\lambda_l}{q}\binom{\lambda_r}{q}2^{-\lambda_l-\lambda_r}. \end{aligned}$$

8.1.4 Concrete parameter examples

This section considers concrete examples in order to show the speedup gained by ball-collision decoding in comparison to collision decoding. The first parameters were proposed in Section 5.2.5 to achieve 256-bit security against current attacks. The second parameters are designed according to similar rules to achieve a 1000-bit security level against current attacks. Of course, 1000-bit security is not suggested to be of any real-world relevance; this section considers it to demonstrate the asymptotic superiority of ball-collision decoding.
Finiasz and Sendrier in [FS09] presented "lower bounds on the effective work factor of existing real algorithms, but also on the future improvements that could be implemented"; and said that beating these bounds would require the introduction of "new techniques, never applied to code-based cryptosystems". For each set of parameters the Finiasz–Sendrier lower bound and the costs of three algorithms is evaluated:

(1) collision decoding ($q_1 = q_2 = 0$),

(2) collision decoding using the birthday trick from [FS09] as analyzed in Section 5.3.2, and

(3) ball-collision decoding.

Ball-collision decoding achieves complexities lying below the Finiasz–Sendrier lower bound in both of these examples. The main reason for this is that ball-collision decoding dodges the secondary disadvantage described in Section 8.1.2; the lower bound assumes that each new vector requires $\ell_1 + \ell_2$ bit operations to update $\phi(A)$, but in ball-collision decoding each new vector requires just 1 bit operation to compute a new $\phi'(A_0, A_1)$.
All of these costs and bounds use the same model of computation, counting the number of bit operations for arithmetic and disregarding costs of memory access, copies, etc. A table-indexing join operation can easily be carried out for free in this model. In general it would be desirable to have a more carefully defined model of computation that includes realistic memory-access costs, such as the Brent–Kung circuit model [BK81], but the bit-operation model is simpler and is standard in articles on this topic. Note that any reasonable accounting for memory-access costs would need at least one memory access for each new $\phi'(A_0, A_1)$ in ball-collision decoding (for the join) but would need at least *two* memory accesses for each new $\phi(A)$ in the Finiasz–Sendrier lower bound (one for A and one for the join).

Remark 8.6 (256-security revisited)**.** According to Section 5.2.5 a binary code with length $n = 6624$, $k = 5129$, $w = 117$ achieves 256-bit security. see also Section 5.2.5. The best collision-decoding parameters are actually slightly below 2^{256} bit operations: they use $2^{181.4928}$ iterations (on average), each taking $2^{74.3741}$ bit operations, for a total of $2^{255.8669}$ bit operations.
Collision decoding with the birthday trick takes, with optimal parameters, $2^{255.54880}$ bit operations. The birthday trick increases the cost per iteration by a factor of

2.2420 compared to the classical collision-decoding algorithm, to $2^{75.5390}$ bit operations. However, the trick increases the chances of finding the desired error vector noticeably, reducing the number of iterations by a factor of 2.7951, to $2^{180.0099}$. Thus the birthday trick yields an overall 1.2467× speedup.
The Finiasz–Sendrier lower bound is $2^{255.1787}$ bit operations, 1.6112× smaller than the cost of collision decoding.
Ball-collision decoding with parameters $\lambda_l = \lambda_r = 47$, $p = 8$, and $q = 1$ needs only $2^{254.1519}$ bit operations to attack the same system. On average the algorithm needs $2^{170.6473}$ iterations each taking $2^{83.504570}$ bit operations.
Ball-collision decoding thus costs 3.2830× less than collision decoding, 2.6334× less than collision decoding with the birthday trick, and 2.0375× less than the Finiasz–Sendrier lower bound.

Remark 8.7 (1000-bit security)**.** Attacking a system based on a code of length $n = 30332$, $k = 22968$, $w = 494$ requires $2^{1000.9577}$ bit operations using collision decoding with the optimal parameters $\lambda_l = \lambda_r = 140$, $p = 27$ and $q = 0$.
The birthday trick reduces the cost by a factor of 1.7243, to $2^{1000.1717}$ bit operations. This means that this system offers 1000-bit security against all previously known attacks.
The Finiasz–Sendrier lower bound is $2^{999.45027}$ bit operations, 2.8430× smaller than the cost of collision decoding and 1.6488× smaller than the cost of collision decoding with the birthday trick.
Ball-collision decoding with parameters $\lambda_l = \lambda_r = 156$, $p = 29$, and $q = 1$ needs only $2^{996.21534}$ bit operations. This is 26.767× smaller than the cost of collision decoding, 15.523× smaller than the cost of collision decoding with the birthday trick, and 9.415× smaller than the Finiasz–Sendrier lower bound.

8.2 Choosing McEliece parameters: a new bound

The traditional approach to selecting cryptosystem parameters is as follows:

- Consider the fastest known attacks against the system. For example, in the case of RSA, consider the latest refinements [KAF+10] of the number-field sieve.
- Restrict attention to parameters for which these attacks take time at least $2^{b+\delta}$. Here b is the desired security level, and δ is a "security margin" meant to protect against the possibility of further improvements in the attacks.
- Within the remaining parameter space, choose the most efficient parameters. The definition of efficiency depends on the target application: it could mean minimal key size, for example, or minimum decryption time.

This approach does not make clear how to choose the security margin δ. Some applications have ample time and space for cryptography, and can simply increase

δ to the maximum value for which the costs of cryptography are still insignificant; but in some applications cryptography is an important bottleneck, and users insist on minimizing δ for the sake of performance.

Finiasz and Sendrier in [FS09] identified a bound on "future improvements" in attacks against the McEliece cryptosystem, and suggested that designers use this bound to "choose durable parameters". The general idea of identifying bottlenecks in any possible attack, and of using those bottlenecks to systematically choose δ, is quite natural and attractive, and has been used successfully in many contexts. However, ball-collision decoding disproves the specific bound in [FS09], leaving open the question of how the general idea can be applied to the McEliece cryptosystem.

Instead of using the bound in [FS09] one should use the simpler bound

$$\min\left\{\frac{1}{2}\binom{n}{w}\binom{n-k}{w-p}^{-1}\binom{k}{p}^{-1/2} : p \geq 0\right\};$$

i.e., choosing the code length n, code rate k/n, and error fraction w/n so that this bound is at least 2^b. As usual, implementors can exploit the remaining flexibility in parameters to optimize decryption time, compressed key size $k(n-k)$, or efficiency in any other metric of interest.

This bound has several attractive features. It is easy to estimate via standard binomial-coefficient approximations. It is easy to compute exactly. It covers a very wide class of attacks, as will be explained in a moment. It is nevertheless in the same ballpark as the cost of known attacks: for example, it is $2^{49.69}$ for the original parameters $(n,k,w) = (1024,524,50)$, and $2^{236.49}$ for $(n,k,w) = (6624,5129,117)$. Note that these numbers give lower bounds on the cost of the attack. Parameters protecting against this bound pay only about a 20% performance penalty at high security levels, compared to parameters that merely protect against known attacks. The reader can easily verify that parameters $(n,k,w) = (3178,2384,68)$ achieve 128-bit security against this bound. For 256-bit security $(n,k,w) = (6944,5208,136)$ are recommended.

Here is the class of attacks mentioned above. Assume that each iteration of the attack chooses an information set, hoping for exactly p errors in the set; that the choices of information sets are independent of the target syndrome; that each iteration considers at least $\binom{k}{p}^{1/2}$ error patterns within the information set; and that testing each pattern costs at least 1. The $\binom{k}{p}^{1/2}$ iterations model the cost of a birthday-type attack on all vectors of length k with Hamming weight p.

For each $\epsilon \geq 0$, a cost bound of $\epsilon\binom{n}{w}\binom{n-k}{w-p}^{-1}\binom{k}{p}^{-1/2}$ allows at most $\epsilon\binom{n}{w}\binom{n-k}{w-p}^{-1}\binom{k}{p}^{-1}$ iterations, and each iteration covers at most $\binom{n-k}{w-p}\binom{k}{p}$ patterns of w errors, so overall the iterations cover at most $\epsilon\binom{n}{w}$ possible patterns; i.e., the attack succeeds with probability at most ϵ. The average attack time is therefore at least $\frac{1}{2}\binom{n}{w}\binom{n-k}{w-p}^{-1}\binom{k}{p}^{-1/2}$. Note that batching attacks, i.e., attacking multiple targets at once, does not provide any benefits in this approach. Thus the Johansson–Jonsson

speedups for attacking batches of McEliece ciphertexts [JJ02] are subject to the same bound, as are the Fossorier–Kobara–Imai speedups [FKI07].
One can object that this class does not include, e.g., attacks that hope for *at most* p errors in the information set, or attacks that consider fewer error patterns per iteration at the expense of success probability. One can object, in the opposite direction, that the conditional success probability per error pattern inspected is actually a constant factor smaller than the $\binom{k}{p}^{-1/2}$ hypothesized above; see generally [FS09, Appendix A]. A more complicated bound that accounts for these variations and limitations would be slightly larger than the bound stated above but would also be more difficult to compute; a simpler, slightly smaller bound is more useful. In any event, it is clear that beating this bound would be an astonishing breakthrough.

8.3 Asymptotic analysis

As in Section 5.4 we consider codes and error vectors of very large length n, where the codes have dimension $k \approx Rn$, and the error vectors have weight $w \approx Wn$. More precisely, fix functions $k, w : \{1, 2, \ldots\} \to \{1, 2, \ldots\}$ that satisfy $\lim_{n\to\infty} k(n)/n = R$ and $\lim_{n\to\infty} w(n)/n = W$; more concisely, $k/n \to R$ and $w/n \to W$. The error fraction W is a real number satisfying $0 < W < 1/2$, and the code rate R is a real number with $-W \log_2 W - (1-W)\log_2(1-W) \leq 1 - R < 1$. The latter condition uses the assumption that the number of possible weight-w words in such a code of length n is bounded by 2^{n-k}.
Note that this section repeatedly invokes the standard asymptotic formula for binomial coefficients, namely

$$\frac{1}{n}\log_2 \binom{(a+o(1))n}{(b+o(1))n} \to a\log_2 a - b\log_2 b - (a-b)\log_2(a-b) = H_2(b/a),$$

to compute asymptotic exponents.
This section analyzes the asymptotic behavior of the cost of ball-collision decoding, and shows that it always has a smaller asymptotic exponent than the cost of collision decoding.

8.3.1 Asymptotic cost of ball-collision decoding

Fix real numbers P, Q, L with $0 \leq P \leq R/2$, $0 \leq Q \leq L$, and $0 \leq W - 2P - 2Q \leq 1 - R - 2L$. Fix ball-collision parameters p, q, λ_l, λ_r with $p/n \to P$, $q/n \to Q$, $\lambda_l/n \to L$, and $\lambda_r/n \to L$.
In the formulas below, expressions of the form $x \log_2 x$ are extended (continuously but not differentiably) to 0 at $x = 0$. For example, the expression $P \log_2 P$ means 0 if $P = 0$.
The asymptotic exponent of the success probability of a single iteration of ball-collision decoding is computed as:

Remark 8.8 (Asymptotic exponent of the success probability)**.**

$$\begin{aligned}B(P,Q,L) &= \lim_{n\to\infty}\frac{1}{n}\log_2\left(\binom{n}{w}^{-1}\binom{n-k-(\lambda_l+\lambda_r)}{w-2p-2q}\binom{k/2}{p}^2\binom{\lambda_l}{q}\binom{\lambda_r}{q}\right)\\ &= W\log_2 W+(1-W)\log_2(1-W)+(1-R-2L)\log_2(1-R-2L)\\ &\quad-(W-2P-2Q)\log_2(W-2P-2Q)\\ &\quad-(1-R-2L-(W-2P-2Q))\log_2(1-R-2L-(W-2P-2Q))\\ &\quad+R\log_2(R/2)-2P\log_2 P-(R-2P)\log_2(R/2-P)\\ &\quad+2L\log_2 L-2Q\log_2 Q-2(L-Q)\log_2(L-Q).\end{aligned}$$

The success probability of a single iteration is asymptotically $2^{n(B(P,Q,L)+o(1))}$.

We similarly compute the asymptotic exponent of the cost of an iteration:

Remark 8.9 (Asymptotic exponent of the cost of an iteration)**.**

$$\begin{aligned}C(P,Q,L) &= \lim_{n\to\infty}\frac{1}{n}\log_2\left(\binom{k/2}{p}\left(\binom{\lambda_l}{q}+\binom{\lambda_r}{q}\right)+\binom{k/2}{p}^2\binom{\lambda_l}{q}\binom{\lambda_r}{q}2^{-\lambda_l-\lambda_r}\right)\\ &= \max\{(R/2)\log_2(R/2)-P\log_2 P-(R/2-P)\log_2(R/2-P)\\ &\qquad+L\log_2 L-Q\log_2 Q-(L-Q)\log_2(L-Q),\\ &\qquad R\log_2(R/2)-2P\log_2 P-(R-2P)\log_2(R/2-P)\\ &\qquad+2L\log_2 L-2Q\log_2 Q-2(L-Q)\log_2(L-Q)-2L\}.\end{aligned}$$

The cost of a single iteration is asymptotically $2^{n(C(P,Q,L)+o(1))}$. Note that this is a simplified version of the iteration cost, namely $\binom{k/2}{p}(\binom{\lambda_l}{q}+\binom{\lambda_r}{q})+\binom{k/2}{p}^2\binom{\lambda_l}{q}\binom{\lambda_r}{q}2^{-\lambda_l-\lambda_r}$. The cost is actually larger than this, but only by a factor $\le \mathrm{poly}(n)$, which we are free to disregard since $\frac{1}{n}\log_2\mathrm{poly}(n)\to 0$. Note that the bounds are valid whether or not $q=0$.

Remark 8.10 (Overall attack cost)**.** The overall asymptotic ball-collision-decoding-cost exponent is the difference $D(P,Q,L)$ of the iteration-cost exponent $C(P,Q,L)$ and the success-probability exponent $B(P,Q,L)$.

$$\begin{aligned}D(P,Q,L) &= \max\{-(R/2)\log_2(R/2)+P\log_2 P+(R/2-P)\log_2(R/2-P)\\ &\qquad-L\log_2 L+Q\log_2 Q+(L-Q)\log_2(L-Q),-2L\}\\ &\quad-W\log_2 W-(1-W)\log_2(1-W)\\ &\quad-(1-R-2L)\log_2(1-R-2L)\\ &\quad+(W-2P-2Q)\log_2(W-2P-2Q)\\ &\quad+(1-R-2L-(W-2P-2Q))\\ &\qquad\cdot\log_2(1-R-2L-(W-2P-2Q)).\end{aligned}$$

For example, take $W = 0.04$ and $R = 1 + W \log_2 W + (1 - W) \log_2(1 - W) = 0.7577078109\ldots$. Choose $P = 0.004203556640625$, $Q = 0.000192998046875$, and $L = 0.017429431640625$. The success-probability exponent is $-0.0458435310\ldots$, and the iteration-cost exponent is $0.0348588632\ldots$, so the ball-collision decoding exponent is $0.0807023942\ldots$. Ball-collision decoding with these parameters therefore costs $2^{(0.0807023942\ldots+o(1))n}$ to correct $(0.04 + o(1))n$ errors in a code of rate $0.7577078109\ldots + o(1)$.

Remark 8.11 (Collision-decoding cost and the lower bound)**.** Traditional collision decoding is the special case $p_1 = p_2$, $k_1 = k_2$, $\lambda_l = \lambda_r$, $q_1 = q_2 = 0$ of ball-collision decoding. Its asymptotic cost exponent is the case $Q = 0$ of the ball-collision decoding exponent stated above.
Consider again $W = 0.04$ and $R = 1+W \log_2 W+(1-W) \log_2(1-W)$. Choosing $P = 0.00415087890625$, $Q = 0$, and $L = 0.0164931640625$ achieves decoding exponent $0.0809085120\ldots$. Partitioning the (P, L) space into small intervals and performing interval-arithmetic calculations shows that $Q = 0$ cannot do better than 0.0809; ball-collision decoding therefore has a slightly smaller exponent than collision decoding in this case.
Performing similar calculations for other pairs (W, R) shows that in each case the infimum of all collision-decoding-cost exponents is beaten by a ball-collision-decoding-cost exponent. Ball-collision decoding therefore has a smaller exponent than collision decoding.

The interval-arithmetic calculations described above are proofs of the suboptimality of $Q = 0$ for some specific pairs (W, R). These proofs have the advantage of computing explicit bounds on the collision-decoding-cost exponents for those pairs (W, R), but the proofs have two obvious disadvantages.
The first disadvantage is that these proofs do not cover *all* pairs (W, R); they leave open the possibility that ball-collision decoding has the same exponent as collision decoding for other pairs (W, R). The second disadvantage is that the proofs are much too long to verify by hand. The first disadvantage could perhaps be addressed by much more extensive interval-arithmetic calculations, partitioning the space of pairs (W, R) into boxes so small that, within each box, the ball-collision-decoding exponent is uniformly better than the minimum collision-decoding exponent; but this would exacerbate the second disadvantage.
To address both of these disadvantages the next section contains a proof that $Q = 0$ is always suboptimal: for *every* (W, R), ball-collision decoding has a smaller asymptotic cost exponent than collision decoding. Specifically, the following theorem about the overall asymptotic cost exponent is proven:

Theorem 8.12. *For each R, W it holds that*

$$\begin{aligned} &\min\{D(P, 0, L) : 0 \le P \le R/2,\ 0 \le W - 2P \le 1 - R - 2L\} \\ >\ &\min\{D(P, Q, L) : 0 \le P \le R/2,\ 0 \le Q \le L,\ 0 \le W - 2P - 2Q \le 1 - R - 2L\}. \end{aligned}$$

Note that $\{(P, 0, L)\}$ and $\{(P, Q, L)\}$ are compact sets, and D is continuous, so we are justified in writing "min" rather than "inf". The proof strategy analyzes generic perturbations of D and combines all necessary calculations into a small number of elementary inequalities in the proofs of Lemmas 8.15 and 8.16.

8.3.2 Proof of suboptimality of $Q = 0$

This section shows that, for each pair (W, R) within the range considered in the previous section, there are asymptotic parameters (P, Q, L) for ball-collision decoding whose cost exponents are smaller than the minimum collision-decoding-cost exponent, i.e., smaller than the minimum cost exponent for parameters $(P, 0, L)$.
The *parameter space* is the set of vectors (P, Q, L) of real numbers satisfying $0 \leq P \leq R/2$, $0 \leq Q \leq L$, and $0 \leq W - 2P - 2Q \leq 1 - R - 2L$. This parameter space depends implicitly on W and R.
The proof does not rely on this coding-theoretic interpretation of W, R, P, Q, L, but readers already familiar with collision decoding may find the interpretation helpful in understanding Lemma 8.14 below.
Most of the proof consists of analyzing the asymptotic cost exponent D(P,0,L) for collision decoding, namely:

Remark 8.13 (Cost exponent for collision decoding)**.**

$$\begin{aligned}
&= \max\{-(R/2)\log_2(R/2) + P\log_2 P + (R/2-P)\log_2(R/2-P), -2L\} \\
&\quad - W\log_2 W - (1-W)\log_2(1-W) \\
&\quad - (1-R-2L)\log_2(1-R-2L) + (W-2P)\log_2(W-2P) \\
&\quad + (1-R-2L-(W-2P))\log_2(1-R-2L-(W-2P)).
\end{aligned}$$

As mentioned earlier, $D(P, 0, L)$ is a continuous function of the parameters $(P, 0, L)$, and the parameter space is compact, so there exist optimal collision-decoding parameters $(P, 0, L)$, i.e., parameters that achieve the infimum of collision-decoding costs. This does not imply, and the proof of Theorem 8.12 does not use, *uniqueness* of the optimal parameters.
The proof that ball-collision decoding beats collision decoding relies on the following three facts about optimal collision-decoding parameters $(P, 0, L)$:

Lemma 8.14. *If (P, L) are optimal collision-decoding parameters than*

$$0 < L; \qquad 0 < W - 2P; \qquad \text{and} \qquad W - 2P < (1 - R - 2L)/2.$$

In other words, the collision space $\mathbb{F}_2^{\lambda_l+\lambda_r}$ is asymptotically quite large, and the uncontrolled $n - k_1 - k_2 - (\lambda_l + \lambda_r)$ positions include asymptotically many error positions, although asymptotically more non-error positions than error positions.
The three facts in Lemma 8.14 might not be news to the many authors who have written previous articles on collision decoding. However, to the best of our knowledge there were no proofs of these facts in the literature before [BLP10]. So for completeness this section includes these proofs.

The proofs do not require any background in coding theory. The main tool is nothing more than basic calculus. In order to study the growth of the collision-cost exponent induced by an increase or decrease in the values of P and L, use the Taylor-series expansion of the logarithm function: for example, a term such as $(L+\epsilon)\log_2(L+\epsilon)$ has series expansion $L\log_2 L + \epsilon\log_2(eL) + O(\epsilon^2)$ around $\epsilon = 0$. Here $e = \exp(1)$ is the base of natural logarithms. Beware that extra work is required in moving away from corners of the parameter space: for example, $L\log_2 L$ is not differentiable at $L = 0$.

The following proof is the actual proof of how to deduce the main Theorem 8.12 from Lemma 8.14 which will be proven afterwards. The proof is constructive, showing how to slightly adjust optimal parameters for collision decoding to obtain better parameters for ball-collision decoding.

Proof. (Proof of Theorem 8.12) Start with optimal collision-decoding parameters $(P, 0, L)$. Now consider the impact of increasing Q from 0 to δ and increasing L by $-(1/2)\delta\log_2\delta$, for very small δ. Of course, the increase in Q requires generalizing from collision decoding to ball-collision decoding. Lemma 8.14 says that optimal collision-decoding parameters $(P, 0, L)$ must have $0 < L$ and $0 < W - 2P < (1 - R - 2L)/2$; consequently the parameter space has room for Q and L to increase.

The quantity $L\log_2 L - Q\log_2 Q - (L-Q)\log_2(L-Q)$ increases by $-\delta\log_2\delta + O(\delta)$, and $2L\log_2 L - 2Q\log_2 Q - 2(L-Q)\log_2(L-Q) - 2L$ also increases by $-\delta\log_2\delta + O(\delta)$. The iteration-cost exponent therefore increases by $-\delta\log_2\delta + O(\delta)$. The success-probability exponent increases by $\delta\log_2\delta\log_2(e(1-R-2L)) - \delta\log_2\delta\log_2(e(1-R-2L-(W-2P))) - 2\delta\log_2\delta + O(\delta)$. The total cost exponent therefore increases by $(\delta\log_2\delta)(1 + \log_2(1-R-2L-(W-2P)) - \log_2(1-R-2L)) + O(\delta)$.

Rewrite $W - 2P < (1-R-2L)/2$ as $1 + \log_2(1-R-2L-(W-2P)) - \log_2(1-R-2L) > 0$, and deduce that the increase in the cost exponent is negative for all sufficiently small $\delta > 0$; note here that $\log_2\delta$ is negative, and that $O(\delta)/(\delta\log_2\delta) \to 0$ as $\delta \to 0$. Consequently the optimal collision-decoding parameters $(P, 0, L)$ are beaten by $(P, \delta, L - (1/2)\delta\log_2\delta)$ for all sufficiently small $\delta > 0$. □

Lemma 8.14 is proven using several lemmas. The first two lemmas require the most difficult calculations, establishing a useful inequality. The next three lemmas show that optimal collision-decoding parameters (P, L) can never have $L = 0$: Lemma 8.17 covers the case $P = 0$; Lemma 8.18 covers the case $P = R/2$; Lemma 8.19 covers the intermediate cases $0 < P < R/2$. Each of the proofs is constructive, showing how to move from $(P, 0)$ to better collision-decoding parameters.

The next two lemmas show similarly that optimal collision-decoding parameters (P, L) cannot have $0 = W - 2P = 1 - R - 2L$, and cannot have $0 = W - 2P < 1 - R - 2L$, so they must have $0 < W - 2P$. Proving Lemma 8.14 then boils down to proving $W - 2P < (1 - R - 2L)/2$; that proof concludes this section.

If $(P', 0, L')$ and $(P, 0, L)$ are in the parameter space and $D(P', 0, L') < D(P, 0, L)$ then we say that (P', L') *improves upon* (P, L). We also say that (P', L') improves upon (P, L) in the vacuous case that $(P, 0, L)$ is not in the parameter space.

Lemma 8.15. *Write* $G = (1 + W \log_2 W + (1 - W) \log_2(1 - W))/2$ *and*

$$X = \frac{1 - 2G - G \log_2 G + (W/2) \log_2(W/2) + (G - (W/2)) \log_2(G - (W/2))}{W}.$$

If $0 < W \leq 0.099$ *then* $X > 2$.

Proof. Simply graphing the function $W \mapsto X$ for $0 < W < 0.2$ suggests that X drops to a minimum close to 2.8 for $W \approx 0.13$. However, most graphing programs are easily fooled; a graph is not the same as a proof! It is easy to imagine a better graphing program that uses interval arithmetic to automatically prove a lower bound on X for a *closed* interval of W, but handling $W \to 0$ would still take more work, such as the work shown in Case 4 below.

The proof begins by rewriting X as

$$\frac{-\log_2 W}{2} + \frac{-2(1 - W) \log_2(1 - W) + 2G \log_2(1 - W/(2G)) - W \log_2(2G - W)}{2W}.$$

Note that G is a monotonic function of W for $0 < W < 0.5$, decreasing from 0.5 down to 0.

Case 1: $0.079 \leq W \leq 0.099$. Evaluating G for $W = 0.079$ and $W = 0.099$ shows that G is in the interval $[0.267090, 0.300678]$. Continuing in the same way through interval evaluations of $W/(2G)$, $\log_2(1-W/(2G))$, $2G \log_2(1-W/(2G))$, etc. eventually shows that $X \in [2.21, 3.66]$.

Case 2: $0.057 \leq W \leq 0.079$. The same sequence of calculations shows first that $G \in [0.300676, 0.342291]$ and then that $X \in [2.17, 4.03]$.

Case 3: $0.036 \leq W \leq 0.057$. Here $G \in [0.342289, 0.388180]$ and $X \in [2.21, 4.56]$.

Case 4: $0 < W \leq 0.036$. This is the heart of the proof.

Write $E = 1 - 2G = -W \log_2 W - (1 - W) \log_2(1 - W)$. Then E and $E + W$ are increasing functions of W; evaluating them for $W = 0.036$ shows that $0 < E < E + W < 1$ for $0 < W \leq 0.036$.

The Taylor-series expansion $\log(1 - t) = -t - t^2/2 - t^3/3 - t^4/4 - t^5/5 - \cdots$, where log denotes the natural logarithm, implies $\log(1 - t) \leq -t$ and $\log(1 - t) \geq -t - t^2 - t^3 - t^4 - \cdots = -t/(1 - t)$ for $0 < t < 1$. In other words, for each t with $0 < t < 1$ there exists $u \in [0, 1]$ such that $(1 - t) \log(1 - t) = -t + ut^2$. In particular, $(1 - W) \log_2(1 - W) = (-W + \alpha W^2)/\log 2$ for some $\alpha \in [0, 1]$; $(1-E) \log_2(1-E) = (-E+\beta E^2)/\log 2$ for some $\beta \in [0, 1]$; and $(1-E-W) \log_2(1-E-W) = (-(E + W) + \gamma(E + W)^2)/\log 2$ for some $\gamma \in [0, 1]$.

Now $E = -W(\log W)/\log 2 - (-W + \alpha W^2)/\log 2$ so

$$\begin{aligned}
\frac{X}{-\log_2 W} &= \frac{1}{2} - \frac{(1-W)\log(1-W)}{-W\log W} - \frac{(1-E)\log(1-E)}{-2W\log W} \\
&\quad + \frac{(1-E-W)\log(1-E-W)}{-2W\log W} \\
&= \frac{1}{2} + \frac{-W+\alpha W^2}{W\log W} + \frac{-E+\beta E^2}{2W\log W} - \frac{-E-W+\gamma(E+W)^2}{2W\log W} \\
&= \frac{1}{2} + \frac{-1+2\alpha W}{2\log W} + \frac{\beta E^2 - \gamma(E+W)^2}{2W\log W} \\
&= \frac{1}{2} + \frac{-1+2\alpha W}{2\log W} - \frac{\gamma W(\log 2 - 2\log W + 2(1-\alpha W))}{2\log 2\log W} \\
&\quad + \frac{(\beta-\gamma)W(1+\log^2 W - 2\alpha W + \alpha^2 W^2 - 2(1-\alpha W)\log W)}{2\log^2 2\log W}
\end{aligned}$$

so

$$\begin{aligned}
X &= \frac{1}{2\log 2} - \frac{\log W}{2\log 2} - \frac{\alpha W}{\log 2} + \frac{\gamma W(\log 2 - 2\log W + 2(1-\alpha W))}{2\log^2 2} \\
&\quad - \frac{(\beta-\gamma)W(1+\log^2 W - 2\alpha W + \alpha^2 W^2 - 2(1-\alpha W)\log W)}{2\log^3 2} \\
&= \frac{1}{2\log 2} - \frac{\log W}{2\log 2} - \frac{(\beta-\gamma)W\log^2 W}{2\log^3 2} \\
&\quad + \frac{(\beta-\gamma(1+\log 2))W\log W}{\log^3 2} + \frac{(\gamma(\log^2 2 + 2\log 2 + 1) - \alpha 2\log^2 2 - \beta)W}{2\log^3 2} \\
&\quad - \frac{(\beta-\gamma)\alpha W^2\log W}{\log^3 2} + \frac{(\beta-\gamma(1+\log 2))\alpha W^2}{\log^3 2} - \frac{(\beta-\gamma)\alpha^2 W^3}{2\log^3 2} \\
&\in \frac{1}{2\log 2} - \frac{\log W}{2\log 2} - \frac{[-1,1]W\log^2 W}{2\log^3 2} \\
&\quad + \frac{[-1-\log 2, 1]W\log W}{\log^3 2} + \frac{[-1-2\log^2 2, \log^2 2 + 2\log 2 + 1]W}{2\log^3 2} \\
&\quad - \frac{[-1,1]W^2\log W}{\log^3 2} + \frac{[-1-\log 2, 1]W^2}{\log^3 2} - \frac{[-1,1]W^3}{2\log^3 2}.
\end{aligned}$$

Now put a separate lower bound on each term, using the positivity of W, $-W\log W$, etc.:

$$\begin{aligned}
X &\geq \frac{1}{2\log 2} - \frac{\log W}{2\log 2} - \frac{W\log^2 W}{2\log^3 2} + \frac{W\log W}{\log^3 2} \\
&\quad + \frac{(-1-2\log^2 2)W}{2\log^3 2} - \frac{-W^2\log W}{\log^3 2} + \frac{(-1-\log 2)W^2}{\log^3 2} - \frac{W^3}{2\log^3 2}.
\end{aligned}$$

Each term here is monotonically decreasing for $0 < W \leq 0.036$: for example, $W\log^2 W$ has derivative $2\log W + \log^2 W$, which is zero only for $W = 1/e^2 > 0.1$.

Each quantity is therefore bounded below by its value at 0.036: i.e.,

$$\begin{aligned} X &\geq \frac{1}{2\log 2} - \frac{\log 0.036}{2\log 2} - \frac{0.036\log^2 0.036}{2\log^3 2} + \frac{0.036\log 0.036}{\log^3 2} \\ &\quad + \frac{(-1-2\log^2 2)0.036}{2\log^3 2} - \frac{-0.036^2\log 0.036}{\log^3 2} + \frac{(-1-\log 2)0.036^2}{\log^3 2} - \frac{0.036^3}{2\log^3 2} \\ &\geq 2.02 \end{aligned}$$

for $0 < W \leq 0.036$. □

Lemma 8.16. *Each $(P, 0, L)$ in the parameter space satisfies*

$$1 - R - ((R/2)\log_2(R/2) - P\log_2 P - (R/2-P)\log_2(R/2-P)) > 2W.$$

Proof. Case 1: $0.099 < W < 0.5$.
Recall that $1 - R \geq -W\log_2 W - (1-W)\log_2(1-W)$. Hence $1-(3/2)R-2W \geq -(1/2)-2W-(3/2)W\log_2 W-(3/2)(1-W)\log_2(1-W)$. The values of this lower bound at $0.099, 0.3, 0.5$ are $\approx 0.000726, \approx 0.2218, 0$ respectively; the derivative of the lower bound is $-2-(3/2)\log_2(eW)+(3/2)\log_2(e(1-W))$, which has a unique zero at $W = 1/(1+2^{4/3}) \approx 0.2841$; so the lower bound is positive for all W with $0.099 < W < 0.5$. In particular $2W < 1-(3/2)R$ if $0.099 < W < 0.5$.
The maximum possible value of $(R/2)\log_2(R/2)-P\log_2 P-(R/2-P)\log_2(R/2-P)$ is $(R/2)\log_2(R/2)-2(R/4)\log_2(R/4) = R/2$, so $1-R-((R/2)\log_2(R/2)-P\log_2 P-(R/2-P)\log_2(R/2-P)) \geq 1-(3/2)R > 2W$ as claimed.
Case 2: $0 < W \leq 0.099$.
Define G and X as in Lemma 8.15; then $X > 2$. Note that $G - W$ has derivative $(1/2)\log_2(eW) - (1/2)\log_2(e(1-W)) - 1 < 0$ for $0 < W \leq 0.099$, and value $0.168091\ldots > 0$ at $W = 0.099$, so $G > W$ for $0 \leq W \leq 0.099$.
Furthermore $R/2 \leq G$ by definition of G and the definition of the parameter space. So $(R/2)\log_2(R/2) - P\log_2 P - (R/2-P)\log_2(R/2-P) \leq G\log_2 G - P\log_2 P - (G-P)\log_2(G-P)$; note that for any fixed $c > 0$ and $x > c$ the function $x\log_2 x - (x-c)\log_2(x-c)$ is increasing (check its derivative).
Note also that the function $x\log_2 x+(c-x)\log_2(c-x)$ is decreasing for $x < c/2$. The parameter space forces $P \leq W/2 < G/2$ so $G\log_2 G - P\log_2 P - (G-P)\log_2(G-P) \leq G\log_2 G - (W/2)\log_2(W/2) - (G-(W/2))\log_2(G-(W/2))$.
Combining all of these inequalities produces $1-R-((R/2)\log_2(R/2)-P\log_2 P-(R/2-P)\log_2(R/2-P)) \geq 1-R-(G\log_2 G-(W/2)\log_2(W/2)-(G-(W/2))\log_2(G-(W/2))) \geq XW > 2W$ as claimed. □

Lemma 8.17. *There is a real number $\delta > 0$ such that $(\delta, -(1/2)\delta\log_2\delta)$ improves upon $(0,0)$.*

Proof. If $\delta \geq 0$ is a sufficiently small real number then the parameters $(P, L) = (\delta, -(1/2)\delta\log_2\delta)$ satisfy the constraints $0 \leq P \leq R/2$, $0 \leq L$, and $0 \leq W - 2P \leq 1 - R - 2L$ since $0 < R$ and $0 < W$. The collision-cost exponent is $\max\{\delta\log_2\delta + O(\delta), \delta\log_2\delta\} - (1-W)\log_2(1-W) - (1-R)\log_2(1-R) + (1-R-$

$W)\log_2(1-R-W)-(\delta\log_2\delta)\log_2(e(1-R))+(\delta\log_2\delta)\log_2(e(1-R-W))+O(\delta)=-(1-W)\log_2(1-W)-(1-R)\log_2(1-R)+(1-R-W)\log_2(1-R-W)+(\delta\log_2\delta)(1+\log_2(1-R-W)-\log_2(1-R))+O(\delta)$. The inequality $2W<1-R$ implies $1+\log_2(1-R-W)-\log_2(1-R)>0$, so $(\delta\log_2\delta)(1+\log_2(1-R-W)-\log_2(1-R))+O(\delta)$ is negative for all sufficiently small $\delta>0$, improving upon $(0,0)$. □

Lemma 8.18. *There is a real number $\delta>0$ such that $(R/2-\delta,-(1/2)\delta\log_2\delta)$ improves upon $(R/2,0)$.*

Proof. If $W<R$ then $(R/2,0,0)$ is outside the parameter space so the conclusion is vacuously satisfied. Assume from now on that $W\geq R$.
For all sufficiently small $\delta\geq 0$ the parameters $(P,L)=(R/2-\delta,-(1/2)\delta\log_2\delta)$ satisfy the constraints $0\leq P\leq R/2$, $0\leq L$, and $0\leq W-2P\leq 1-R-2L$. The iteration-cost exponent is $\max\{\delta\log_2\delta+O(\delta),\delta\log_2\delta\}-W\log_2 W-(1-R)\log_2(1-R)+(W-R+2\delta)\log_2(W-R+2\delta)-(\delta\log_2\delta)\log_2(e(1-R))+(\delta\log_2\delta)\log_2(e(1-W))+O(\delta)=-W\log_2 W-(1-R)\log_2(1-R)+(W-R+2\delta)\log_2(W-R+2\delta)+(\delta\log_2\delta)(1+\log_2(1-W)-\log_2(1-R))+O(\delta)$.
If $W=R$ then $(W-R+2\delta)\log_2(W-R+2\delta)+(\delta\log_2\delta)(1+\log_2(1-W)-\log_2(1-R))+O(\delta)=3\delta\log_2\delta+O(\delta)$. This is negative for all sufficiently small $\delta>0$.
Otherwise $W>R$ so the difference between $(W-R+2\delta)\log_2(W-R+2\delta)+(\delta\log_2\delta)(1+\log_2(1-W)-\log_2(1-R))+O(\delta)$ and $(W-R)\log_2(W-R)$ is $(\delta\log_2\delta)(1+\log_2(1-W)-\log_2(1-R))+O(\delta)$. This difference is also negative for all sufficiently small $\delta>0$: recall that $2(1-W)>1>1-R$, so the coefficient $1+\log_2(1-W)-\log_2(1-R)$ is positive.
Either way $(P,L)=(R/2-\delta,-(1/2)\delta\log_2\delta)$ improves upon $(R/2,0)$. □

Lemma 8.19. *If $0<P<R/2$ then there is a real number $\delta>0$ such that (P,δ) improves upon $(P,0)$.*

Proof. Consider the impact of changing the parameter L from 0 to δ. The quantity $-(R/2)\log_2(R/2)+P\log_2 P+(R/2-P)\log_2(R/2-P)$ is negative and unchanged, and $-2L$ changes from 0 to -2δ, so $\max\{-(R/2)\log_2(R/2)+P\log_2 P+(R/2-P)\log_2(R/2-P),-2L\}$ increases by -2δ if δ is sufficiently small. The quantity $-(1-R-2L)\log_2(1-R-2L)$ increases by $2\delta\log_2(e(1-R))+O(\delta^2)$. The quantity $(1-R-2L-(W-2P))\log_2(1-R-2L-(W-2P))$ increases by $-2\delta\log_2(e(1-R-(W-2P)))+O(\delta^2)$.
The total collision-cost exponent increases by $2\delta(-1+\log_2(1-R)-\log_2(1-R-(W-2P)))+O(\delta^2)$. The coefficient $-1+\log_2(1-R)-\log_2(1-R-(W-2P))$ is negative since $W-2P<(1-R)/2$. Hence (P,δ) improves upon $(P,0)$ for all sufficiently small $\delta>0$. □

Lemma 8.20. *There is a real number $c\geq 2$ satisfying the following condition: if $W<R$ then $c\log_2 c-(c-1)\log_2(c-1)>(1/2)(\log_2(R-W)-\log_2 W)$. For any such c there is a real number $\delta>0$ such that $((W-\delta)/2,(1-R-c\delta)/2)$ improves upon $(W/2,(1-R)/2)$.*

Proof. If $W > R$ then $(W/2, (1-R)/2)$ is outside the parameter space and the conclusions are vacuously satisfied for, e.g., $c = 2$ and $\delta = 1$. Assume from now on that $W \leq R$.
Choose a real number c large enough to meet both of the following constraints: first, $c \geq 2$; second, if $W < R$ then $c \log_2 c - (c-1)\log_2(c-1) > (1/2)(\log_2(R-W) - \log_2 W)$. This can always be done: $c \log_2 c - (c-1)\log_2(c-1) \to \infty$ as $c \to \infty$.
Consider the impact of changing L from $(1-R)/2$ to $(1-R-c\delta)/2$, and at the same time changing P from $W/2$ to $(W-\delta)/2$. This change fits the parameter constraints for sufficiently small $\delta > 0$.
The quantity $-(1-R-2L)\log_2(1-R-2L)$ changes from 0 to $-c\delta \log_2(c\delta)$. The quantity $(W-2P)\log_2(W-2P)$ changes from 0 to $\delta \log_2 \delta$. The quantity $(1-R-2L-(W-2P))\log_2(1-R-2L-(W-2P))$ changes from 0 to $(c-1)\delta \log_2((c-1)\delta)$.
The quantity $\max\{-(R/2)\log_2(R/2) + P\log_2 P + (R/2-P)\log_2(R/2-P), -2L\}$ is dominated by its first term since $2L = 1-R > 2W + ((R/2)\log_2(R/2) - P\log_2 P - (R/2-P)\log_2(R/2-P))$ by Lemma 8.16. It thus increases by $((W-\delta)/2)\log_2((W-\delta)/2) - (W/2)\log_2(W/2) + ((R-W+\delta)/2)\log_2((R-W+\delta)/2) - ((R-W)/2)\log_2((R-W)/2)$ if δ is sufficiently small.
The total cost exponent increases by the quantity $\delta((c-1)\log_2(c-1) - c\log_2 c) + ((W-\delta)/2)\log_2((W-\delta)/2) - (W/2)\log_2(W/2) + ((R-W+\delta)/2)\log_2((R-W+\delta)/2) - ((R-W)/2)\log_2((R-W)/2)$.
If $W = R$ then this increase is $(\delta/2)\log_2(\delta/2) + O(\delta)$ and is therefore negative for all sufficiently small $\delta > 0$.
If $W < R$ then this increase is $\delta((c-1)\log_2(c-1) - c\log_2 c + (1/2)(\log_2(e(R-W)/2) - \log_2(eW/2))) + O(\delta^2)$, The coefficient of δ is negative by choice of c, so the increase is negative for all sufficiently small $\delta > 0$.
In all cases $((W-\delta)/2, (1-R-c\delta)/2)$ improves upon $(W/2, (1-R)/2)$. □

Lemma 8.21. *Assume that* $0 < 1-R-2L$. *Then there is a real number* $\delta > 0$ *such that* $((W-\delta)/2, L)$ *improves upon* $(W/2, L)$.

Proof. Consider collision-decoding parameters (P, L) with $0 = W-2P < 1-R-2L$. If $W > R$ then $(W/2, 0, L)$ is outside the parameter space so the conclusion is vacuously satisfied. Assume from now on that $W \leq R$.
Consider the impact of changing P from $W/2$ to $(W-\delta)/2$. This change fits the parameter constraints for sufficiently small $\delta > 0$.
The quantity $(W-2P)\log_2(W-2P)$ increases by $\delta \log_2 \delta$. The quantity $(1-R-2L-(W-2P))\log_2(1-R-2L-(W-2P))$ increases by $O(\delta)$. The quantity

$$\max\{-(R/2)\log_2(R/2) + P\log_2 P + (R/2-P)\log_2(R/2-P), -2L\}$$

increases by a value between 0 and $((W-\delta)/2)\log_2((W-\delta)/2) - (W/2)\log_2(W/2) + ((R-W+\delta)/2)\log_2((R-W+\delta)/2 - ((R-W)/2)\log_2((R-W)/2)$, which is $(\delta/2)\log_2(\delta/2) + O(\delta)$ if $W = R$ and $O(\delta)$ if $W < R$. The total increase in the cost is between $\delta \log_2 \delta + O(\delta)$ and $(3/2)\delta \log_2 \delta + O(\delta)$, and is therefore negative for all sufficiently small $\delta > 0$. □

Proof. (Proof of Lemma 8.14) The hypothesis is that (P, L) minimizes $D(P, 0, L)$, i.e., that nothing improves upon (P, L).
The definition of the parameter space implies $L \geq 0$. Suppose that $L = 0$. Then $P < 0$ would contradict the definition of the parameter space; $P = 0$ would contradict Lemma 8.17; $0 < P < R/2$ would contradict Lemma 8.19; $P = R/2$ would contradict Lemma 8.18; and $P > R/2$ would contradict the definition of the parameter space. Hence $L > 0$.
The definition of the parameter space also implies $0 \leq W - 2P$. Suppose that $0 = W - 2P$. Then $0 = 1 - R - 2L$ would force $(P, L) = (W/2, (1 - R)/2)$, contradicting Lemma 8.20; $0 < 1 - R - 2L$ would contradict Lemma 8.21; and $0 > 1 - R - 2L$ would contradict the definition of the parameter space. Hence $0 < W - 2P$.
Suppose that $2L > (R/2)\log_2(R/2) - P\log_2 P - (R/2 - P)\log_2(R/2 - P)$. Consider the impact of decreasing L by δ. The quantity

$$\max\{-(R/2)\log_2(R/2) + P\log_2 P + (R/2 - P)\log_2(R/2 - P), -2L\}$$

is dominated by the first term, so it is unchanged for sufficiently small δ. The total cost decreases by $(2\log_2(1 - R - 2L) - 2\log_2(1 - R - 2L - (W - 2P)))\delta + O(\delta^2)$, contradicting the optimality of (P, L); note that the coefficient $2\log_2(1 - R - 2L) - 2\log_2(1 - R - 2L - (W - 2P))$ is positive since $W - 2P > 0$.
Therefore $2L \leq (R/2)\log_2(R/2) - P\log_2 P - (R/2 - P)\log_2(R/2 - P)$, and $1 - R - 2L > 2W \geq 2(W - 2P)$ as claimed. □

Chapter 9

Attacking the FSB compression function

Code-based cryptography is mainly concerned with public-key cryptography. This chapter however deals with symmetric code-based cryptography, namely with an attack on the code-based cryptographic hash function FSB.
The results presented here are based on the article "FSBday: Implementing Wagner's generalized birthday attack against the SHA-3 round-1 candidate FSB" [BLN+09] which is joint work with Bernstein, Lange, Niederhagen, and Schwabe. The so-called "FSBday" attack has been implemented by Niederhagen and Schwabe, and was carried out successfully.

- This chapter essentially follows [BLN+09] which focused on applying the generalized birthday attack in a storage-restricted environment. As the emphasis of this thesis lies on code-based cryptography, this chapter shuffles results of the first three sections of [BLN+09] and in particular puts the description of the FSB hash function in the perspective of coding theory.

- This thesis omits the details of Niederhagen's and Schwabe's implementation which can be found in [BLN+09, Section 5] and also in more detail in Schwabe's Ph.D. thesis [Sch11].

9.1 The FSB hash function

The "Fast Syndrome Based hash function" (FSB) was introduced by Augot, Finiasz and Sendrier in 2003 in [AFS03]. The security of the compression function of FSB is related to the syndrome-decoding problem discussed in Remark 4.11. On the other hand generic attacks can be applied: Coron and Joux pointed out in [CJ04] that Wagner's generalized birthday attack [Wag02a] can be used to find preimages and collisions in the compression function of FSB.
Augot, Finiasz, and Sendrier adjusted their parameters in a follow-up article [AFS05] to withstand Wagner's attack. The design of the compression function, in particular the choice of codes, was tuned in [FGS07] and [Fin08]. In 2008 Augot, Finiasz, Gaborit, Manuel, and Sendrier submitted an improved version of FSB to NIST's SHA-3 competition [NIS07]; see also [NPS10]. FSB was one of the 64 hash functions

submitted to the competition, and one of the 51 hash functions selected for the first round. However, FSB was significantly slower than most submissions, and was not one of the 14 hash functions selected for the second round. The FSB proposal also contains a reduced-size version FSB_{48} which was suggested as a training case by the designers of FSB.
This section briefly describes the construction of the FSB hash function as defined in the SHA-3 proposal. Details which are necessary for the implementation of FSB but which do not influence the attack are omitted.
The *FSB hash function* processes a message in three steps: first the message is converted by a so-called domain extender into suitable inputs for the compression function. The compression function digests the inputs in the second step. In the third and final step the Whirlpool hash function designed by Barreto and Rijmen [BR03] is applied to the output of the compression function in order to produce the desired length of output. This chapter only deals with attacking the compression function. In order to place the compression function into coding theory we introduce "regular" words.

Definition 9.1. Let n, w be integers with w dividing n. An element in $\mathbb{F}_2^n$ of Hamming weight w is called a *regular word* or simply a *regular n-bit string* if there is exactly a single 1 in each interval $[(i-1)\frac{n}{w}, i\frac{n}{w} - 1]_{1 \leq i \leq w}$. Any such interval is called a *block*.

The *FSB compression function* is set up as follows:

- The main parameters of the compression function are called n, r, and w. The parameter w is chosen such that it divides n.
- Consider n strings of length r which are chosen uniformly at random and which can be written as an $r \times n$ binary matrix H.
- The *input* of the compression function is a regular n-bit string x.
- The matrix H is split into w *blocks* of n/w columns. Each nonzero entry of the input x indicates exactly one column in each block. The *output* of the compression function is an r-bit string which is produced by computing the sum of all the w columns of the matrix H indicated by the input string.

The matrix H can be seen as the parity-check matrix of a binary linear code C of length n and dimension $n - r$; thus the parameter r denotes the *codimension* of C. From the coding-theory perspective the compression function computes the syndrome of $x \in \mathbb{F}_2^n$ with respect to H; the peculiarity of the FSB compression function lies in its restriction to regular inputs x. The FSB proposal [AFG+09] actually specifies a particular structure of H for efficiency; the details are outlined in Remark 9.6 but do not affect Wagner's attack.

Definition 9.2. A regular word x which has syndrome s with respect to the FSB compression matrix H is called a *preimage* for s.

Definition 9.3. A *collision* in the FSB compression function means a pair of distinct regular n-bit strings x, x' such that the columns indicated by x and x' sum up to the same syndrome.

Note that by linearity the sum (mod 2) of the columns of H indicated by x and x' yields the zero-syndrome. Thus, when looking for collisions we could look for 2-*regular codewords* in C, i.e., codewords of weight $2w$ having exactly 2 ones in each block of length n/w. Note that this perspective leads to a small inefficiency. In fact, since columns could cancel each other one should also allow any codeword of weight $2w'$ for $0 < w' \leq w$ having weight exactly 0 or 2 in each block of length n/w. For the sake of simplicity those cases are ignored; our attack will look for 2-regular words of weight exactly $2w$.
In [AFS05] the problem of finding preimages was called the "regular syndrome decoding problem" and the problem of finding collisions was called "2-regular null-syndrome decoding" problem. As certain instances of the syndrome decoding problem (Remark 4.11) both problems were proven to be NP-complete in [AFS05].
Following the notation in [AFG+09] $\text{FSB}_{\texttt{length}}$ denotes the version of FSB which produces a hash value of length `length`. Note that the output of the compression function has r bits where r is considerably larger than `length`.

Remark 9.4 (FSB versions)**.** NIST demands hash lengths of 224, 256, 384, and 512 bits, respectively. The SHA-3 proposal contains five versions of FSB: FSB_{160}, FSB_{224}, FSB_{256}, FSB_{384}, and FSB_{512}. The parameters for those versions are listed in Table 9.1.

The proposal also contains FSB_{48} which is a reduced-size version of FSB and the main attack target in this chapter. The specifications are as follows.

Remark 9.5 (FSB_{48} parameters)**.** The binary matrix H for FSB_{48} has dimensions $192 \times 3 \cdot 2^{17}$; i.e., r equals 192 and n is $3 \cdot 2^{17}$. In each round a message chunk is converted into a regular $3 \cdot 2^{17}$-bit string of Hamming weight $w = 24$. The matrix H contains 24 blocks of length 2^{14}. Each 1 in the regular bit string indicates exactly one column in a block of the matrix H. The output of the compression function is the sum of those 24 columns.

The attack against FSB_{48} considers a pseudo-random matrix H which is constructed as described in [AFG+09, Section 1.2.2]:

Remark 9.6 (A pseudo-random matrix for FSB_{48})**.** The matrix H consists of 2048 submatrices, each of dimension 192×192. In order to build the first submatrix consider a slightly larger matrix of dimension 197×192. Its first column consists of the first 197 digits of π where each digit is taken modulo 2. The remaining 191 columns of this submatrix are cyclic shifts of the first column. The matrix is then truncated to its first 192 rows which form the first submatrix of H. For the second submatrix consider digits 198 up to 394 of π. Again build a 197×192 bit matrix where the first column corresponds to the selected digits (each taken modulo 2)

and the remaining columns are cyclic shifts of the first column. In order to get the second block matrix of H truncate this submatrix to the first 192 rows. Construct the remaining blocks in the same way.

Note that this is one possible choice for the matrix H. The attack described in [BLN+09] does not make use of the structure of this particular matrix. The FSB-day attack nevertheless uses this matrix since it is contained in the FSB reference implementation submitted to NIST by the FSB designers.

Remark 9.7 (Note on information-set decoding)**.** In [AFG+09] the FSB designers say that Wagner's attack is the fastest known attack for finding preimages, and for finding collisions for small FSB parameters, but that another attack — information-set decoding — is better than Wagner's attack for finding collisions for large FSB parameters. In general, information-set decoding can be used to find an n-bit string of weight 48 indicating 48 columns of H which add up to zero.
However, classical information-set decoding will not take into account that the target is a *regular* n-bit string. The only known way to obtain a regular n-bit string is running the algorithm repeatedly until the output happens to be regular. Thus, the running times given in [AFG+09] certainly provide lower bounds for information-set decoding, but in practice they are not likely to hold.

9.2 Wagner's generalized birthday attack

The *birthday paradox* describes the curiosity that it takes a group of only 23 random people in order to have a chance of more than 50% that two people in that group have their birthday on the same day. Most people find it surprising that the number $n = 23$ is quite small compared to $N = 365$ days. However, simply calculating probabilities shows that the probability P_n of "no two people having the same birthday" for a group of n people and N possible birthdays is less than $\frac{1}{2}$ if $n \approx \sqrt{2 \log 2}\sqrt{N}$. This square-root effect also shows up in a similar problem, the so-called *birthday problem*, which relates to finding collisions in many cryptographic algorithms. In particular, let L and L' be two lists containing uniform random bit strings of length B which are uniformly distributed among all 2^B possible values. Then one can expect to find at least one pair $(x, x') \in L \times L'$ with $x = x'$ if both lists have size about $2^{B/2}$. The search for (x, x') can be performed by first sorting L and then checking for each $x' \in L'$ if it is contained in L. Note that from now on we rather speak of bit strings, which we see as elements from the set $\{0,1\}^B$, rather than of vectors in $\mathbb{F}_2^B$.

Definition 9.8. The *generalized birthday problem* considers k lists containing uniform random B-bit strings. The task is to find k nonzero elements — exactly one in each list — whose sum modulo 2 equals 0.

Another name for the generalized birthday problem with k lists is the *k-sum problem*. Wagner [Wag02a] gave an efficient algorithm for this problem. His algorithm is most efficient if k is a power of 2 and we will restrict to this case from now on.

Note that the idea of generalized birthday attacks is much older than Wagner's algorithm. Camion and Patarin [CP91] introduced a similar attack at Eurocrypt 1991. However, the next section follows Wagner's description.

9.2.1 Wagner's tree algorithm

Fix an integer $i > 2$. Wagner's algorithm builds a binary tree starting from input lists $L_{0,0}, L_{0,1}, \ldots, L_{0,2^{i-1}-1}$ where we denote list k on level j by $L_{j,k}$ (see Figure 9.3). In order to identify 2^{i-1} elements in $\{0,1\}^B$ summing up to zero, these elements are seen as a concatenation of i times B/i bits read from left to right. For the moment we simply assume that B is a multiple of i and refer the reader to Remark 9.9 for the general case. Each level of the algorithm is concerned with B/i bits. The speed and success probability of the algorithm are analyzed under the following assumption: for $0 \leq j < i-2$ each list $L_{j,k}$ contains about $2^{B/i}$ elements which — if restricted to the B/i bits considered at level j — are uniformly distributed over all possible B/i elements in $\{0,1\}^{B/i}$; similarly, each of the two lists at level $i-2$ contain $2^{B/i}$ elements whose right-most $2B/i$ bits are uniformly distributed over all possible $2B/i$ elements in $\{0,1\}^{2B/i}$.

- On level 0 take the first two lists $L_{0,0}$ and $L_{0,1}$ and compare their list elements on their left-most B/i bits. Given that each list contains about $2^{B/i}$ uniform random elements one can expect about $2^{B/i}$ pairs of elements which are equal on their left-most B/i bits. For each of these pairs compute the sum of both elements on all their B bits and put this sum into a new list $L_{1,0}$. Similarly compare the other lists — always two at a time — and look for elements matching on their left-most B/i bits which are added and put into new lists. This process of *merging* yields 2^{i-2} lists each containing about $2^{B/i}$ elements which are zero on their left-most B/i bits; those B/i bits are *clamped* in the sense that they can be neglected on the next level. This completes level 0.

- On level 1 take the first two lists $L_{1,0}$ and $L_{1,1}$ which are the results of merging the lists $L_{0,0}$ and $L_{0,1}$ as well as $L_{0,2}$ and $L_{0,3}$ from level 0. Compare the elements of $L_{1,0}$ and $L_{1,1}$ on their left-most $2B/i$ bits. As a result of the merging in the previous level, the last B/i bits are already known to be 0, so it suffices to compare the next B/i bits. Each list on level 1 contains about $2^{B/i}$ elements which are again assumed to be uniformly distributed over all possible elements; thus one can expect about $2^{B/i}$ elements matching on B/i bits. For each pair of matching elements (x, x') compute the sum $x + x'$ and put it into a new list $L_{2,0}$. Similarly the remaining lists on level 1 are handled.

- Continue in the same way until level $i-2$. On each level j consider the elements on their left-most $(j+1)B/i$ bits of which jB/i bits are known to be zero as a result of the previous merge. The merge operation on level $i-2$ yields two lists containing about $2^{B/i}$ elements. The left-most $(i-2)B/i$ bits of each element in both lists are zero. Comparing the elements of both lists on their

Figure 9.1: Clamping $B/5$ bits on level 0, 1, and 2 and $2B/5$ bits on level 3 given 16 lists of B-bit strings.

$2B/i$ remaining bits gives 1 expected match, i.e., an element being zero on all B entries. Since each element is the sum of elements from the previous steps this final sum is the sum of 2^{i-1} elements from the original lists and thus a solution to the generalized birthday problem.

Note that we will refine the clamping operation in the next section.

Remark 9.9. We can also handle non-integer values of B/i, i.e., cases where B is not a multiple of i. We can round up the real number B/i to the nearest integer and then clamp $\lceil B/i \rceil$ bits on levels 0 to $i-3$ and $B-(i-2)\lceil B/i \rceil$ bits on level $i-2$. If the input lists each contain about $2^{\lceil B/i \rceil}$ elements we can expect a collision under the same randomness assumptions as before.

9.2.2 Wagner in storage-restricted environments

In [AFS05] Augot, Finiasz, and Sendrier also analyzed Wagner's attack in the case where the size of the input lists does not equal $2^{B/i}$. They noticed that the FSB compression matrix H yields more elements than needed for Wagner's algorithm; in particular, the number of possible elements 2^ℓ per list coming from columns of H is usually not anywhere near $2^{B/i}$, no matter which B is given and which i is chosen. For ℓ smaller than B/i [AFS05] proposes a precomputation step which balances the number of possible elements with the number of elements needed for Wagner's algorithm to succeed. As a result of the precomputation the number of bits clamped per level is smaller than B/i. Note that the parameters i and ℓ dictate the amount of storage used. Bernstein in [Ber07] considered a more general scenario where only a limited amount of storage is available. He introduces an extra parameter in order to control storage; in a precomputation step the amount of list entries per list is reduced as well as the amount of clamped bits per level. Another special case of the same techniques appeared in a 2009 article [MS09] by Minder and Sinclair. We describe and generalize the "precomputation" technique following [Ber07].

Remark 9.10 (Clamping through precomputation)**.** Suppose that there is space for lists of size only 2^b with $b < B/i$. Bernstein suggests to generate $2^{b+(B-ib)}$ entries per list and only consider those of which the left-most $B-ib$ bits are zero.
This idea can be generalized as follows: the left-most $B-ib$ bits can have an arbitrary value which is called *clamping value* or *clamping constant*. This clamping value does

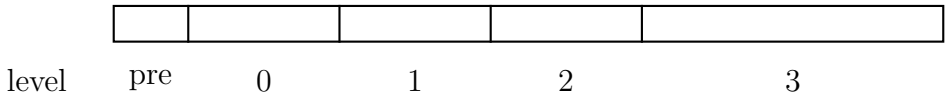

Figure 9.2: Clamping $B-5b$ in a *pre*computation step and b bits on level 0, 1, and 2, and $2b$ bits on level 3 given 16 lists of B-bit strings where there is space for storing only lists with 2^b elements.

not even have to be the same on all lists as long as the *sum* of all clamping values is zero. This will be important if an attack does not produce a collision; then the attack is restarted with a different clamping value.

Note that the idea of clamping constants can be applied to any step of the algorithm. We refine the *clamping* operation by allowing that at any level of Wagner's tree algorithm the sum of two elements on b bits can be any value as long as the sum of all the clamping values on that level is zero.
Clamping through precomputation may be limited by the maximal number of entries which can be generated per list. Furthermore, halving the available storage space increases the precomputation time by a factor of 2^i.
Note that clamping some bits through precomputation might be a good idea even if enough storage is available as one can reduce the amount of data in later steps and thus make those steps more efficient.
After the precomputation step Wagner's tree algorithm starts with lists containing bit strings of length B' where B' equals B minus the number of clamped bits. The performance evaluation of the Wagner attack in this chapter only considers lists on level 0 *after* clamping through precomputation and then uses B instead of B' for the number of bits in these entries.

Remark 9.11 (Repeating the attack)**.** Another way to mount Wagner's attack in storage-restricted environments is to carry out the whole computation with smaller lists leaving some bits at the end "uncontrolled." One can deal with the lower success probability by repeatedly running the attack with different clamping values.
In the context of clamping through precomputation one can simply vary the clamping values used during precomputation. If for some reason it is not possible to clamp bits through precomputation the same idea of changing clamping values can be applied in an arbitrary merge step of the tree algorithm. Note that any solution to the generalized birthday problem can be found by some choice of clamping values.

Wagner's algorithm, without clamping through precomputation, produces an expected number of exactly one collision. However this does not mean that running the algorithm necessarily produces a collision.

Remark 9.12 (Expected number of runs)**.** In general, the expected number of runs of Wagner's attack is a function of the number of remaining bits in the entries of the two input lists of the last merge step and the number of elements in these lists.

Assume that b bits are clamped on each level and that lists have length 2^b. Then the probability to have at least one collision after running the attack once is

$$P_{\text{success}} = 1 - \left(\frac{2^{B-(i-2)b} - 1}{2^{B-(i-2)b}}\right)^{2^{2b}}$$

and the expected number of runs $E(R)$ is the reciprocal of P_{success}.
For larger values of $B - ib$ the expected number of runs is about 2^{B-ib}. The time for the attack t_W is modelled as being linear in the amount of data on level 0, i.e.,

$$t_W \in \Theta\left(2^{i-1}2^{B-ib}2^b\right). \tag{9.1}$$

Here 2^{i-1} is the number of lists, 2^{B-ib} is approximately the number of runs, and 2^b is the number of entries per list. Observe that this formula will usually underestimate the real time of the attack by assuming that all computations on subsequent levels are together still linear in the time required for computations on level 0.

If because of storage restrictions the number of uncontrolled bits is high, it may be more efficient to use a variant of Wagner's attack that uses Pollard iteration [Pol78], [Knu97, Chapter 3, exercises 6 and 7].

Remark 9.13 (Using Pollard iteration)**.** Assume that $L_0 = L_1$, $L_2 = L_3$, etc., and that combinations $x_0 + x_1$ with $x_0 = x_1$ are excluded. Then the output of the generalized birthday attack will be a collision between two distinct elements of $L_0 + L_2 + \cdots + L_{2^{i-1}}$. Another approach is to start with only 2^{i-2} lists $L_0, L_2, \ldots, L_{2^{i-1}}$ and apply the usual Wagner tree algorithm, with a nonzero clamping constant to enforce the condition that $x_0 \neq x_1$. The number of clamped bits before the last merge step is now $(i-3)b$. The last merge step produces 2^{2b} possible values, the smallest of which has an expected number of $2b$ leading zeros, leaving $B - (i-1)b$ uncontrolled. Think of this computation as a function mapping clamping constants to the final $B - (i-1)b$ uncontrolled bits and apply Pollard iteration to find a collision between the output of two such computations; combination then yields a collision of 2^{i-1} B-bit strings. As Pollard iteration has square-root running time, the expected number of runs for this variant is $2^{B/2-(i-1)b/2}$, each taking time $2^{i-2}2^b$ (compare to (9.1)), so the expected running time is

$$t_{PW} \in \Theta\left(2^{i-2}2^{B/2-(i-1)b/2+b}\right). \tag{9.2}$$

The Pollard variant of the attack becomes more efficient than plain Wagner with repeated runs if $B > (i+2)b$.

9.3 Attacking the compression function of FSB_{48}

This section describes how Wagner's generalized birthday attack can be applied to FSB_{48}. It turns out that a straightforward implementation would need 20 TB of

storage as will be shown in the following. However, the FSBday attack was designed to run on 8 nodes of the Coding and Cryptography Computer Cluster (CCCC) at Technische Universiteit Eindhoven [Lan] which have a total hard-disk space of only 5.5 TB. This section describes the methods in [BLN+09] for carrying out the attack in this storage-restricted environment.

9.3.1 Applying Wagner's attack to FSB_{48}

Coron and Joux described in [CJ04] how to to find preimages and collisions in the compression function of FSB using Wagner's generalized birthday attack. This section presents a slightly streamlined version of the attack of [CJ04] in the case of FSB_{48}.

Remark 9.14. A *collision* for FSB_{48} is given by 48 columns of the matrix H which add up to zero; the collision has exactly two columns per block. Each block contains 2^{14} columns and each column is a 192-bit string.

The FSBday attack uses 16 lists to solve this particular 48-sum problem. Note that 16 is the largest power of 2 dividing 48.
Each list entry will be the sum of three columns coming from one and a half blocks. This ensures that no overlaps occur, i.e., more than two columns coming from one matrix block in the end. In order to estimate the complexity of the generalized birthday attack for FSB_{48} assume that taking sums of the columns of H does not bias the distribution of 192-bit strings. Applying Wagner's attack in a straightforward way means that each list contains about $2^{\lceil 192/5 \rceil}$ entries. Using the usual randomness assumption clamping away 39 bits in each step should yield one collision after one run of the tree algorithm.
The algorithm builds 16 lists containing 192-bit strings each being the sum of three distinct columns of the matrix H. Each triple of three columns is selected from one and a half blocks of H in the following way:

Remark 9.15 (Constructing lists)**.** List $L_{0,0}$ contains the sums of columns i_0, j_0, k_0, where columns i_0 and j_0 come from the first block of 2^{14} columns, and column k_0 is picked from the following block with the restriction that it is taken from the first half of it. There are about 2^{27} possible sums of two columns i_0 and j_0 coming from the first block. These two columns are then added to all possible columns k_0 coming from the first 2^{13} elements of the second block of the matrix H. The resulting list $L_{0,0}$ contains about 2^{40} elements.
Note that by splitting every second block in half several solutions of the 48-sum problem are neglected. For example, a solution involving two columns from the first half of the second block cannot be found by this algorithm. This design choice is reasonable since fewer lists would nevertheless require more storage and a longer precomputation phase to build the lists.
The second list $L_{0,1}$ contains sums of columns i_1, j_1, k_1, where column i_1 is picked from the second half of the second block of H and j_1 and k_1 come from the third block of 2^{14} columns. This again yields about 2^{40} elements.

The lists $L_{0,2}$, $L_{0,3}$,..., $L_{0,15}$ are constructed in a similar way.

For each list more than twice the amount needed for a straightforward attack is generated in a precomputation step. In order to reduce the amount of data for the following steps note that about $2^{40}/4$ elements are likely to be zero on their left-most two bits. Clamping those two bits away should thus yield a list of 2^{38} bit strings. As the left-most two bits of the list elements are known they can be ignored and the list elements can be seen as 190-bit strings. A straightforward application of Wagner's attack to 16 lists with about $2^{190/5}$ elements is expected to yield a collision after completing the tree algorithm.

Remark 9.16 (Note on complexity in the FSB proposal)**.** Given the parameters n, w, and r for the matrix H the FSB proposal estimates the complexity of a generalized birthday attack by first choosing the parameter i dictating the number of lists in the tree algorithm; this choice is done following a simpler version of the formula given in [AFS05], namely

$$\frac{r}{i} \leq \frac{w}{2^{i-1}} \log_2 \left(\binom{\frac{n}{w}}{2} + 1 \right).$$

This formula bounds the number of prospective list entries $2^{r/i}$ from above by all possible 2-regular words coming from $w/2^{i-1}$ blocks of length n/w. We comment that this is a good rule of thumb but that storage limitations might force attackers to be more flexible in their choice of list sizes, choice of how to group blocks (even to split blocks as we suggested above), the number of bits to be clamped on each level, etc. The SHA-3 proposal is even cruder in its suggestions and does not consider the number of regular words but in general the number of weight-$2w$ words as an upper bound when choosing parameters for generalized birthday attacks. After a precomputation consideration the SHA-3 proposal [AFG+09] estimates the complexity of a generalized birthday attack as $O(2^{r/i}r)$ where i is chosen according to a crude version of the formula above. We comment that this does not take storage into account, and in general is an underestimate of the work required by Wagner's algorithm. We will show in the following that attacks of this type against FSB are more difficult than claimed by the FSB designers.

9.3.2 Attack strategy

This section discusses the necessary measures for mounting the attack on the Coding and Cryptography Computer Cluster (CCCC) at Technische Universiteit Eindhoven. We first consider the storage required for one list entry.
The number of bytes required to store one list entry depends on the representation of the entry. This section considers four different ways of representing an entry which will be explained in the following.

Remark 9.17 (Value-only representation)**.** The obvious way of representing a list entry is as a 192-bit string, the sum of columns of the matrix. Bits which are known to be zero do not have to be stored, so on each level of the tree the number of bits

per entry decreases by the number of bits clamped on the previous level. Note that the *value* of the entry is not of any interest since a successful attack will produce the all-zero bit string; the goal is to find the column positions in the matrix that lead to this all-zero value. However, Section 9.3.4 will show that computations only involving the *value* can be useful if the attack has to be run multiple times due to storage restrictions.

Remark 9.18 (Value-and-positions representation)**.** If enough storage is available positions in the matrix can be stored alongside the value.

Remark 9.19 (Compressed positions)**.** Instead of storing full positions one can save storage by only storing, e.g., positions modulo 256. After the attack has successfully finished the full position information can be computed by checking which of the possible positions lead to the appropriate intermediate results on each level.

Remark 9.20 (Dynamic recomputation)**.** In the case that full positions are kept the value of the sum of the corresponding values does not have to be stored. Every time the value (or parts of it) is needed it can be dynamically recomputed from the positions. In each level the size of a single entry doubles (because the number of positions doubles), the expected number of entries per list remains the same but the number of lists halves, so the total amount of data is the same on each level when using dynamic recomputation. As discussed in the previous section there are 2^{40} possibilities to choose columns to produce entries of a list, so positions on level 0 can be encoded using 40 bits (5 bytes).

Observe that representations can be switched during the computation if at some level another representation becomes more efficient: we can switch from one of the above to compressed positions and we can switch from any other representation to value-only representation. The FSBday implementation switches from dynamic recomputation to value-only representation; it omits the possibility of compressed positions and value-and-positions representation.

9.3.3 What list sizes can be handled?

To estimate the storage requirements it is convenient to consider *dynamic recomputation* (storing positions only) because in this case the amount of required storage is constant over all levels and this representation has the smallest memory consumption on level 0.
As described before the attack starts with 16 lists of size 2^{38}, each containing bit strings of length $r' = 190$. However, storing 16 lists with 2^{38} entries, each entry encoded in 5 bytes requires 20 TB of storage space.
The computer cluster used for the attack consists of 8 nodes with a storage space of 700 GB each[1]. The goal is to adapt the generalized birthday attack to cope with total storage limited to 5.5 TB.

[1]Units such as GB, TB, PB and EB are always assumed to have base 1024 instead of 1000. In particular, this chapter assumes 700 GB as the size of a hard disk advertised as 750 GB.

The 16 lists on Level 0 need at least 5 bytes per list entry in order to encode positions; in order to store 16 lists on the cluster at most $5.5 \cdot 2^{40}/2^4/5 = 1.1 \times 2^{36}$ entries per list can be allowed. Note that some of the disk space is used for the operating system and so a straightforward implementation would use lists of size 2^{36}. First computing one half tree and switching to compressed-positions representation on level 2 would still not allow us to use lists of size 2^{37}.
At the beginning for each list 2^{40} entries are generated and 4 bits are clamped following [Ber07] resulting in 2^{36} values. These values have a length of 188 bits represented through 5 bytes holding the positions from the matrix. Clamping 36 bits in each of the 3 steps leaves two lists of length 2^{36} with 80 nonzero bits. A collision can be expected after about 256.5 runs of the attack following Remark 9.12.
The only way of increasing the list size to 2^{37} and thus reducing the number of runs is to use value-only representation on higher levels.

9.3.4 Clamping constants and collision computing

The main idea of the attack strategy is to distinguish between the task of

- finding clamping constants that yield a final collision, and
- the task of actually computing the collision.

The following remarks outline this idea.

Remark 9.21 (Finding appropriate clamping constants)**.** The idea is to find a particular set of clamping constants producing a collision. For this task it is not necessary to store the positions as the question is only whether the chosen clamping constants yield a collision or not. Whenever storing the value needs less space than storing positions one can thus *compress* entries by switching representation from positions to values. As a side effect this speeds up the computations because less data has to be loaded and stored.

Starting from lists $L_{0,0}, \ldots, L_{0,7}$, each containing 2^{37} entries first list $L_{3,0}$ is computed on 8 nodes (see Figure 9.3). This list has entries with 78 remaining bits each. The attack implementation presorts these entries on hard disk according to 9 bits that do not have to be stored. Another 3 bits are determined by the node holding the data so only 66 bits or 9 bytes of each entry have to be stored, yielding a total storage requirement of 1152 GB versus 5120 GB necessary for storing entries in positions-only representation.
Continue with the computation of list $L_{2,2}$, which has entries of 115 remaining bits. Again 9 of these bits do not have to be stored due to presorting, 3 are determined by the node, so only 103 bits or 13 bytes have to be stored, yielding a storage requirement of 1664 GB instead of 2560 GB for uncompressed entries.
After these lists have been stored persistently on disk, the attack proceeds with the computation of list $L_{2,3}$, then $L_{3,1}$ and finally check whether $L_{4,0}$ contains at least one element. These computations require another 2560 GB.

Therefore the total amount of storage sums up to 1152 GB + 1664 GB + 2560 GB = 5376 GB; obviously all data fits onto the hard disk of the 8 nodes.
If a computation with given clamping constants is not successful, only the computation of $L_{2,3}$ is repeated; this time with different clamping constants. The lists $L_{3,0}$ and $L_{2,2}$ do not have to be computed again. All combinations of clamping values for lists $L_{0,12}$ to $L_{0,15}$ summing up to 0 are allowed. Therefore there are a large number of valid clamp-bit combinations.
With 37 bits clamped on every level and 3 clamped through precomputation there are 4 uncontrolled bits in the end and therefore, according to Remark 9.12 one should expect 16.5 runs of this algorithm.

Remark 9.22 (Computing the matrix positions of the collision)**.** After appropriate clamping constants have been found in the first phase of the attack, the second phase recomputes lists $L_{3,0}$ and $L_{3,1}$ without compression to obtain the matrix positions. For this task only positions are stored which requires dynamic recomputation of the corresponding values. On level 0 and level 1 this is the most space-efficient approach and no significant speedup is expected from switching to compressed-positions representation on higher levels. In total one half-tree computation requires 5120 GB of storage, hence, they have to be performed one after the other on 8 nodes.

9.4 Results and evaluation

Niederhagen and Schwabe implemented the FSB_{48} attack which was successfully carried out at Technische Universiteit Eindhoven. The code can be found at `http://www.polycephaly.org/fsbday`. This section shows the results and discusses the security analysis in the FSB proposal.

9.4.1 Cost estimates and measurements

This section presents the estimates, before starting the attack, of the amount of time that the attack would need; measurements of the amount of time actually consumed by the attack; and comments on how different amounts of storage would have changed the attack time.
The estimates for the time of the attack are the following.

- (Phase one) As described before the first major step is to compute a set of clamping values which leads to a collision. In this first step entries are stored by positions on level 0 and 1 and from level 2 on list entries consist of values.

  Computation of list $L_{3,0}$ takes about 32 hours and list $L_{2,2}$ about 14 hours, summing up to 46 hours. These computations need to be done only once.

  The time needed to compute list $L_{2,3}$ is about the same as for $L_{2,2}$ (14 hours), list $L_{3,1}$ takes about 4 hours and checking for a collision in lists $L_{3,0}$ and $L_{3,1}$ on level 4 about another 3.5 hours, summing up to about 21.5 hours. The

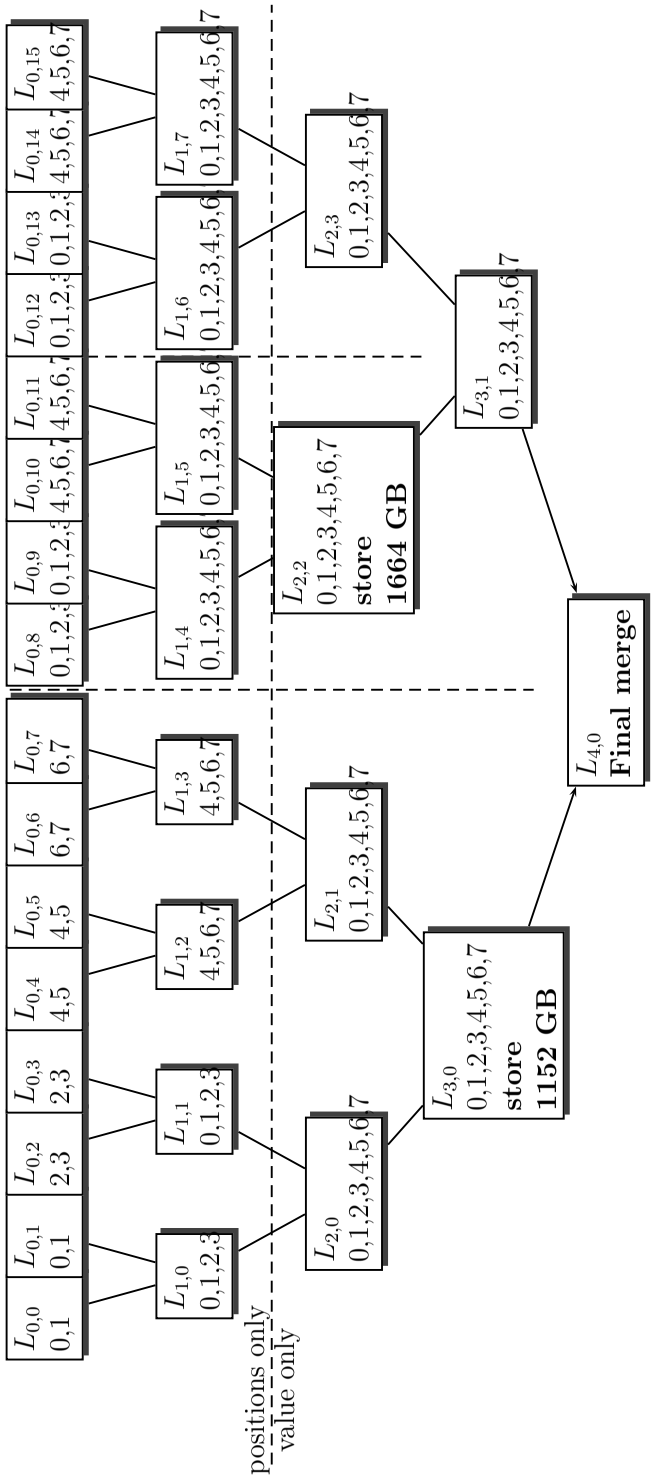

Figure 9.3: Structure of the generalized birthday attack against FSB_{48}: in each box the upper line denotes the list, the lower line gives the nodes holding fractions of this list

expected value of repetitions of these steps is 16.5 and we thus expected them to take about 355 hours.

- (Phase two) Finally, computing the matrix positions after finding a collision requires recomputation with uncompressed lists. Note that the comparison of the final two lists $L_{3,0}$ and $L_{3,1}$ can be aborted as soon as a collision is found. In the worst case this computation with uncompressed (positions-only) entries takes 33 hours for each half-tree, summing up to 66 hours.

- (Total) Before starting the attack we expected to find a collision for the FSB_{48} compression function using the cluster in 467 hours or about 19.5 days.

In the end the FSBday attack found successful clamping constants after only five iterations (instead of the expected 16.5). In total the first phase of the attack took 5 days, 13 hours and 20 minutes.
Recomputation of the positions in $L_{3,0}$ took 1 day, 8 hours and 22 minutes and recomputation of the positions in $L_{3,1}$ took 1 day, 2 hours and 11 minutes. In total the attack took 7 days, 23 hours and 53 minutes.

Remark 9.23 (Output)**.** The matrix used in the attack is the pseudo-random matrix defined in Remark 9.6. The FSBday attack found that matrix positions (734, 15006, 20748, 25431, 33115, 46670, 50235, 51099, 70220, 76606, 89523, 90851, 99649, 113400, 118568, 126202, 144768, 146047, 153819, 163606, 168187, 173996, 185420, 191473 198284, 207458, 214106, 223080, 241047, 245456, 247218, 261928, 264386, 273345, 285069, 294658, 304245, 305792, 318044, 327120, 331742, 342519, 344652, 356623, 364676, 368702, 376923, 390678) yield a collision.

9.4.2 Time-storage tradeoffs

As described in Section 9.3, the main restriction on the attack strategy was the total amount of background storage.
Given 10496 GB of storage at hand the attack could have been started with lists of size 2^{38}, again using the compression techniques described in Section 9.3. As described in Section 9.3 this would give exactly one expected collision in the last merge step and thus reduce the expected number of required runs to find the right clamping constants from 16.5 to 1.58. With a total storage of 20 TB a straightforward Wagner attack without compression could be carried out; this would eliminate the need to recompute two half trees at the end.
Increasing the size of the background storage even further would eventually allow to store list entry values alongside the positions and thus eliminate the need for dynamic recomputation. However, the bottleneck of the performance of the attack is the hard-disk throughput rather than CPU time so this measure is not likely to yield any improvement of the performance.
On clusters with even less background storage the computation time will (asymptotically) increase by a factor of 16 with each halving of the storage size. For example

a cluster with 2688 GB of storage can only handle lists of size 2^{36}. The attack would then require (expected) 256.5 computations to find appropriate clamping constants. Of course the time required for one half-tree computation depends on the amount of data. As long as the performance is mainly bottlenecked by hard-disk (or network) throughput the running time is linearly dependent on the amount of data, i.e., a Wagner computation involving 2 half-tree computations with lists of size 2^{38} is about 4.5 times as fast as a Wagner computation involving 18 half-tree computations with lists of size 2^{37}.

9.4.3 Scalability analysis

The attack described in this chapter including the variants discussed in the previous section are much more expensive in terms of time and especially memory than a brute-force attack against the 48-bit hash function FSB_{48}. In conclusion, the FSB designers overestimated the power of Wagner's attack for all versions of FSB; not only FSB_{48}. They assumed that the cost can be estimated by roughly the size of one of the lists on level 0, i.e., $2^{r/i}r$ where 2^{i-1} is the number of lists and r the output length of the compression function. Schwabe implemented the hash function FSB_{48} and found a collision by repeatedly producing outputs of the hash function in less than 1 minute and 20 seconds; see his Ph.D. thesis [Sch11, Section 7.6] for details. This section gives estimates of the power of Wagner's attack against the larger versions of FSB. Table 9.1 gives the parameters of all FSB hash functions.

Table 9.1: Parameters of the FSB variants and estimates for the cost of generalized birthday attacks against the compression function. For Pollard's variant the number of lists is marked with an asterisk *. Storage is measured in bytes.

	n	w	r	# lists	list size	bits/ entry	total storage	time
FSB_{48}	3×2^{17}	24	192	16	2^{38}	190	$5 \cdot 2^{42}$	$5 \cdot 2^{42}$
FSB_{160}	7×2^{18}	112	896	16	2^{127}	632	$17 \cdot 2^{131}$	$17 \cdot 2^{131}$
				16^*	2^{60}	630	$9 \cdot 2^{64}$	$9 \cdot 2^{224}$
FSB_{224}	2^{21}	128	1024	16	2^{177}	884	$24 \cdot 2^{181}$	$24 \cdot 2^{181}$
				16^*	2^{60}	858	$13 \cdot 2^{64}$	$13 \cdot 2^{343}$
FSB_{256}	23×2^{16}	184	1472	16	2^{202}	1010	$27 \cdot 2^{206}$	$27 \cdot 2^{206}$
				16^*	2^{60}	972	$14 \cdot 2^{64}$	$14 \cdot 2^{386}$
				32^*	2^{56}	1024	$18 \cdot 2^{60}$	$18 \cdot 2^{405}$
FSB_{384}	23×2^{16}	184	1472	16	2^{291}	1453	$39 \cdot 2^{295}$	$39 \cdot 2^{295}$
				32^*	2^{60}	1467	$9 \cdot 2^{65}$	$18 \cdot 2^{618.5}$
FSB_{512}	31×2^{16}	248	1987	16	2^{393}	1962	$53 \cdot 2^{397}$	$53 \cdot 2^{397}$
				32^*	2^{60}	1956	$12 \cdot 2^{65}$	$24 \cdot 2^{863}$

A straightforward Wagner attack against FSB_{160} uses 16 lists of size 2^{127} containing elements with 632 bits. The entries of these lists are generated as sums of 10 columns from 5 blocks, yielding 2^{135} possibilities to generate the entries. Precomputation includes clamping of 8 bits. Each entry then requires 135 bits of storage so each list occupies more than 2^{131} bytes. For comparison, the largest currently available storage systems offer a few petabytes (2^{50} bytes) of storage.
To limit the amount of memory one can instead generate, e.g., 32 lists of size 2^{60}, where each list entry is the sum of 5 columns from 2.5 blocks, with 7 bits clamped during precomputation. Each list entry then requires 67 bits of storage.
Clamping 60 bits in each step leaves 273 bits uncontrolled so the Pollard variant of Wagner's algorithm (see Section 9.2.2) becomes more efficient than the plain attack. This attack generates 16 lists of size 2^{60}, containing entries which are the sum of 5 columns from 5 distinct blocks each. This gives us the possibility to clamp 10 bits through precomputation, leaving $B = 630$ bits for each entry on level 0.
The time required by this attack is approximately 2^{224} (see (9.2)). This is substantially faster than a brute-force collision attack on the compression function, but is clearly much slower than a brute-force collision attack on the hash function, and even slower than a brute-force *preimage* attack on the hash function.
Similar statements hold for the other full-size versions of FSB. Table 9.1 gives rough estimates for the time complexity of Wagner's attack without storage restriction and with storage restricted to a few hundred exabytes (2^{60} entries per list). These estimates only consider the number and size of lists being a power of 2 and the number of bits clamped in each level being the same. The estimates ignore the time complexity of precomputation. Time is computed according to (9.1) and (9.2) with the size of level-0 entries (in bytes) as a constant factor.
Although fine-tuning the attacks might give small speedups compared to the estimates, it is clear that the compression function of FSB is oversized, assuming that Wagner's algorithm in a somewhat memory-restricted environment is the most efficient attack strategy.

Bibliography

[AB98] Alexei E. Ashikhmin and Alexander Barg. Minimal vectors in linear codes. *IEEE Transactions on Information Theory*, 44(5):2010–2017, 1998. ISSN 0018–9448. (Cited on page 87).

[ABC10] Daniel Augot, Morgan Barbier, and Alain Couvreur. List-decoding of binary Goppa codes up to the binary Johnson bound. *CoRR*, abs/1012.3439, 2010. (Cited on page 76).

[AFG+09] Daniel Augot, Matthieu Finiasz, Philippe Gaborit, Stéphane Manuel, and Nicolas Sendrier. SHA-3 Proposal: FSB, 2009. `http://www-rocq.inria.fr/secret/CBCrypto/index.php?pg=fsb` (accessed version March 26, 2011). (Cited on pages 3, 5, 156, 157, 158, and 164).

[AFS03] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A fast provably secure cryptographic hash function, 2003. `http://eprint.iacr.org/2003/230`. (Cited on pages 3 and 155).

[AFS05] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. In Ed Dawson and Serge Vaudenay, editors, *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 64–83. Springer-Verlag Berlin Heidelberg, 2005. ISBN 3-540-28938-0. (Cited on pages 3, 155, 157, 160, and 164).

[AM87] Carlisle M. Adams and Henk Meijer. Security-related comments regarding McEliece's public-key cryptosystem. In Carl Pomerance, editor, *CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 224–228. Springer-Verlag Berlin Heidelberg, 1987. ISBN 3-540-18796-0. See newer [AM89]. (Cited on pages 133 and 173).

[AM89] Carlisle M. Adams and Henk Meijer. Security-related comments regarding McEliece's public-key cryptosystem. *IEEE Transactions on Information Theory*, 35(2):454–455, 1989. See older [AM87]. (Cited on page 173).

[AM93] A. O. L. Atkin and Francois Morain. Finding suitable curves for the elliptic curve method of factorization. *Mathematics of Computation*, 60: 399–405, 1993. ISSN 0025–5718. (Cited on pages 58 and 59).

[Bar94] Alexander Barg. Some new NP complete coding problems. *Problemy Peredachi Informatsii*, 30(3):23–28, 1994. in Russian. (Cited on page 74).

[Bar98] Alexander Barg. *Complexity Issues in Coding Theory*, chapter 7 in [HPB98], pages 649–754. 1998. (Cited on pages 74 and 75).

[Bar06] Gregory V. Bard. Accelerating cryptanalysis with the Method of Four Russians. Cryptology ePrint Archive: Report 2006/251, 2006. `http://eprint.iacr.org/2006/251`. (Cited on page 91).

[BBD08] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-Quantum cryptography*. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 9783540887010. (Cited on page 187).

[BBJ+08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards curves. In Serge Vaudenay, editor, *AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer-Verlag Berlin Heidelberg, 2008. ISBN 3540681590. (Cited on pages 4, 9, 10, 13, 14, 15, 25, and 26).

[BBL10] Daniel J. Bernstein, Peter Birkner, and Tanja Lange. Starfish on strike. In Michel Abdalla and Paulo Barreto, editors, *LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 61–80. Springer-Verlag Berlin Heidelberg, 2010. ISBN 9783642147111. (Cited on page 45).

[BBLP] Daniel J. Bernstein, Peter Birkner, Tanja Lange, and Christiane Peters. EECM: ECM using Edwards curves. `http://eecm.cr.yp.to/index.html` (accessed version March 26, 2011). (Cited on pages 60 and 63).

[BBLP07] Daniel J. Bernstein, Peter Birkner, Tanja Lange, and Christiane Peters. Optimizing double-base elliptic-curve single-scalar multiplication. In Srinathan et al. [SRY07], pages 167–182. ISBN 978–3–540–77025–1. (Cited on pages 4, 9, 10, 28, 31, 35, and 37).

[BBLP08] Daniel J. Bernstein, Peter Birkner, Tanja Lange, and Christiane Peters. ECM using Edwards curves. Cryptology ePrint Archive: Report 2008/016, 2008. `http://eprint.iacr.org/2008/016`. (Cited on pages 4, 9, 10, 13, 45, 46, 49, 50, 60, 62, and 63).

[BCGO09] Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing key length of the McEliece cryptosystem. In Bart Preneel, editor, *AFRICACRYPT 2008*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97. Springer-Verlag Berlin Heidelberg, 2009. (Cited on pages 111, 118, 119, and 128).

[Ber84] Elwyn R. Berlekamp. *Algebraic Coding Theory*. Aegean Park Press, Laguna Hills, CA, USA, 1984. ISBN 0894120638. Revised edition; first edition McGraw-Hill, 1968. (Cited on page 76).

[Ber97] Thomas A. Berson. Failure of the McEliece public-key cryptosystem under message-resend and related-message attack. In Burton S. Kaliski Jr., editor, *CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 213–220. Springer-Verlag Berlin Heidelberg, 1997. ISBN 3-540-63384-7. (Cited on page 80).

[Ber02] Daniel J. Bernstein. Arbitrarily tight bounds on the distribution of smooth integers. In Michael A. Bennett, Bruce C. Berndt, Nigel Boston, Harold G. Diamond, Adolf J. Hildebrand, and Walter Philipp, editors, *Number Theory for the millenium*, pages 49–66. A. K. Peters, Natick, Massachusetts, 2002. ISBN 1–56881–126–8. (Cited on page 68).

[Ber07] Daniel J. Bernstein. Better price-performance ratios for generalized birthday attacks. In *Workshop Record of SHARCS'07: Special-purpose Hardware for Attacking Cryptographic Systems*, 2007. `http://cr.yp.to/papers.html#genbday`. (Cited on pages 160 and 166).

[Ber08a] Daniel J. Bernstein. Fast multiplication and its applications. volume 44 of *Mathematical Sciences Research Institute Publications*, pages 325–384. Cambridge University Press, 2008. ISBN 0521808545. (Cited on page 125).

[Ber08b] Daniel J. Bernstein. List decoding for binary Goppa codes, 2008. `http://cr.yp.to/papers.html#goppalist`. (Cited on pages 76 and 126).

[Ber10a] Daniel J. Bernstein. *EECM–MPFQ*, 2010. `http://eecm.cr.yp.to/mpfq.html` (accessed version March 26, 2011). (Cited on page 45).

[Ber10b] Daniel J. Bernstein. Grover vs. McEliece. In Sendrier [Sen10], pages 73–80. ISBN 978-3-642-12928-5. (Cited on page 3).

[BFvT02] Mario Blaum, Patrick G. Farrell, and Henk C. A. van Tilborg, editors. *Information, Coding and Mathematics*, volume 687 of *Kluwer International Series in Engineering and Computer Science*. Kluwer, 2002. ISBN 1402070799. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday. (Cited on pages 188 and 189).

[BK81] Richard P. Brent and H. T. Kung. The area-time complexity of binary multiplication. *Journal of the ACM*, 28:521–534, 1981. `http://wwwmaths.anu.edu.au/~brent/pub/pub055.html`. (Cited on page 141).

[BKvT99] Alexander Barg, Evgueni Krouk, and Henk C. A. van Tilborg. On the complexity of minimum distance decoding of long linear codes. *IEEE Transactions on Information Theory*, 45(5):1392–1405, 1999. (Cited on pages 87, 137, and 138).

[BL05] Thierry P. Berger and Pierre Loidreau. How to mask the structure of codes for a cryptographic use. *Designs, Codes and Cryptography*, 35(1): 63–79, 2005. ISSN 0925–1022. (Cited on page 111).

[BL07a] Daniel J. Bernstein and Tanja Lange. Explicit-formulas database, 2007. `http://hyperelliptic.org/EFD` (accessed version March 26, 2011). (Cited on pages 25, 31, 32, and 33).

[BL07b] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer-Verlag Berlin Heidelberg, 2007. ISBN 3540768998. (Cited on pages 9, 10, 11, 12, 13, 14, 23, 25, 26, 32, and 40).

[BL07c] Daniel J. Bernstein and Tanja Lange. Inverted Edwards coordinates. In Serdar Boztas and Hsiao-Feng Lu, editors, *AAECC–17*, volume 4851 of *Lecture Notes in Computer Science*, pages 20–27. Springer-Verlag Berlin Heidelberg, 2007. ISBN 3540772235. (Cited on pages 9, 10, 15, 16, 26, and 32).

[BL08] Daniel J. Bernstein and Tanja Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In Gary L. Mullen, Daniel Panario, and Igor E. Shparlinski, editors, *Finite Fields and Applications*, volume 461 of *Contemporary Mathematics*, pages 1–19. American Mathematical Society, 2008. ISBN 0821843095. (Cited on pages 35 and 50).

[BL10] Daniel J. Bernstein and Tanja Lange. A complete set of addition laws for incomplete Edwards curves. *Journal of Number Theory*, In Press, Corrected Proof, 2010. ISSN 0022-314X. `http://eprint.iacr.org/2009/580`. (Cited on pages 9, 11, 16, 17, and 18).

[BLF08] Daniel J. Bernstein, Tanja Lange, and Reza Rezaeian Farashahi. Binary edwards curves. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 244–265. Springer-Verlag Berlin Heidelberg, 2008. ISBN 978-3-540-85052-6. (Cited on page 9).

[BLN+09] Daniel J. Bernstein, Tanja Lange, Ruben Niederhagen, Christiane Peters, and Peter Schwabe. FSBday: Implementing Wagner's generalized birthday attack against the SHA-3 round-1 candidate FSB. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT 2009*, volume 5922 of

Lecture Notes in Computer Science, pages 18–38. Springer-Verlag Berlin Heidelberg, 2009. ISBN 978-3-642-10627-9. (Cited on pages 5, 155, 158, and 163).

[BLP08] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In Johannes Buchmann and Jintai Ding, editors, *PQCrypto 2008*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer-Verlag Berlin Heidelberg, 2008. ISBN 3540884025. (Cited on pages 4, 72, 82, 83, 87, 90, 92, 93, 99, 106, 110, 111, 114, and 118).

[BLP10] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: ball-collision decoding, 2010. `http://eprint.iacr.org/2010/585`. (Cited on pages 5, 100, 134, 136, 137, 138, 139, and 147).

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Wild McEliece. In Alex Biryukov, Guang Gong, and Douglas Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 143–158. Springer-Verlag Berlin Heidelberg, 2011. ISBN 978-3-642-19573-0. (Cited on pages 4, 72, 112, and 121).

[BLPvT09] Daniel J. Bernstein, Tanja Lange, Christiane Peters, and Henk C. A. van Tilborg. Explicit bounds for generic decoding algorithms for code-based cryptography. In Alexander Kholosha, Eirik Rosnes, and Matthew Parker, editors, *Pre-proceedings of WCC 2009*, pages 168–180. Bergen, 2009. (Cited on pages 4, 83, 100, 102, 107, and 140).

[BMvT78] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24:384–386, 1978. (Cited on pages 3 and 74).

[BR03] Paulo S. L. M. Barreto and Vincent Rijmen. The WHIRLPOOL Hashing Function, 2003. Submitted to NESSIE, September 2000, revised version May 2003. `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html` (accessed version March 26, 2011). (Cited on page 156).

[Bra39] Alfred Brauer. On addition chains. *Bulletin of the American Mathematical Society*, 45:736–739, 1939. (Cited on page 36).

[Bre86] Richard P. Brent. Some integer factorization algorithms using elliptic curves. *Australian Computer Science Communications*, 8:149–163, 1986. ISSN 0157–3055. (Cited on pages 48, 66, 67, and 189).

[BSS99] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, editors. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999. ISBN 0521653746. (Cited on page 32).

[CC81] George C. Clark, Jr. and J. Bibb Cain. *Error-correcting coding for digital communication*. Plenum, New York, 1981. ISBN 0-306-40615-2. (Cited on page 90).

[CC86] David V. Chudnovsky and Gregory V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, 7:385–434, 1986. (Cited on pages 48 and 49).

[CC93] Herve Chabanne and Bernard Courteau. Application de la méthode de décodage itérative d'Omura à la cryptanalyse du système de McEliece. Technical report, 1993. Université de Sherbrooke, Rapport de Recherche, number 122. (Cited on page 90).

[CC94] Anne Canteaut and Hervé Chabanne. A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. In Pascale Charpin, editor, *EUROCODE 94*, pages 169–173, 1994. (Cited on page 90).

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. ISSN 0018–9448. (Cited on pages 81, 88, 90, 92, 93, 94, 96, 102, and 110).

[CF10] Carlos Cid and Jean-Charles Faugere, editors. *Proceedings of the 2nd International Conference on Symbolic Computation and Cryptography (SCC 2010), Royal Holloway, University of London, Egham, June, 2010*. Royal Holloway, University of London, 2010. `http://scc2010.rhul.ac.uk/scc2010-proceedings.pdf`. (Cited on pages 181 and 186).

[CFD05] Henri Cohen, Gerhard Frey, and Christophe Doche, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. 2005. ISBN 1584885181. (Cited on pages 9, 180, and 181).

[CFS01] Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a McEliece-based digital signature scheme. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 157–174. Springer-Verlag Berlin Heidelberg, 2001. ISBN 3-540-42987-5. (Cited on page 127).

[CGF91] John T. Coffey, Rodney M. Goodman, and Patrick G. Farrell. New approaches to reduced complexity decoding. *Discrete and Applied Mathematics*, 33:43–60, 1991. (Cited on page 140).

[Cha93] Florent Chabaud. Asymptotic analysis of probabilistic algorithms for finding short codewords. In Paul Camion, Pascale Charpin, and Sami

Harari, editors, *Eurocode '92*, pages 175–183. Springer-Verlag Berlin Heidelberg, 1993. ISBN 3-211-82519-3. (Cited on page 90).

[CJ04] Jean-Sébastien Coron and Antoine Joux. Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive: Report 2004/013, 2004. `http://eprint.iacr.org/2004/013`. (Cited on pages 155 and 163).

[CP91] Paul Camion and Jacques Patarin. The knapsack hash function proposed at crypto'89 can be broken. In Donald W. Davies, editor, *EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 39–53. Springer-Verlag Berlin Heidelberg, 1991. ISBN 3–540–54620–0. (Cited on page 159).

[CS98] Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199. Springer-Verlag Berlin Heidelberg, 1998. ISBN 3540651098. (Cited on pages 88, 90, 92, 93, 94, and 96).

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. (Cited on pages 1 and 35).

[DI06] Christophe Doche and Laurent Imbert. Extended Double-Base Number System with applications to Elliptic Curve Cryptography. In Rana Barua and Tanja Lange, editors, *INDOCRYPT 2006*, volume 4329 of *Lecture Notes in Computer Science*, pages 335–348. Springer-Verlag Berlin Heidelberg, 2006. ISBN 3540497676. (Cited on pages 32, 36, 37, 38, and 39).

[Dic30] Karl Dickman. On the frequency of numbers containing primes of a certain relative magnitude. *Arkiv för Matematik, Astronomi och Fysik*, 22:1–14, 1930. ISSN 0365–4133. (Cited on page 66).

[DIK06] Christophe Doche, Thomas Icart, and David R. Kohel. Efficient scalar multiplication by isogeny decompositions. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 191–206. Springer-Verlag Berlin Heidelberg, 2006. ISBN 3540338519. (Cited on page 33).

[DIM05] Vassil Dimitrov, Laurent Imbert, and Pradeep K. Mishra. Efficient and secure elliptic curve point multiplication using double-base chains. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 59–78. Springer-Verlag Berlin Heidelberg, 2005. ISBN 3540306846. (Cited on pages 32, 35, 37, 38, and 39).

[DL05] Christophe Doche and Tanja Lange. *Arithmetic of elliptic curves*, chapter 13 in [CFD05], pages 267–302. 2005. (Cited on pages 21, 32, and 35).

[Doc05] Christophe Doche. *Exponentiation*, chapter 9 in [CFD05], pages 145–168. 2005. (Cited on page 50).

[Dum89] Ilya Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25(1):24–32, 1989. (Cited on page 86).

[Dum91] Ilya Dumer. On minimum distance decoding of linear codes. In Grigori A. Kabatianskii, editor, *Proceedings 5th Joint Soviet-Swedish International Workshop Information Theory, Moscow*, pages 50–52, 1991. (Cited on page 86).

[Duq07] Sylvain Duquesne. Improving the arithmetic of elliptic curves in the Jacobi model. *Information Processing Letters*, 104:101–105, 2007. (Cited on page 33).

[ECR09] ECRYPT-II yearly report on algorithms and keysizes (2008-2009). Technical report, ECRYPT-II – European Network of Excellence in Cryptology, EU FP7, ICT-2007-216676, 2009. Published as deliverable D.SPA.7, `http://www.ecrypt.eu.org/documents/D.SPA.7.pdf`. (Cited on pages 2 and 11).

[Edw07] Harold M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007. (Cited on pages 9 and 11).

[EGHP09] Thomas Eisenbarth, Tim Güneysu, Stefan Heyse, and Christof Paar. MicroEliece: McEliece for embedded devices. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 49–64. Springer-Verlag Berlin Heidelberg, 2009. ISBN 978-3-642-04137-2. (Cited on pages 3 and 4).

[EOS06] Daniela Engelbert, Raphael Overbeck, and Arthur Schmidt. A summary of McEliece-type cryptosystems and their security. Cryptology ePrint Archive: Report 2006/162, 2006. `http://eprint.iacr.org/2006/162`. (Cited on page 80).

[FGS07] Matthieu Finiasz, Philippe Gaborit, and Nicolas Sendrier. Improved fast syndrome based cryptographic hash functions. In *Proceedings of ECRYPT Hash Workshop 2007*, 2007. `http://www-roc.inria.fr/secret/Matthieu.Finiasz/research/2007/finiasz-gaborit-sendrier-ecrypt-hash-workshop07.pdf`. (Cited on page 155).

[Fin08] Matthieu Finiasz. Syndrome based collision resistant hashing. In Johannes Buchmann and Jintai Ding, editors, *PQCrypto 2008*, volume 5299 of *Lecture Notes in Computer Science*, pages 137–147. Springer-Verlag Berlin Heidelberg, 2008. ISBN 3540884025. (Cited on page 155).

[FKI07] Marc P. C. Fossorier, Kazukuni Kobara, and Hideki Imai. Modeling bit flipping decoding based on nonorthogonal check sums with application to iterative decoding attack of McEliece cryptosystem. *IEEE Transactions on Information Theory*, 53(1):402–411, 2007. ISSN 0018–9448. (Cited on page 144).

[FL05] Gerhard Frey and Tanja Lange. *Background on Curves and Jacobians*, chapter 4 in [CFD05], pages 45–85. 2005. (Cited on page 9).

[FM08] Cédric Faure and Lorenz Minder. Cryptanalysis of the McEliece cryptosystem over hyperelliptic codes. In *Proceedings of the 11th international workshop on Algebraic and Combinatorial Coding Theory, ACCT 2008*, pages 99–107, 2008. (Cited on page 123).

[FOPT10] Jean-Charles Faugére, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer-Verlag Berlin Heidelberg, 2010. ISBN 3642131891. (Cited on pages 3, 111, 118, and 128).

[FS09] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912, pages 88–105. Springer-Verlag Berlin Heidelberg, 2009. ISBN 3642103650. (Cited on pages 100, 101, 111, 127, 138, 141, 143, and 144).

[GL10] Valerie Gauthier Umana and Gregor Leander. Practical key recovery attacks on two McEliece variants. In Cid and Faugere [CF10], pages 27–44. `http://eprint.iacr.org/2009/509`. (Cited on pages 3, 111, 118, and 128).

[Gop70] Valery D. Goppa. A new class of linear error correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970. (Cited on pages 75 and 76).

[Gop71] Valery D. Goppa. Rational representation of codes and (L, g)-codes. *Problemy Peredachi Informatsii*, 7(3):41–49, 1971. (Cited on page 75).

[Gop77] Valery D. Goppa. Codes associated with divisors. *Problemy Peredachi Informatsii*, 13(1):33–39, 1977. (Cited on page 77).

[Gra] Torbjörn Granlund. *The GNU MP Bignum Library*. `http://gmplib.org/` (accessed version March 26, 2011). (Cited on pages 45 and 96).

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC'96*, pages 212–219. ACM, New York, NY, USA, 1996. ISBN 0897917855. (Cited on page 3).

[Gro97] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79:325–328, 1997. (Cited on page 3).

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999. ISSN 0018–9448. (Cited on pages 76 and 126).

[GT] Pierrick Gaudry and Emmanuel Thomé. *MPFQ – A finite field library*. See also [GT07]. `http://mpfq.gforge.inria.fr/` (accessed version March 26, 2011). (Cited on page 45).

[GT07] Pierrick Gaudry and Emmanuel Thomé. The $\mathtt{mp}\mathbb{F}_q$ library and implementing curve-based key exchanges. In *SPEED: software performance enhancement for encryption and decryption*, pages 49–64, 2007. `http://www.loria.fr/~gaudry/papers.en.html`. (Cited on page 182).

[Har77] Robin Hartshorne. *Algebraic Geometry*, volume 52 of *Graduate texts in mathematics*. Springer-Verlag, Berlin, Heidelberg, 1977. ISBN 0387902449. (Cited on page 17).

[HCD07] Huseyin Hisil, Gary Carter, and Ed Dawson. New formulae for efficient elliptic curve arithmetic. In Srinathan et al. [SRY07], pages 138–151. ISBN 978–3–540–77025–1. (Cited on pages 28, 32, and 33).

[His10] Huseyin Hisil. *Elliptic curves, group law, and efficient computation*. Ph.D. thesis, Queensland University of Technology, 2010. `http://eprints.qut.edu.au/33233/`. (Cited on page 31).

[HLP+] Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Philippe Théveny, and Paul Zimmermann. *The GNU MPFR library*. `http://www.mpfr.org/` (accessed version March 26, 2011). (Cited on page 96).

[HMV04] Darrel Hankerson, Alfred J. Menezes, and Scott A. Vanstone. *Guide to elliptic curve cryptography*. Springer-Verlag, Berlin, Heidelberg, 2004. ISBN 038795273X. (Cited on page 32).

[HP03] W. Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521782805. (Cited on pages 72, 74, 75, and 76).

[HPB98] W. Cary Huffman, Vera Pless, and Richard A. Brualdi. *Handbook of Coding Theory*. Elsevier/North Holland, Amsterdam, 1998. ISBN 0-444-50088-X. (Cited on page 174).

[HWCD08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards curves revisited. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 326–343. Springer-Verlag Berlin Heidelberg, 2008. ISBN 3540892540. (Cited on pages 9, 10, 14, 15, 27, and 28).

[IP10] Laurent Imbert and Fabrice Philippe. Strictly chained (p, q)-ary partitions. *Contributions to Discrete Mathematics*, 5(2), 2010. ISSN 1715–0868. (Cited on page 31).

[Jab01] Abdulrahman Al Jabri. A statistical decoding algorithm for general linear block codes. In Bahram Honary, editor, *Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 1–8. Springer-Verlag Berlin Heidelberg, 2001. ISBN 3-540-43026-1. (Cited on page 87).

[JJ02] Thomas Johansson and Fredrik Jonsson. On the complexity of some cryptographic problems based on the general decoding problem. *IEEE Transactions on Information Theory*, 48:2669–2678, 2002. ISSN 0018–9448. (Cited on page 144).

[JL06] Antoine Joux and Reynald Lercier. The function field sieve in the medium prime case. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 254–270. Springer-Verlag Berlin Heidelberg, 2006. ISBN 3-540-34546-9. (Cited on page 3).

[JLSV06] Antoine Joux, Reynald Lercier, Nigel P. Smart, and Frederik Vercauteren. The number field sieve in the medium prime case. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 326–344. Springer-Verlag Berlin Heidelberg, 2006. ISBN 3-540-37432-9. (Cited on page 3).

[JM96] Heeralal Janwa and Oscar Moreno. McEliece public key cryptosystems using algebraic-geometric codes. *Designs, Codes and Cryptography*, 8(3): 293–307, 1996. ISSN 0925–1022. (Cited on pages 111, 121, and 123).

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer-Verlag Berlin Heidelberg, 2010. (Cited on page 142).

[KI01] Kazukuni Kobara and Hideki Imai. Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *Lecture Notes in Computer*

Science, pages 19–35. Springer-Verlag Berlin Heidelberg, 2001. ISBN 3540416587. (Cited on pages 80, 118, 126, and 127).

[Knu97] Donald E. Knuth. *The Art of Computer Programming. Vol. 2, Seminumerical Algorithms*. Addison-Wesley Publishing Co., Reading, Mass., third edition, 1997. Addison-Wesley Series in Computer Science and Information Processing. (Cited on page 162).

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. (Cited on page 2).

[Kro89] Evgueni A. Krouk. Decoding complexity bound for linear block codes. *Problemy Peredachi Informatsii*, 25(3):103–107, 1989. (Cited on page 84).

[Kru10] Alexander Kruppa. *Améliorations de la multiplication et de la factorisation d'entier*. Ph.D. thesis, Université Henri Poincaré - Nancy I, 2010. (Cited on pages 46 and 60).

[KT89] Gregory L. Katsman and Michael A. Tsfasman. A remark on algebraic geometric codes. In I. Martin Isaacs, Alexander I. Lichtman, Donald S. Passman, Sudarshan K. Sehgal, Neil J. A. Sloane, and Hans J. Zassenhaus, editors, *Representation theory, group rings, and coding theory*, volume 93, pages 197–199. American Mathematical Society, 1989. ISBN 0821850989. (Cited on page 123).

[Lan] Tanja Lange. Coding and Cryptography Computer Cluster (CCCC). `http://www.win.tue.nl/cccc/` (accessed version March 26, 2011). (Cited on page 163).

[LB88] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In Christoph G. Günther, editor, *EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer-Verlag Berlin Heidelberg, 1988. ISBN 0387502513. (Cited on page 85).

[LDW94] Yuan Xing Li, Robert H. Deng, and Xin Mei Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994. ISSN 0018–9448. (Cited on page 79).

[Len87a] Hendrik W. Lenstra, Jr. Elliptic curves and number-theoretic algorithms. In Andrew M. Gleason, editor, *Proceedings of the International Congress of Mathematicians*, volume 1, pages 99–120. American Mathematical Society, Providence, 1987. ISBN 0821801104. (Cited on page 47).

[Len87b] Hendrik W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987. (Cited on pages 45 and 67).

[Leo88] Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988. (Cited on pages 84 and 86).

[Maz77] Barry Mazur. Modular curves and the Eisenstein ideal. *Publications Mathématiques de L'IHÉS*, 47:33–186, 1977. ISSN 0073–8301. (Cited on page 51).

[Maz78] Barry Mazur. Rational isogenies of prime degree. *Inventiones Mathematicae*, 44:129–162, 1978. ISSN 0020–9910. (Cited on page 51).

[MB09] Rafael Misoczki and Paulo S. L. M. Barreto. Compact McEliece keys from Goppa codes. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 376–392. Springer-Verlag Berlin Heidelberg, 2009. ISBN 3642054439. (Cited on pages 3, 111, 118, 119, 121, and 128).

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory, 1978. Jet Propulsion Laboratory DSN Progress Report 42–44. `http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF`. (Cited on pages 3, 71, 79, 81, 98, and 133).

[McK99] James McKee. Subtleties in the distribution of the numbers of points on elliptic curves over a finite prime field. *Journal of the London Mathematical Society*, 59(2):448–460, 1999. (Cited on pages 47 and 67).

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag Berlin Heidelberg, 1986. ISBN 3540164634. (Cited on page 2).

[Min07] Lorenz Minder. *Cryptography based on error-correcting codes*. Ph.D. thesis, EPFL, no. 3846, 2007. (Cited on page 123).

[MK08] Peter L. Montgomery and Alexander Kruppa. Improved stage 2 to p±1 factoring algorithms. In Alfred J. van der Poorten and Andreas Stein, editors, *ANTS-VIII*, volume 5011 of *Lecture Notes in Computer Science*, pages 180–195. Springer-Verlag Berlin Heidelberg, 2008. ISBN 9783540794554. (Cited on page 46).

[Mon83] Peter L. Montgomery. Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. Technical report, 1983. Revised version January, 1992. `http://cr.yp.to/bib/1992/montgomery-lucas.ps`. (Cited on page 49).

[Mon87] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987. (Cited on pages 21, 24, 48, 58, and 189).

[Mon92] Peter L. Montgomery. *An FFT extension of the elliptic curve method of factorization*. Ph.D. thesis, University of California at Los Angeles, 1992. (Cited on pages 58, 60, and 66).

[MS77] F. Jessie MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. Elsevier/North Holland, Amsterdam, 1977. ISBN 0444851933. (Cited on pages 72, 74, 75, 76, and 122).

[MS09] Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 586–595. SIAM, 2009. See also newer version [MS10]. (Cited on pages 160 and 186).

[MS10] Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm, 2010. Newer version of [MS09]. `http://www.cs.berkeley.edu/~sinclair/ktree.pdf`. (Cited on pages 117 and 186).

[NCBB10] Robert Niebuhr, Pierre-Louis Cayrel, Stanislav Bulygin, and Johannes Buchmann. On lower bounds for information set decoding over $\mathbf{f}_q$. In Cid and Faugere [CF10], pages 143–157. `http://scc2010.rhul.ac.uk/scc2010-proceedings.pdf`. (Cited on page 117).

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory. Problemy Upravlenija i Teorii Informacii*, 15(2):159–166, 1986. ISSN 0370–2529. (Cited on pages 79 and 121).

[NIS07] NIST – National Institute of Standards and Technology. SHA-3 cryptographic hash algorithm competition, 2007. `http://csrc.nist.gov/groups/ST/hash/` (accessed version March 26, 2011). (Cited on page 155).

[NPS10] Michael Naehrig, Christiane Peters, and Peter Schwabe. SHA-2 will soon retire, 2010. `http://www.anagram.com/jcrap/Volume_7/`. (Cited on page 155).

[OKS00] Katsuyuki Okeya, Hiroyuki Kurumatani, and Kouichi Sakurai. Elliptic curves with the Montgomery-form and their cryptographic applications. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 238–257. Springer-Verlag Berlin Heidelberg, 2000. ISBN 3540669671. (Cited on page 24).

[OS08] Raphael Overbeck and Nicolas Sendrier. *Code-based cryptography*, chapter in [BBD08], pages 95–145. 2008. (Cited on pages 80, 89, 108, 118, and 135).

[Ove06] Raphael Overbeck. Statistical decoding revisited. In Lynn Batten and Reihaneh Safavi-Naini, editors, *ACISP 2006*, volume 4058 of *Lecture Notes in Computer Science*, pages 283–294. Springer-Verlag Berlin Heidelberg, 2006. ISBN 3540354581. (Cited on page 87).

[P1300] P1363: Standard specifications for public key cryptography, 2000. `http://grouper.ieee.org/groups/1363/`. (Cited on page 32).

[Pat75] Nicholas J. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, IT-21:203–207, 1975. (Cited on pages 76 and 126).

[Pet10] Christiane Peters. Information-set decoding for linear codes over $\mathbf{F}_q$. In Sendrier [Sen10], pages 81–94. ISBN 978-3-642-12928-5. (Cited on pages 4, 83, 100, 111, 115, and 127).

[Pip79] Nicholas Pippenger. The minimum number of edges in graphs with prescribed paths. *Mathematical Systems Theory*, 12:325–346, 1979. ISSN 0025–5661. `http://cr.yp.to/bib/entries.html#1979/pippenger`. (Cited on page 91).

[Pol74] John M. Pollard. Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society*, 76:521–528, 1974. ISSN 0305–0041. (Cited on page 48).

[Pol78] John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978. (Cited on page 162).

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, September 1962. (Cited on pages 3, 85, and 105).

[Rob55] Herbert Robbins. A remark on Stirling's formula. *American Mathematical Monthly*, 62(1):26–29, 1955. ISSN 0002–9890. `http://www.jstor.org/stable/2308012`. (Cited on page 103).

[RR] Fabrice Rouillier and Nathalie Revol. *The MPFI 1.0 library*. `http://perso.ens-lyon.fr/nathalie.revol/mpfi.html`, (accessed version March 26, 2011). (Cited on page 96).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. (Cited on page 2).

[S+10] William A. Stein et al. *Sage Mathematics Software (Version 4.4.3)*. The Sage Development Team, 2010. `http://www.sagemath.org`. (Cited on pages 22, 56, and 126).

[Sch11] Peter Schwabe. *High-Speed Cryptography and Cryptanalysis*. Ph.D. thesis, Eindhoven University of Technology, Netherlands, 2011. (Cited on pages 155 and 170).

[Sen00] Nicolas Sendrier. Finding the permutation between equivalent linear codes: the support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000. (Cited on page 127).

[Sen02] Nicolas Sendrier. On the security of the McEliece public-key cryptosystem. In Blaum et al. [BFvT02], pages 141–163. ISBN 1402070799. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday. (Cited on page 98).

[Sen10] Nicolas Sendrier, editor. *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25–28, 2010. Proceedings*, volume 6061 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2010. ISBN 978-3-642-12928-5. (Cited on pages 175 and 187).

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423,623–656, 1948. `http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html`. (Cited on page 71).

[Sho94] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In Shafi Goldwasser, editor, *35th annual IEEE symposium on the foundations of Computer Science*, pages 124–134. IEEE, 1994. ISBN 0-8186-6580-7. See newer [Sho97]. (Cited on page 2).

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5): 1484–1509, 1997. (Cited on page 188).

[Sil86] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate texts in mathematics*. Springer-Verlag, New York, 1986. ISBN 0387962034. (Cited on pages 9, 50, 51, and 58).

[SKHN76] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. Further results on goppa codes and their applications to constructing efficient binary codes. *IEEE Transactions on Information Theory*, 22(5):518–526, 1976. ISSN 0018–9448. (Cited on pages 121, 122, and 123).

[SRY07] Kannan Srinathan, Chandrasekaran Pandu Rangan, and Moti Yung, editors. *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9–13, 2007, Proceedings*, volume 4859 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2007. ISBN 978–3–540–77025–1. (Cited on pages 174 and 182).

[SS92] Vladimir M. Sidelnikov and Sergey O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2:439–444, 1992. (Cited on pages 79 and 121).

[Ste89] Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding theory and applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer-Verlag Berlin Heidelberg New York, 1989. ISBN 0387516433. (Cited on pages 83, 85, 86, 87, 92, 102, 103, and 106).

[Sti09] Henning Stichtenoth. *Algebraic Function Fields and Codes*, volume 254 of *Graduate texts in mathematics*. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 3642095569. 2nd edition. (Cited on page 9).

[Suy85] Hiromi Suyama. Informal preliminary report (8), 1985. Cited in [Bre86] as personal communication and in [Mon87]. (Cited on page 58).

[SW93] Robert D. Silverman and Samuel S. Wagstaff, Jr. A practical analysis of the elliptic curve factoring algorithm. *Mathematics of Computation*, 61:445–462, 1993. (Cited on page 66).

[Thu73] Edward G. Thurber. On addition chains $l(mn) \leq l(n) - b$ and lower bounds for $c(r)$. *Duke Mathematical Journal*, 40:907–913, 1973. (Cited on page 36).

[VDvT02] Eric R. Verheul, Jeroen M. Doumen, and Henk C. A. van Tilborg. Sloppy Alice attacks! Adaptive chosen ciphertext attacks on the McEliece public-key cryptosystem. In Blaum et al. [BFvT02], pages 99–119. ISBN 1402070799. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday. (Cited on page 80).

[vT90] Johan van Tilburg. On the McEliece public-key cryptosystem. In Shafi Goldwasser, editor, *CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 119–131. Springer-Verlag Berlin Heidelberg, 1990. ISBN 3540971963. (Cited on pages 90, 94, and 110).

[vT94] Johan van Tilburg. *Security-analysis of a class of cryptosystems based on linear error-correcting codes*. Ph.D. thesis, Eindhoven University of Technology, Netherlands, 1994. (Cited on pages 90, 94, and 102).

[Wag02a] David Wagner. A generalized birthday problem (extended abstract). In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag Berlin Heidelberg, 2002. ISBN 354044050X. See also newer version [Wag02b]. (Cited on pages 117, 155, 158, and 190).

[Wag02b] David Wagner. A generalized birthday problem (extended abstract) (long version), 2002. See also older version [Wag02a]. `http://www.cs.berkeley.edu/~daw/papers/genbday.html`. (Cited on pages 127 and 190).

[Wir88] Michael Wirtz. On the parameters of Goppa codes. *IEEE Transactions on Information Theory*, 34(5):1341–1343, 1988. ISSN 0018-9448. (Cited on page 123).

[Z+10] Paul Zimmermann et al. *GMP-ECM (Version 6.3)*. INRIA Project GMP-ECM (Elliptic Curve Method), 2010. `http://ecm.gforge.inria.fr/` (accessed version March 26, 2011). (Cited on pages 45 and 47).

[ZD06] Paul Zimmermann and Bruce Dodson. 20 Years of ECM. In Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors, *ANTS VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 525–542. Springer-Verlag Berlin Heidelberg, 2006. ISBN 3540360751. (Cited on pages 45, 48, and 58).

[Zima] Paul Zimmermann. 50 largest factors found by ECM. `http://www.loria.fr/~zimmerma/records/top50.html` (accessed version March 26, 2011). (Cited on page 45).

[Zimb] Paul Zimmermann. Record factors found by Pollard's $p-1$ method. `http://www.loria.fr/~zimmerma/records/Pminus1.html` (accessed version March 26, 2011). (Cited on page 47).

Index

List of Symbols

Elliptic curves

$E(\mathbb{F}_q)$	rational points on E over $\mathbb{F}_q$, (defined on page 2)
$E(k)$	rational points on E over k, (defined on page 50)
$E(\mathbb{Q})$	rational points on E over $\mathbb{Q}$, (defined on page 47)
$E_{\mathrm{tor}}(\mathbb{Q})$	group of finite order in $E(\mathbb{Q})$, (defined on page 50)
E_d	Edwards curve with coefficient d, (defined on page 12)
$E_{\mathrm{E},a,d}$	twisted Edwards curve with coefficients a and d, (defined on page 13)
$\overline{E}_{\mathrm{E},a,d}$	completed twisted Edwards curve with coefficients a and d, (defined on page 16)
$E^{\mathrm{hom}}_{\mathrm{E},a,d}$	homogenized twisted Edwards equation, (defined on page 15)
$E_{\mathrm{M},A,B}$	Montgomery curve with coefficients A and B, (defined on page 21)
$\mathbb{F}_q$	finite field with q elements, (defined on page 2)
log	natural logarithm (base $e = \exp(1)$), (defined on page 49)
$\log_2$	base-2 logarithm, (defined on page 2)
$[m]$	multiplication-by-m map, (defined on page 2)

Cost of elliptic-curve arithmetic

a	cost of a field addition, (defined on page 25)
D	cost of a multiplication by a small constant factor, (defined on page 25)
I	cost of a field inversion, (defined on page 25)

M	cost of a field multiplication, (defined on page 25)
S	cost of a squaring, (defined on page 25)

Code-based cryptography

$\alpha(R, W)$	$(1-R-W)\log_2(1-R-W) - (1-R)\log_2(1-R) - (1-W)\log_2(1-W)$, (defined on page 102)
$\beta(R, W)$	$\sqrt{(1-R-W)/((1-R)(1-W))}$, (defined on page 102)
$B(P, Q, L)$	asymptotic exponent of the success probability of ball-collision decoding, (defined on page 145)
$\langle C, y \rangle$	linear code spanned by C and y, (defined on page 81)
$C(P, Q, L)$	asymptotic cost exponent of one iteration of ball-collision decoding, (defined on page 145)
$\mathrm{dist}(\cdot, \cdot)$	Hamming distance, (defined on page 72)
$D(P, 0, L)$	asymptotic cost exponent of collision decoding (overall cost), (defined on page 147)
$D(P, Q, L)$	asymptotic cost exponent of ball-collision decoding (overall cost), (defined on page 145)
G_a	unique row of $G_I^{-1}G$ in which the column indexed by $a \in I$ has a 1, (defined on page 84)
G_I	restriction of the generator matrix G to the positions indexed by a set I, (defined on page 73)
$\Gamma_q(a_1, \ldots, a_n, g)$	Goppa code with support elements $a_1, \ldots, a_n$ and Goppa polynomial g, (defined on page 76)
$\Gamma_{q^m}(a_1, \ldots, a_n, g)$	linear code over $\mathbb{F}_{q^m}$ containing $\Gamma_q(a_1, \ldots, a_n, g)$, (defined on page 76)
$\Gamma_q(a_1, \ldots, a_n, g^{q-1})$	wild Goppa code, (defined on page 122)
H_2	binary entropy function, (defined on page 75)
$L(k, p)$	$L(k,p) = \sum_{i=1}^{p} \binom{k}{i}$, (defined on page 92)
$\mathrm{LBCost}(n, k, w, p)$	model of the average time used by the Lee–Brickell algorithm, (defined on page 103)
$\mathrm{LBErr}(n, k, w, p)$	error term; see analyses of $\mathrm{LBPr}(n, k, w, p)$ and $\mathrm{LBCost}(n, k, w, p)$, (defined on page 104)

$\mathrm{LBPr}(n, k, w, p)$	success chance of the first iteration of the Lee–Brickell algorithm, (defined on page 85)
$L_{j,k}$	list k on level j in Wagner's generalized birthday algorithm, (defined on page 159)
$\log$	natural logarithm (base $e = \exp(1)$), (defined on page 109)
$\log_2$	base-2 logarithm, (defined on page 75)
$\log_q$	base-q logarithm, (defined on page 116)
$\mathrm{STCost}(n, k, w, \ell, p)$	model of the average time used by Stern's algorithm, (defined on page 106)
$\mathrm{STPr}(n, k, w, \ell, p)$	success chance of the first iteration of the plain Stern algorithm, (defined on page 94)
$\mathrm{STErr}(n, k, w, \ell, p)$	error term; see analyses of $\mathrm{STPr}(n, k, w, \ell, p)$ and $\mathrm{STCost}(n, k, w, \ell, p)$, (defined on page 107)
Q^t	transpose of a matrix Q, (defined on page 72)
v^t	transpose of a row vector v, (defined on page 72)
$\mathrm{wt}(\cdot)$	Hamming weight, (defined on page 72)
y_I	restriction of y to the positions indexed by a set I, (defined on page 73)

Summary

Curves, Codes, and Cryptography

This thesis deals with two topics: elliptic-curve cryptography and code-based cryptography.

In 2007 *elliptic-curve cryptography* received a boost from the introduction of a new way of representing elliptic curves. Edwards, generalizing an example from Euler and Gauss, presented an addition law for the curves $x^2 + y^2 = c^2(1 + x^2y^2)$ over non-binary fields. Edwards showed that every elliptic curve can be expressed in this form, sometimes at the cost of going to a finite extension of the original field. Bernstein and Lange found fast explicit formulas for addition and doubling in coordinates $(X : Y : Z)$ representing $(x, y) = (X/Z, Y/Z)$ on these curves, and showed that these explicit formulas save time in elliptic-curve cryptography. It is easy to see that all of these curves are isomorphic to curves $x^2 + y^2 = 1 + dx^2y^2$ which now are called "Edwards curves" and whose shape covers considerably more elliptic curves over a finite field than $x^2 + y^2 = c^2(1 + x^2y^2)$.

In this thesis the Edwards addition law is generalized to cover all curves $ax^2 + y^2 = 1 + dx^2y^2$ which now are called "twisted Edwards curves." The fast explicit formulas for addition and doubling presented here are almost as fast in the general case as they are for the special case $a = 1$. This generalization brings the speed of the Edwards addition law to every Montgomery curve.

Tripling formulas for Edwards curves can be used for double-base scalar multiplication where a multiple of a point is computed using a series of additions, doublings, and triplings. The use of double-base chains for elliptic-curve scalar multiplication for elliptic curves in various shapes is investigated in this thesis. It turns out that not only are Edwards curves among the fastest curve shapes, but also that the speed of doublings on Edwards curves renders double bases obsolete for this curve shape.

Elliptic curves in Edwards form and twisted Edwards form can be used to speed up the Elliptic-Curve Method for integer factorization (ECM). We show how to construct elliptic curves in Edwards form and twisted Edwards form with large torsion groups which are used by the EECM-MPFQ implementation of ECM.

Code-based cryptography was invented by McEliece in 1978. The McEliece public-key cryptosystem uses as public key a hidden Goppa code over a finite field. Encryption is remarkably fast (a matrix-vector multiplication). The McEliece cryptosystem

is rarely used in implementations. The main complaint is that the public key is too large. The McEliece cryptosystem recently regained attention with the advent of *post-quantum cryptography*, a new field in cryptography which deals with public-key systems without (known) vulnerabilities to attacks by quantum computers. McEliece's system is one of them. In this thesis we further investigate the McEliece cryptosystem by improving attacks against it and by coming up with smaller-key variants.

McEliece proposed to use binary Goppa codes. For these codes the most effective attacks rely on information-set decoding. In this thesis we present an attack developed together with Daniel J. Bernstein and Tanja Lange which uses and improves Stern's idea of collision decoding. This attack is faster by a factor of more than 150 than previous attacks, bringing it within reach of a moderate computer cluster. In fact, we were able to extract a plaintext from a ciphertext by decoding 50 errors in a $[1024, 524]$ binary code. The attack should not be interpreted as destroying the McEliece cryptosystem. However, the attack demonstrates that the original parameters were chosen too small. Building on this work the collision-decoding algorithm is generalized in two directions. First, we generalize the improved collision-decoding algorithm for codes over arbitrary fields and give a precise analysis of the running time. We use the analysis to propose parameters for the McEliece cryptosystem with Goppa codes over fields such as $\mathbb{F}_{31}$. Second, collision decoding is generalized to ball-collision decoding in the case of binary linear codes. Ball-collision decoding is asymptotically faster than any previous attack against the McEliece cryptosystem.

Another way to strengthen the system is to use codes with a larger error-correction capability. This thesis presents "wild Goppa codes" which contain the classical binary Goppa codes as a special case. We explain how to encrypt and decrypt messages in the McEliece cryptosystem when using wild Goppa codes. The size of the public key can be reduced by using wild Goppa codes over finite fields of moderate size. We evaluate the security of the "Wild McEliece" cryptosystem against our generalized collision attack for codes over finite fields.

Code-based cryptography deals not only with public-key cryptography: a code-based hash function "FSB" was submitted to NIST's SHA-3 competition, a competition to establish a new standard for cryptographic hashing. Wagner's generalized birthday attack is a generic attack which can be used to find collisions in the compression function of FSB. However, applying Wagner's algorithm is a challenge in storage-restricted environments. The FSBday project showed how to successfully mount the generalized birthday attack on 8 nodes of the Coding and Cryptography Computer Cluster (CCCC) at Technische Universiteit Eindhoven to find collisions in the toy version FSB_{48} which is contained in the submission to NIST.

Curriculum Vitae

Christiane Peters was born on February 15, 1981 in Paderborn, Germany.

In 2000 she obtained her Abitur at the secondary school Pelizaeus Gymnasium in Paderborn. After completing the first semester in mathematical finance at the University of Bielefeld she returned to Paderborn where she studied mathematics at the University of Paderborn. Christiane received her diploma in mathematics "mit Auszeichnung" (with distinction) in December 2006. During her studies she focused on algebraic geometry, complex analysis, and in particular on elliptic curves in algorithmic number theory. Her Diploma thesis titled *Bestimmung des Elkies-Faktors im Schoof-Elkies-Atkin-Algorithmus* was supervised by Prof. Dr. Peter Bürgisser and Prof. Dr. Preda Mihăilescu.

In June 2007 Christiane started her Ph.D. studies on code-based cryptography in the Coding Theory and Cryptology group at Technische Universiteit Eindhoven under the supervision of Prof. Dr. Tanja Lange and Prof. Dr. Daniel J. Bernstein. She continued to work on elliptic curves, changing the focus towards cryptographic applications of elliptic curves in Edwards form. At the same time she started working on code-based cryptography. In her second year together with Daniel J. Bernstein and Tanja Lange she successfully broke the original McEliece parameters. This work made headlines in the Netherlands and internationally. In 2009 she was one of the recipients of the Anita Borg Memorial Scholarship awarded by Google.

The present dissertation contains the results of her work from 2007 to 2011.