QUANTUM CIRCUITS FOR CRYPTANALYSIS

by

Brittanney Jaclyn Amento

A Dissertation Submitted to the Faculty of

The Charles E. Schmidt College of Science

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Florida Atlantic University

Boca Raton, FL

August 2016

QUANTUM CIRCUITS FOR CRYPTANALYSIS

by

Brittanney Jaclyn Amento

This dissertation was prepared under the direction of the candidate's dissertation advisor, Dr. Rainer Steinwandt, Department of Mathematical Sciences, and has been approved by the members of her supervisory committee. It was submitted to the faculty of the Charles E. Schmidt College of Science and was accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

SUPERVISORY COMMITTEE:

Rainer Steinwandt, Ph.D.
Dissertation Advisor

Roger Goldwyn, Ph.D.

Frederick Hoffman, Ph.D.

Koray Karabina, Ph.D.

Lee Klingler, Ph.D.

Rainer Steinwandt, Ph.D.
Chair, Department of Mathematical Sciences

Janet Blanks, Ph.D.
Interim Dean, The Charles E. Schmidt College of Science

Deborah L. Floyd, Ed.D.
Dean, Graduate College

07/20/2016
Date

ACKNOWLEDGEMENTS

I would like to first thank God for all the blessings He has bestowed upon me... without His strength and guidance, I would not be where I am today.

I would like to thank my advisor, Dr. Rainer Steinwandt. I do not believe there are even words I could find to explain my gratitude, but I will try. I will be forever grateful for your unwaivering support and your ability to see my capabilities even when I struggled to see them, for you pushing me to never give up on my dreams of completing this dissertation–even when it seemed a lost cause, and for your willingness and calm demeanor in handling my colorful personality. You are absolutely brilliant and I hope I have made you proud. I would like to thank Dr. Roger Goldwyn for being such a fantastic boss and giving me so many opportunities to shine. I can not begin to tell you how much I have learned and continue to learn from you every day. You are truly a blessing in my life! I would like to thank Dr. Frederick Hoffman for being the father figure I never had. I cherish our countless conversations and the wisdom and confidence you have always shared with me. I would like to thank Dr. Lee Klingler for being a wonderful mentor. I can not tell you how much I enjoy our coffee conversations about school, work, and life in general. Your support and advice has been essential. I would like to thank Dr. Koray Karabina for being a part of my committee and offering your expertise. I would like to thank Dr. Vincent Naudot... you know you are my angel on Earth! Thank you for never allowing me to give up, for your continued faith in me, and for always being available to answer analysis questions during my qualifier preparation.

I would like to thank my wonderful fiance, Chad Adelmann, for putting up with me during this endeavor. For everything from making me dinner to listening

to me vent to being supportive and understanding when I had to put school first... you are truly my best friend and I cannot wait to be your wife. I want to thank Kathryn Delguidice for without you showing me the way, I would surely still be lost. I would like to thank Dr. Drake Harmon for the countless hours you spent working on analysis problems with me... I could NOT have done it without you!

Most importantly, I would like to thank my Mom. You show me everyday what real love is. You are my biggest fan, and I am yours. I love you to the moon and back.

ABSTRACT

Author:	Brittanney Jaclyn Amento
Title:	Quantum Circuits for Cryptanalysis
Institution:	Florida Atlantic University
Dissertation Advisor:	Dr. Rainer Steinwandt
Degree:	Doctor of Philosophy
Year:	2016

Finite fields of the form $\mathbb{F}_{2^m}$ play an important role in coding theory and cryptography. We show that the choice of how to represent the elements of these fields can have a significant impact on the resource requirements for quantum arithmetic. In particular, we show how the Gaussian normal basis representations and "ghost-bit basis" representations can be used to implement inverters with a quantum circuit of depth $\mathrm{O}(m \log(m))$. To the best of our knowledge, this is the first construction with subquadratic depth reported in the literature. Our quantum circuit for the computation of multiplicative inverses is based on the Itoh-Tsujii algorithm which exploits the property that, in a normal basis representation, squaring corresponds to a permutation of the coefficients. We give resource estimates for the resulting quantum circuit for inversion over binary fields $\mathbb{F}_{2^m}$ based on an elementary gate set that is useful for fault-tolerant implementation.

Elliptic curves over finite fields $\mathbb{F}_{2^m}$ play a prominent role in modern cryptography. Published quantum algorithms dealing with such curves build on a short Weierstrass form in combination with affine or projective coordinates. In this thesis we show that changing the curve representation allows a substantial reduction in the

number of T-gates needed to implement the curve arithmetic. As a tool, we present a quantum circuit for computing multiplicative inverses in $\mathbb{F}_{2^m}$ in depth $\mathrm{O}(m \log m)$ using a polynomial basis representation, which may be of independent interest.

Finally, we change our focus from the design of circuits which aim at attacking computational assumptions on asymmetric cryptographic algorithms to the design of a circuit attacking a symmetric cryptographic algorithm. We consider a block cipher, SERPENT, and our design of a quantum circuit implementing this cipher to be used for a key attack using Grover's algorithm as in [18]. This quantum circuit is essential for understanding the complexity of Grover's algorithm.

To my Mom, Neweleen Feldmar, for her unconditional love and strength all the days of my life.

QUANTUM CIRCUITS FOR CRYPTANALYSIS

List of Figures xi

1 Introduction 1

1.1 Quantum binary field inversion 2

1.2 Quantum binary elliptic curve arithmetic 3

1.3 Serpent 4

2 Preliminaries 5

2.1 Quantum computing 5

2.1.1 Quantum terminology 5

2.2 Finite fields $\mathbb{F}_{2^m}$ 11

2.2.1 Polynomial basis representation 11

2.2.2 Ghost-bit basis representation 12

2.2.3 Normal basis representation 14

2.2.4 Computing multiplicative inverses with the Itoh-Tsujii algorithm 17

2.3 Binary elliptic curves 18

2.3.1 Short Weierstrass form 18

2.3.2 Group law 19

2.3.3 Complete binary Edwards curves 19

3 Quantum Circuits for Finite Field Arithmetic 21

3.1 Multiplying in linear depth using ghost-bit and Gaussian normal basis representations 21

3.1.1 Linear depth multiplication using a ghost-bit basis 21

3.1.2 Linear depth multiplication using a Gaussian normal basis . . . 26

3.2 Inversion in depth $O(m \log(m))$ using the Itoh-Tsujii algorithm . . . 31

4 Quantum Circuits for Binary Elliptic Curve Arithmetic . . . 36

4.1 Fixing a finite field representation . . . 36

4.1.1 Polynomial basis representation . . . 37

4.1.2 Gaussian normal basis and ghost-bit basis representations . . . 38

4.1.3 Itoh-Tsujii inversion with a polynomial basis representation . . . 40

4.2 Choosing a curve representation: the cost of adding a fixed point . . . 42

4.2.1 Mixed addition with projective coordinates . . . 43

4.2.2 Mixed addition with a formula by Higuchi and Takagi . . . 45

4.3 Implementing a general point addition using Edwards curves . . . 48

5 A Quantum Circuit for SERPENT . . . 53

5.1 The key schedule . . . 53

5.2 The S-boxes . . . 55

5.2.1 Permutation parity . . . 57

5.3 The linear transformation . . . 58

5.3.1 The rotation operation . . . 58

5.3.2 The shift operation . . . 59

5.4 The SERPENT quantum circuit . . . 60

6 Comparisons and conclusions . . . 64

Bibliography . . . 67

LIST OF FIGURES

2.1 A NOT gate and corresponding truth table 8

2.2 A CONTROLLED NOT (XOR) gate and corresponding truth table 8

2.3 Toffoli gates with corresponding truth tables 9

2.4 Common Clifford gates 10

2.5 Adding two points on the elliptic curve $y^2 + xy = x^3 + a_2x^2 + a_6$ 20

3.1 A ghost-bit basis multiplier for $\alpha \cdot \beta \in \mathbb{F}_{2^4}$ 22

3.2 Graph representing the term for $\sigma = 0$ as described in Example 3.1.2. The 2-coloring of the edges—where the different line styles indicate the colors—translates into a depth 2 circuit for this term. 25

3.3 A ghost-bit basis multiplier for $\alpha \cdot \alpha^{2^2} \in \mathbb{F}_{2^4}$ 26

3.4 A Gaussian normal basis multiplier for $\alpha \cdot \beta \in \mathbb{F}_{2^5}$ 28

3.5 Graph corresponding to the cosets $\delta + (5)$ for $\delta = -1$ as described in Example 3.1.4. The 3-coloring of the edges—where the different line styles in the pentagon indicate the three different colors—translates into a depth 3 circuit. 31

3.6 Part of a Gaussian normal basis multiplier for $\alpha \cdot \alpha^{2^1} \in \mathbb{F}_{2^5}$: computing the terms $\alpha_{4+i}\alpha_{3+i}$ 31

3.7 A ghost-bit or Gaussian normal basis inverter for $\alpha \in \mathbb{F}_{2^7}^*$ 34

5.1 Using S-boxes to transform intermediate (pre)keys into round keys 54

5.2 S-boxes $S_0, ..., S_7$ and their inverses 56

5.3 Mixing four 32-bit words with a linear transformation 59

5.4 S-box parity for 32 rounds of SERPENT. With O = Odd and E = Even, from left to right, we list the S-box associated with K_i, then the S-box associated with $B_i \oplus K_i$, and finally the inverse of the S-box associated with K_i. 63

CHAPTER 1

INTRODUCTION

In quantum computing, arithmetic operations occur in a plurality of contexts [6, 12, 14, 21, 27, 34, 42]. Having good quantum circuits for arithmetic is indispensable for obtaining good resource estimates and efficient circuit implementations of more complex quantum algorithms.

We begin by focusing on quantum circuits to implement arithmetic in finite fields of the form $\mathbb{F}_{2^m}$, based on [2]. In view of the cryptographic significance, it is not surprising that quantum circuits to implement finite field arithmetic have already been explored in a number of publications, including [7, 26, 30, 31]. Important special cases are arithmetic operations in finite prime fields and finite binary fields (cf., for instance, [35]). While there is some common ground between the prime-field case and the characteristic-two case, there are also important differences.

Next, we turn our focus to binary elliptic curves, an especially important family of groups for cryptographic applications. The implementation of their addition law in a quantum circuit has been studied by a number of authors [26, 30]. We show how changing the curve representation can help to reduce the number of T-gates needed to implement elliptic curve arithmetic—and in addition help to reduce the T-depth, based on [1].

Finally, we present design and resource estimates for a quantum circuit which implements SERPENT, one of the finalists for the Advanced Encryption Standard (AES). We are interested in utilizing this circuit for encrypting the plaintext, in a given plaintext/ciphertext pair, during a quest to find the true 256-bit encryption key in a given key space as part of Grover's algorithm [18].

1.1 QUANTUM BINARY FIELD INVERSION

Interestingly, thus far the literature on quantum circuits for $\mathbb{F}_{2^m}$-arithmetic focuses completely on polynomial basis representations, and computing multiplicative inverses by implementing the extended Euclidean algorithm as discussed in [26] appears to be the common choice. The cost of implementing inversion this way is significant as the resulting circuit has a size that is cubic in m. When realizing the group law on a binary elliptic curve as a quantum circuit, the cost of this operation becomes apparent: Maslov et al. presented a solution to the discrete logarithm problem on binary elliptic curves [30]. An important technique for achieving quadratic depth with their solution was to bring down the number of finite field inversions to one. For the asymptotic analysis, the quadratic depth of this single inversion is still as expensive as all other arithmetic operations combined. So, when trying to improve on the discrete logarithm circuit presented in [30]—which from a cryptanalytic point of view is desirable—reducing the complexity of binary finite field inversion is a natural first step.

In Chapter 3, we present linear-depth multipliers using a so-called ghost-bit basis and using Gaussian normal bases. Building on these multipliers, we describe an inverter for $\mathbb{F}_{2^m}^*$ of depth $\mathrm{O}(m \log(m))$ derived from a classical inversion algorithm by Itoh and Tsujii [23], using $\mathrm{O}(m \log(m))$ qubits. We hope that our work stimulates follow-up work on using different representations of finite fields in quantum circuits, and we expect that the circuits presented in this thesis will be useful not only for speeding up the arithmetic for quantum algorithms for computing discrete logarithms on elliptic curves, and also for other algebraic problems that can be tackled on a quantum computer, including hidden polynomial equations [12], hidden shift problems [14, 37, 41], and certain period finding tasks [21, 27, 42].

The elementary gate set consisting of all Clifford gates and the so-called T-gate is a preferable one, see Definition 2.1.6, for the fault-tolerant implementation of quantum

circuits on several error-correcting codes [15, 38]. The actual complexity of a fault-tolerant implementation of T-gates is extremely high; hence, it is preferable to reduce their number as much as possible. We show that in a Gaussian normal basis or a ghost-bit basis representation, an inversion over $\mathbb{F}_{2^m}$ can be computed in a T-depth of $\mathrm{O}(m \log(m))$ and using at most $\mathrm{O}(m^2 \log(m))$ many T-gates.

1.2 QUANTUM BINARY ELLIPTIC CURVE ARITHMETIC

To the best of our knowledge, all discussions prior to this work on implementing the addition law for elliptic curves in a quantum circuit use a short Weierstrass representation in combination with affine or projective coordinates. While this is a natural choice, restricting to such representations does not fully exploit the available technical machinery—there is a substantial body of work on how to optimize elliptic curve arithmetic on classical hardware architectures (cf. [8]), and one may hope that some of these classical results allow for simplification at the circuit level when implementing binary elliptic curve arithmetic in a quantum circuit, e. g., when trying to find discrete logarithms [42]. For an actual implementation, the number of T-gates needed to implement such a circuit is particularly of interest and it is desirable to keep this number as small as possible. While minimizing the total number of T-gates is the prime objective of circuit synthesis at the logical level, the total depth, see Definition 2.1.7, of the computation when arranged as an alternation between T-gates and Clifford gates (the so-called "T-depth") is also an important parameter. It is desirable to keep the T-depth low by parallelizing T-gates as much as possible.

In Chapter 4, we present our quantum circuit for (generic) addition of a fixed curve point which makes use of point addition formulae suggested by Higuchi and Takagi [22] and can in particular be used to reduce the number of T-gates as well as the T-depth, in comparison to the use of ordinary projective coordinates (cf. [30]). Section 4.3 presents a quantum circuit for general point addition on a complete binary

Edwards curve.

Some applications of elliptic curves may require unique representations of curve points (cf. [30]). When dealing with representations for fast arithmetic, deriving a unique point representation may involve an inversion in the underlying finite field. In a polynomial basis representation, a quantum implementation of the extended Euclidean algorithm can be used for this inversion; however, the circuit has $\mathrm{O}(m^3)$ gates and quadratic depth [26,30,31]. For other field representations, an inversion algorithm with depth $\mathrm{O}(m \log m)$ and $\mathrm{O}(m^2 \log m)$ gates is proposed in Section 3.2. In order to compute unique point representations using a polynomial basis more efficiently, we adapt the approach from Chapter 3 to the polynomial basis setting. In this way we obtain the first published quantum circuit using a polynomial basis representation which can compute inverses in $\mathbb{F}_{2^m}^*$ in depth $\mathrm{O}(m \log m)$ with $\mathrm{O}(m^2 \log m)$ gates.

1.3 SERPENT

In Chapter 5, we present our quantum circuit for implementing SERPENT. As we are interested in understanding the complexity of Grover's algorithm, we show this circuit can be realized using 800 qubits, 45,642 CNOT gates and 6,192 NOT gates for linear computations and a maximum of a combination of 172,480 gates and 1,600 NOT gates for non-linear computations.

CHAPTER 2

PRELIMINARIES

2.1 QUANTUM COMPUTING

The security of a significant portion of public-key cryptography and cryptographic primitives relies on the computational hardness of solving some problems, e. g., factoring integers or computing discrete logarithms. Harnessing the power of quantum computation suggests a reduction in the complexity of solving a number of these hard problems from exponential to polynomial time. We should also note a further strengthening of security, due to the nature of quantum systems, is that undetected eavesdropping on a quantum channel is impossible. Let us also mention a few drawbacks of quantum computing as mentioned in [19]. Utilizing the power of the parallelism of a quantum system can pose enormous problems. Measurement is destructive; when we decide to take a measurement, we get only one classical result, randomly chosen, and the remaining quantum information can be irreversibly destroyed. Further, just the interaction between a quantum system and its environment can lead to the leakage and loss of information from the quantum system into the environment. In the following, we review some basic facts about the components of quantum computing necessary for reading this thesis. All of these components are known, and we claim no originality for this section.

2.1.1 Quantum terminology

We begin by introducing a Hilbert space, as it provides a mathematical framework suitable for the study of quantum algorithms and processes. The underlying concept

of a Hilbert space is that of a vector space. We call the elements of a Hilbert space, vectors, and if they have norm 1, we refer to them as (pure) states. It is important to keep in mind that this space allows us to describe theoretically how things should behave in a quantum system but the design and actual application occur in a laboratory.

Definition 2.1.1 (Hilbert space)**.** *An inner-product space $\mathcal{H}$ is called complete, if for any sequence $\{\phi_i\}_{i=1}^{\infty}$, with $\phi_i \in \mathcal{H}$, and with the property that $\lim_{i,j\to\infty} \parallel \phi_i - \phi_j \parallel = 0$, there is a unique element $\phi \in \mathcal{H}$ such that $\lim_{i\to\infty} \parallel \phi - \phi_i \parallel = 0$. A complete inner-product space is called a Hilbert space. A vector ψ of a Hilbert space $\mathcal{H}$ is denoted $|\psi\rangle$ and referred to as a ket-vector.*

Example 2.1.1. *Some examples of finite dimensional Hilbert spaces:*

- *The real n-space $\mathbb{R}^n$ with inner-product the vector dot product.*
- *The complex n-space $\mathbb{C}^n$ with inner-product the vector dot product of an element with its complex conjugate.*

An example of an infinite Hilbert space is $\mathcal{L}_2$:

- *Let (a, b) be an interval, with finite or infinite bounds, on the real axis. By $\mathcal{L}_2$, we denote the set of all complex valued functions such that $\int_a^b \mid f(x) \mid^2 dx$ exists, equipped with the inner-product $\langle f \mid g \rangle = \int_a^b f^*(t)g(t)dt < \infty$.*

Definition 2.1.2 (Quantum state)**.** *A quantum state is a complete description of a quantum system. A state is a ray (an equivalence class of vectors) in the corresponding Hilbert space over the complex numbers. We often consider vectors of norm 1 as the representatives for such equivalence classes. A quantum state can be either pure or mixed (see [19] for more information).*

We consider a qubit as a unit vector in a two-dimensional inner-product space and assume the qubit representation is that of a particular fixed basis $\{|0\rangle, |1\rangle\}$.

Definition 2.1.3 (Qubit)**.** *A qubit (quantum bit) is a quantum state*

$$|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where $\alpha, \beta \in \mathbb{C}$ *and* $|\ \alpha\ |^2 + |\ \beta\ |^2 = 1$*. Here,* α, β *are called amplitudes (probabilities of being in one of the two basis states).*

It is important to recognize that a large part of the computational power harnessed by quantum computing lies in the fact that quantum carriers exist in all possible states simultaneously, called superposition, but exhibit only one state when measured. Hence, classical computer bits are either a 0 or a 1, whereas quantum qubits are both a 0 and 1 at the same time.

Definition 2.1.4 (Basis states)**.** *The set* $\mathcal{B} = \{i \mid i \text{ denotes a state}\}$ *is a set of basis states, if for all* $i, j \in \mathcal{B}$*,*

$$\langle i, j\rangle = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ otherwise} \end{cases} \tag{2.1}$$

Definition 2.1.5 (Quantum gates)**.** *A quantum gate with* n *inputs and* n *outputs is specified by a unitary operator* $\mathcal{U} : \mathcal{H}_{2^n} \to \mathcal{H}_{2^n}$*, and it is represented by a unitary matrix of degree* 2^n*. For more information on unitary matrices and unitary operators, see [19].*

Quantum gates must be reversible or information will be irreversibly lost during gate operation. Reversible gates contain appropriate information to determine from the output exactly the input. During computation, garbage (previous non-needed computations) can build up. We can reverse these computations (uncompute them) to clear some qubits (wires) and re-use them for further computation. It is also predicted that reversible gates will be the only way to improve energy efficiency as cooling of dissipated energy generated by irreversible operations will not be an option (it would destroy the superposition of states).

Definition 2.1.6 (Elementary gate set)**.** *The following gates are commonly used in this thesis:*

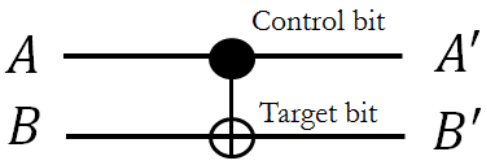
- *NOT gate (N): This gate basically flips a bit*

Figure 2.1: A NOT gate and corresponding truth table

A	A'
0	1
1	0

- *CONTROLLED NOT gate (CNOT or XOR): If the control bit is a* 1*, xor the target bit with* 1

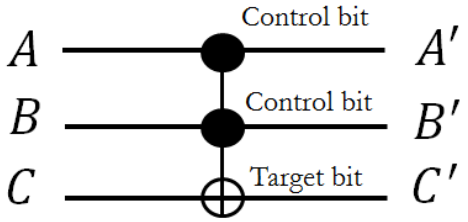
Figure 2.2: A CONTROLLED NOT (XOR) gate and corresponding truth table



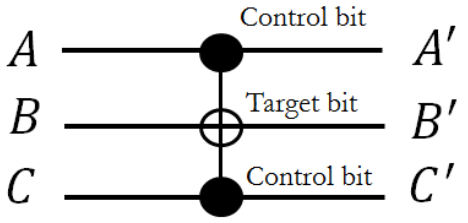
A	B	A'	B'
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

- *TOFFOLI gate (CCN): If both control bits are* 1*, xor the target bit with* 1

Figure 2.3: Toffoli gates with corresponding truth tables



A	B	C	A'	B'	C'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



A	B	C	A'	B'	C'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	1	0	1

The construction of a Toffoli gate requires Clifford gates and T-gates. A single Toffoli gate requires the use of 7 *T-gates (or* $T^\dagger$*-gates which we assume to have the same cost). It is desirable to keep the number of T-gates in a circuit to a minimum. The reason for this is that for most fault-tolerant quantum computing schemes, the implementation of T-gates is achieved via so-called magic state distillation [10, 15, 38], a process which is costly in terms of physical resources required. For instance, in the case of the surface code, it is reasonable to assume that a single T-gate has a cost that is about* 100 *times higher than a single CNOT [15].*

- *T-gate: the local unitary* $T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
- *Clifford gates: as seen in Figure 2.4, this group on n qubits is defined as the normalizer of* $\mathcal{P}_n$ *in* $\mathcal{U}(2^n)$ *where* $\mathcal{P}_n$ *is the set consisting of single qubit Pauli Matrices [3] and* $\mathcal{U}(2^n)$ *is the unitary group of degree* 2^n.

Figure 2.4: Common Clifford gates

$$\text{Hadamard gate } H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \text{ Phase gate } P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

$$\text{CNOT gate} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$\text{Pauli gates } I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Definition 2.1.7 (Depth)**.** *The depth of a circuit, as in [3], is defined to be the length of any critical path through the ciruit. A critical path is a path of maximum length from the input of the circuit to an output. We represent the circuit as a directed acyclic graph in which nodes correspond to gates and edges correspond to gate inputs/outputs.*

Definition 2.1.8 (Quantum boolean circuit)**.** *A quantum Boolean circuit is a collection of quantum gates acyclicly connected (by "quantum wires"). The size and the depth of a circuit refer to the number of nodes and depth of the underlying connection graph.*

2.2 FINITE FIELDS $\mathbb{F}_{2^m}$

Perhaps the most popular representation of finite fields $\mathbb{F}_{2^m}$ is the use of a polynomial basis. In the following, we briefly review some basic facts about this representation as well as two alternatives—the use of a ghost-bit basis and of a Gaussian normal basis. All of these representations are known, and we claim no originality for this section.

2.2.1 Polynomial basis representation

Denoting by $f = x^m + \sum_{i=0}^{m-1} x^i \in \mathbb{F}_2[x]$ an irreducible polynomial of degree m over the prime field $\mathbb{F}_2$, we can identify $\mathbb{F}_{2^m}$ with the quotient ring $\mathbb{F}_2[x]/(f)$, and this identification forms a popular representation of binary finite fields.

Definition 2.2.1 (Polynomial basis representation)**.**
With the above notation, let $x^0+(f), x^1+(f), \ldots, x^{m-1}+(f)$ *be the canonical* $\mathbb{F}_2$*-vector space basis of* $\mathbb{F}_2[x]/(f)$*. In the* polynomial basis representation*, each* $\alpha \in \mathbb{F}_{2^m}$ *is represented by the unique tuple* $(\alpha_0, \ldots, \alpha_{m-1}) \in \mathbb{F}_2^m$ *such that* $\alpha = \sum_{i=0}^{m-1} \alpha_i \cdot (x^i+(f))$*.*

Example 2.2.1. *The polynomial* $x^4 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x]$ *is irreducible, and so the field with* 16 *elements can be identified with* $\mathbb{F}_2[x]/(x^4+x^3+x^2+x+1)$*. Choosing* $f = x^4+x^3+x^2+x+1$ *in the above definition, in the polynomial basis representation, the tuple* $(1, 0, 1, 0) \in \mathbb{F}_2^4$ *represents the field element* $x^2 + 1 + (f)$*.*

In the current literature on quantum arithmetic for binary finite fields, the representation from Definition 2.2.1 seems to be the only one considered. Beauregard et al. [7], Maslov et al. [31], and Kaye and Zalka [26] provide circuits for addition, multiplication and inversion using a polynomial basis.

- Using one qubit per coefficient of $\alpha = \sum_{i=0}^{m-1} \alpha_i \cdot (x^i + (f))$, adding $|\alpha\rangle$ to an m-qubit input $|\beta\rangle$ can be done in the obvious way with m CNOT gates, each conditioned on one of the α_i. These CNOT gates operate on disjoint wires, and hence this adder can be realized in depth 1.

- Building on a classical Mastrovito multiplier [32, 33, 39], the multiplication of two m-qubit inputs $|\alpha\rangle$ and $|\beta\rangle$ can be realized in depth $9m+\mathrm{O}(1)$ using Toffoli gates. If the irreducible polynomial f is the all-one polynomial or a trinomial, m^2-m-1 gates suffice [31].

- Computing the inverse of a non-zero $\alpha \in \mathbb{F}_{2^m}$, using the extended Euclidean algorithm, can be implemented in depth $\mathrm{O}(m^2)$ and $2m+\mathrm{O}(\log(m))$ qubits [26, 30].

In this thesis, we will look at two different representations of binary fields which—from an algorithmic point of view—suggest an interesting alternative to the use of a polynomial basis.

2.2.2 Ghost-bit basis representation

Keeping the notation from above, suppose the irreducible polynomial f we use is the all-ones polynomial $x^m+\cdots+1$. In this case, $m+1$ is prime and 2 is a generator of the cyclic group $\mathbb{Z}^*_{m+1}$ (cf. [23]). Then f divides $x^{m+1}+1=(x+1)\cdot(x^m+\cdots+1) \in \mathbb{F}_2[x]$, and we can define the map

$$\begin{array}{rccc} \phi: & \mathbb{F}_2[x]/(f) & \longrightarrow & \mathbb{F}_2[x]/(x^{m+1}+1) \\ & \sum_{i=0}^{m-1}\alpha_i\cdot x^i+(f) & \longmapsto & \sum_{i=0}^{m-1}\alpha_i\cdot x^i+(x^{m+1}+1) \end{array}.$$

The map ϕ may be seen as appending an extra (zero) bit to the coefficient vector of a polynomial basis representation of $\alpha \in \mathbb{F}_2[x]/(f)$. As detailed by Silverman [43] (who suggests attributing the construction to Itoh and Tsujii [23]), instead of adding, multiplying, and inverting elements in $\mathbb{F}_2[x]/(f)$ directly, we can apply ϕ to the operands, perform the needed additions, multiplications, and inversions in $\mathbb{F}_2[x]/(x^{m+1}+1)$, and then map the result back into $\mathbb{F}_2[x]/(f)$ by applying

$$\begin{array}{rccc} & \mathbb{F}_2[x]/(x^{m+1}+1) & \longrightarrow & \mathbb{F}_2[x]/(f) \\ & \sum_{i=0}^{m}\alpha_i\cdot x^i+(x^{m+1}+1) & \longmapsto & \sum_{i=0}^{m-1}(\alpha_i+\alpha_m)\cdot x^i+(f) \end{array}. \tag{2.2}$$

Definition 2.2.2 (Ghost-bit basis representation)**.**

With the above notation, assume that $1 + \cdots + x^m$ *is irreducible. In the* ghost-bit basis representation*, each* α *is represented by a tuple* $(\alpha_0, \ldots, \alpha_m) \in \mathbb{F}_2^{m+1}$ *such that* $(\alpha_0 + \alpha_m, \ldots, \alpha_{m-1} + \alpha_m)$ *is the polynomial basis representation of* α *using the irreducible polynomial* $1 + \cdots + x^m$*.*

Thus, a conversion from the *ghost-bit basis representation* to a polynomial basis representation boils down to dropping the ghost bit and adding (XOR) it to the remaining m bits. In a quantum circuit, this translates into a single CNOT with multiple fan-out at the very end, provided we do not have to restore the initial $|0\rangle$-value of the ghost (qu)bit. We note that for adding field elements alone, applying the map ϕ has no advantage—but also no dramatic drawback.

- Using one qubit per coefficient of $\alpha = \sum_{i=0}^{m} \alpha_i \cdot x^i + (x^{m+1} + 1)$, adding $|\alpha\rangle$ to an $(m+1)$-qubit input $|\beta\rangle$ can be done in the obvious way with $m+1$ CNOT gates, conditioned on the individual α_i. These CNOT gates operate on disjoint wires, and hence this adder can be realized in depth 1.

To realize quantum circuits for multiplying and inverting field elements, we are interested in exploiting the following properties of $\mathbb{F}_2[x]/(x^{m+1} + 1)$:

- Squaring corresponds to a shuffle of the coefficient vector:

$$\left(\sum_{i=0}^{m} \alpha_i \cdot x^i + (x^{m+1} + 1) \right)^2 = \sum_{i=0}^{m} \alpha_{\pi^{-1}(i)} \cdot x^i + (x^{m+1} + 1), \tag{2.3}$$

  where $\pi(i) = 2 \cdot i \bmod (m+1)$ for $i = 0, \ldots, m$.

  Example 2.2.2. *As noted in Example 2.2.1, the polynomial* $x^4+x^3+x^2+x+1 \in \mathbb{F}_2[x]$ *is irreducible, and so* $\mathbb{F}_{2^4}$ *affords a ghost-bit basis representation: the above map* ϕ *translates operations in* $\mathbb{F}_{2^4}$ *into operations in* $\mathbb{F}_2[x]/(x^5+1)$*. Applying* ϕ *to* $x^2+1+(x^4+x^3+x^2+x+1)$*, we obtain* $x^2+1+(x^5+1)$*; i. e., the polynomial*

basis representation $(1,0,1,0)$ *from Example 2.2.1 translates into the ghost-bit basis representation* $(1,0,1,0,0)$.

For $m = 4$*, the permutation* π *in Equation* (2.3) *is* $(0)(1,2,4,3)$*, so the ghost-bit basis representation of* $(x^2+1+(x^5+1))^2$ *is* $(1,0,0,0,1)$*—corresponding to* $x^4+1+(x^5+1)$*. Applying the map from Equation* (2.2)*, we obtain the polynomial basis representation* $(0,1,1,1)$*—corresponding to* $x^3+x^2+x+(x^4+x^3+x^2+x+1)$.

- To multiply two elements $\alpha = \sum_{i=0}^{m} \alpha_i \cdot x^i + (x^{m+1}+1)$ and $\beta = \sum_{i=0}^{m} \beta_i \cdot x^i + (x^{m+1}+1)$, the following formula for the coefficients of their product $\gamma = \sum_{i=0}^{m} \gamma_i \cdot x^i + (x^{m+1}+1)$ can be used:

$$\gamma_i = \sum_{j=0}^{m} \alpha_j \beta_{(i-j) \bmod (m+1)} \tag{2.4}$$

As explained in Section 3.1.1 below, in combination with an observation in [31], Equation (2.4) yields a linear-depth circuit for multiplication in $\mathbb{F}_2[x]/(x^{m+1}+1)$.

Remark 2.2.1. *The idea of a ghost-bit basis can be generalized to a representation with more redundancy—whenever the polynomial* $x^n + 1 \in \mathbb{F}_2[x]$ *has an irreducible factor* f *of degree* m*, then we can define a map* ϕ *analogously as above, using* $n - m$ *"ghost bits." Geiselmann and Lukhaub [16] discuss the implementation of* $\mathbb{F}_{2^m}$*-multiplication in such a representation with a classical reversible circuit.*

2.2.3 Normal basis representation

The possibility of an inexpensive squaring operation will be of great benefit for the inversion algorithm below, and a natural type of field representation to be considered in this context is the *normal basis representation*.

Definition 2.2.3 (Normal basis representation)**.**
Let $\eta \in \mathbb{F}_{2^m}$ *be such that* $\{\eta, \eta^2, \eta^{2^2}, \ldots, \eta^{2^{m-1}}\}$ *is an* $\mathbb{F}_2$*-vector space basis of* $\mathbb{F}_{2^m}$*. In*

a normal basis representation *of* $\mathbb{F}_{2^m}$*, we represent each* $\alpha \in \mathbb{F}_{2^m}$ *by the unique tuple* $(\alpha_0, \alpha_1, \cdots, \alpha_{m-1}) \in \mathbb{F}_2^m$ *with* $\alpha = \sum_{i=0}^{m-1} \alpha_i \cdot (\eta^{2^i})$.

A normal basis representation exists for every field $\mathbb{F}_{2^m}$ of degree $m \geq 1$, and more background information on normal bases can be found, for instance, in [25]. By construction, squaring in such a representation is just a cyclic shift, and addition can be implemented as bit-wise addition—just as in the case of a polynomial or ghost-bit basis representation. To ensure the availability of an efficient multiplication procedure, one often restricts to a particular type of normal basis, which exists whenever $8 \nmid m$. In this thesis we focus entirely on these so-called *Gaussian normal bases*; see also [13, 24] for further background and proofs of the properties that are relevant for our purposes.

Definition 2.2.4 (Gaussian normal basis)**.**
Assume that $t \geq 1$ *such that* $p = tm + 1$ *is prime and the index of the subgroup generated by* $2 \in \mathbb{F}_p^*$ *is coprime to* m*. Let* $\alpha \in \mathbb{F}_{2^{mt}}$ *be a primitive* p*-th root of unity, and let* $u \in \mathbb{F}_p^*$ *have order* t*. Then*

$$\left\{ \sum_{j=0}^{t-1} \alpha^{u^j}, \left(\sum_{j=0}^{t-1} \alpha^{u^j} \right)^{2^1}, \ldots, \left(\sum_{j=0}^{t-1} \alpha^{u^j} \right)^{2^{m-1}} \right\}$$

is a normal basis of $\mathbb{F}_{2^m}$*, commonly referred to as* type t Gaussian normal basis.[1]

The complexity of multiplication with respect to a Gaussian normal basis representation is reflected by its type t. The Digital Signature Standard [35, Appendix D.1.3] offers several practical examples for (extension degree, type)-pairs of binary fields $\mathbb{F}_{2^m}$: $(163, 4)$, $(233, 2)$, $(283, 6)$, $(409, 4)$, and $(571, 10)$. For cryptographic applications, one is interested in situations where the type t is small. Hence, in our analysis we regard t as a (small) constant.

[1] The basis elements are known as *Gauss periods of type* (m, t), but we do not need this terminology here.

- Using one qubit per coefficient of α, adding $|\alpha\rangle$ to an m-qubit input $|\beta\rangle$ can be done in the obvious way with m CNOT gates, conditioned on the individual α_i. These CNOT gates operate on disjoint wires, and hence this adder can be realized in depth 1.

- Squaring corresponds to a cyclic (right-)shift of the coefficient vector:

$$\begin{array}{ccc} \mathbb{F}_{2^m} & \longrightarrow & \mathbb{F}_{2^m} \\ \sum_{i=0}^{m-1} \alpha_i \eta^{2^i} & \longmapsto & \sum_{i=0}^{m-1} \alpha_{i-1(\bmod m)} \eta^{2^i} \end{array}$$

- With the notation from Definition 2.2.4, define $F(1), F(2), \ldots, F(p-1)$ through $F(2^i u^j \bmod p) = i$ for $0 \leq i < m$ and $0 \leq j < t$. Then the representation $(\gamma_0, \ldots, \gamma_{m-1})$ of the product $\gamma = \alpha \cdot \beta$ can be computed as $\gamma_i =$

$$\begin{cases} \sum_{k=1}^{tm-1} \alpha_{F(k+1)+i} \beta_{F(p-k)+i} & , \text{if } 2 \mid t \\ \sum_{k=1}^{tm-1} \alpha_{F(k+1)+i} \beta_{F(p-k)+i} + \sum_{k=1}^{m/2} (\alpha_{k-1+i} \beta_{k-1+\frac{m}{2}+i} + \alpha_{k-1+\frac{m}{2}+i} \beta_{k-1+i}), & \text{if } 2 \nmid t \end{cases} \tag{2.5}$$

for $i = 0, \ldots, m-1$ (with all indices being understood modulo m).

Example 2.2.3 (Gaussian normal basis)**.** *For $\mathbb{F}_{2^5}$ there exists a Gaussian normal basis of type $t = 2$: we have $p = 2 \cdot 5 + 1 = 11$, and 2 is a generator of $\mathbb{F}_p^*$, so the index of the subgroup generated by $2 \in \mathbb{F}_p^*$ is certainly coprime to $m = 5$. Choosing $u = 10 \in \mathbb{F}_{11}^*$ as an element of order $t = 2$, we compute*

$F(1)$	$F(2)$	$F(3)$	$F(4)$	$F(5)$	$F(6)$	$F(7)$	$F(8)$	$F(9)$	$F(10)$
0	1	3	2	4	4	2	3	1	0

.

Now, from Equation (2.5) for the general multiplication $\gamma = \alpha \cdot \beta$, we obtain

$$\begin{aligned} \gamma_i \;=\; & \alpha_{1+i}\beta_i + \alpha_{3+i}\beta_{1+i} + \alpha_{2+i}\beta_{3+i} + \alpha_{4+i}\beta_{2+i} + \alpha_{4+i}\beta_{4+i} + \\ & \alpha_{2+i}\beta_{4+i} + \alpha_{3+i}\beta_{2+i} + \alpha_{1+i}\beta_{3+i} + \alpha_i\beta_{1+i} \end{aligned} \tag{2.6}$$

for $i = 0, \ldots, m-1$.

2.2.4 Computing multiplicative inverses with the Itoh-Tsujii algorithm

With a field representation where squaring is inexpensive, looking at an alternative to Euclid's algorithm that is exponentiation-based for computing multiplicative inverses becomes worthwhile. For any $\alpha \in \mathbb{F}_{2^m}^*$, we have $\alpha^{2^m-1} = 1$, and hence $\alpha^{-1} = \alpha^{2^m-2}$ can be found by raising α to the power $2^m - 2$. The almost maximal Hamming weight of the latter makes a naive square-and-multiply implementation problematic. Happily, a technique by Itoh and Tsujii [23] enables an efficient implementation of this exponentiation (see, e. g., [20, 23, 40, 45]). We begin by writing

$$m-1 = \sum_{i=1}^{\mathrm{hw}(m-1)} 2^{k_i} \quad \text{, where } \lfloor \log_2(m-1) \rfloor = k_1 > k_2 > \cdots > k_{\mathrm{hw}(m-1)} \geq 0,$$

and $\mathrm{hw}(\cdot)$ denotes the Hamming weight. Now, for fixed $\alpha \in \mathbb{F}_{2^m}^*$ and for $i \geq 0$, we define $\beta_i = \alpha^{2^i-1}$. In particular, $\beta_0 = 1$, $\beta_1 = \alpha$, and the inverse of α can be obtained as $\alpha^{-1} = (\beta_{m-1})^2$. So once we know β_{m-1}, only one final squaring is needed—which for a ghost-bit or a normal basis representation is just a permutation. To compute β_{m-1}, we exploit the fact that for all non-negative integers i, j the relation

$$\beta_{i+j} = \beta_i \cdot \beta_j^{2^i} \tag{2.7}$$

holds. By repeatedly applying Equation (2.7) with $i = j$, we see that computing all of $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$ requires no more than $\lfloor \log_2(m-1) \rfloor$ multiplications in $\mathbb{F}_{2^m}^*$ and $\lfloor \log_2(m-1) \rfloor$ exponentiations by a power of 2. In a ghost-bit or a Gaussian normal basis representation, all occurring exponentiations are (α-independent) permutations, and as the multiplications are of the form $\beta_j \cdot (\beta_j)^{2^j}$, to save resources we will exploit the fact that $(\beta_j)^{2^j}$ can be derived from β—there is no need to implement a general multiplier.

Beginning with $\beta_{2^{k_1}}$, we use Equation (2.7) to calculate $\beta_{2^{k_1}+2^{k_2}}$ and then iterate this process to obtain $\beta_{2^{k_1}+2^{k_2}+2^{k_3}}$, etc., until we finally reach

$$\beta_{m-1} = \beta_{2^{k_1}+2^{k_2}+\cdots+2^{k_{\mathrm{hw}(m-1)}}}.$$

Hence, with $\beta_{2^{k_1}}, \ldots, \beta_{2^{k_{\mathrm{hw}(m-1)}}}$ being available, $\mathrm{hw}(m-1)-1$ multiplications in $\mathbb{F}_{2^m}^*$ and $\mathrm{hw}(m-1)-1$ exponentiations by a power of 2 suffice to derive β_{m-1}.

Example 2.2.4 (Itoh-Tsujii inversion)**.** *For $m = 7$, we have $m-1 = 6 = 2^2 + 2^1$, so given an input $\alpha = \beta_{2^0} \in \mathbb{F}_{2^7}^*$, with $2 \leq \lfloor \log_2(6) \rfloor$ applications of Equation* (2.7) *we can find β_{2^1} and β_{2^2}. Then, with $1 = \mathrm{hw}(6)-1$ additional application of Equation* (2.7)*, we obtain $\beta_{2^2+2^1}$. After a final squaring—which in the case of a ghost-bit or a Gaussian normal basis representation is just a permutation of coefficients—yields $\alpha^{-1} = \beta_{2^2+2^1}^2$.*

2.3 BINARY ELLIPTIC CURVES

In the following, we review some basic information on binary elliptic curve representations, coordinate choices, and arithmetic operations. We claim no originality for this section.

2.3.1 Short Weierstrass form

Perhaps the most common representation of ordinary elliptic curves in characteristic 2 is a short Weierstrass form.

Definition 2.3.1 (Short Weierstrass)**.** *Let $m \in \mathbb{N}$ be a positive integer and $\mathbb{F}_{2^m}$ a finite field of size 2^m. For cryptographic applications, typical values are $m \in \{163, 233, 283\}$ [35]. A representation of an ordinary elliptic curve in characteristic 2 is a* short Weierstrass form *given by a polynomial in $\mathbb{F}_{2^m}[x, y]$:*

$$y^2 + xy = x^3 + a_2 x^2 + a_6 \tag{2.8}$$

Here $a_2, a_6 \in \mathbb{F}_{2^m}$, with $a_6 \neq 0$, and for practical purposes one often has $a_2 \in \{0, 1\}$ (cf. [35]). We write

$$\mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^m}) := \{(u, v) \in \mathbb{F}_{2^m}^2 : v^2 + uv = u^3 + a_2 u^2 + a_6\} \cup \{\mathcal{O}\}$$

for the ($\mathbb{F}_{2^m}$-rational points on the) elliptic curve given by Equation (2.8)*. The point* $\mathcal{O} \in \mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^m})$ *corresponds to the 'point at infinity.'*[2]

2.3.2 Group law

Because $a_6 \neq 0$, we have $(0,0) \notin \mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^m})$, suggesting $(0,0) \in \mathbb{F}_{2^m}^2$ as a convenient representation of $\mathcal{O}$. Hence, each curve point can be naturally represented as a pair of two field elements (which fit into $2m$ qubits) and we introduce the below group law for addition.

Definition 2.3.2 (Group law - point addition)**.** *The elliptic curve* $\mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^m})$ *is equipped with a natural group structure, where* $\mathcal{O}$ *serves as the identity. Namely, for* $P_1 = (x_1, y_1)$ *and* $P_2 = (x_2, y_2)$*, their sum* $P_3 = P_1 + P_2$ *can be computed by the procedure in Figure 2.5, which is taken from [44].*

2.3.3 Complete binary Edwards curves

Definition 2.3.3 (Complete binary Edwards curves)**.** *Let* $d_1, d_2 \in \mathbb{F}_{2^m}$ *with* $\mathrm{Tr}(d_2) = 1$.[3] *Then the* complete binary Edwards curve with coefficients d_1 and d_2 *is the affine curve defined by*

$$d_1(x+y) + d_2(x^2+y^2) = xy + xy(x+y) + x^2y^2.$$

We will write $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^m})$ *for the set of ($\mathbb{F}_{2^m}$-rational) points on this curve.*

[2] More technically, $\mathcal{O}$ is the unique point that is obtained when passing to the projective closure of E_{a_2,a_6}.

[3] For $\alpha \in \mathbb{F}_{2^m}$, the *trace* $\mathrm{Tr}(\alpha)$ can be obtained as $\mathrm{Tr}(\alpha) = \alpha^{2^0} + \alpha^{2^1} + \cdots + \alpha^{2^{m-1}}$.

Figure 2.5: Adding two points on the elliptic curve $y^2 + xy = x^3 + a_2x^2 + a_6$

if $\mathcal{P}_1 = O$ **then**

    return $\mathcal{P}_2$

if $\mathcal{P}_2 = O$ **then**

    return $\mathcal{P}_1$

if $x_1 = x_2$ **then**

    if $y_1 + y_2 = x_2$ and $\mathcal{P}_1 = -P_2$ **then**

        return $\mathcal{O}$

    else if $y_1 + y_2 = x_2$ and $\mathcal{P}_1 = \mathcal{P}_2$ **then**

        $\lambda \leftarrow x_2 + y_2/x_2$

        $x_3 \leftarrow \lambda^2 + \lambda + a_2$

        $y_3 \leftarrow x_2^2 + (\lambda + 1)x_3$

    else $(P_1 \neq \pm P_2)$

        $\lambda \leftarrow (y_1 + y_2)/(x_1 + x_2)$

        $x_3 \leftarrow \lambda^2 + \lambda + x_1 + x_2 + a_2$

        $y_3 \leftarrow (x_2 + x_3)\lambda + x_3 + y_2$

return (x_3, y_3)

CHAPTER 3

QUANTUM CIRCUITS FOR FINITE FIELD ARITHMETIC

3.1 MULTIPLYING IN LINEAR DEPTH USING GHOST-BIT AND GAUSSIAN NORMAL BASIS REPRESENTATIONS

For implementing the inverter discussed in the sequel, the multiplication of field elements plays a crucial role. As we are interested in Gaussian normal basis and ghost-bit basis representations, we begin by detailing linear-depth circuits for multiplication in each of these representations.

3.1.1 Linear depth multiplication using a ghost-bit basis

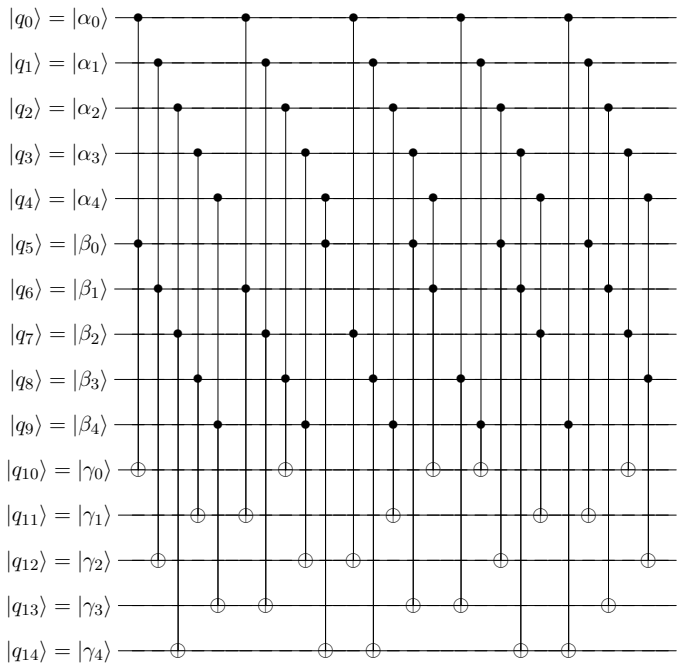
To multiply two $(m+1)$-bit inputs $|\alpha\rangle$ and $|\beta\rangle$ which represent field elements $\alpha, \beta \in \mathbb{F}_{2^m}$ in a ghost-bit basis, Formula (2.4) immediately yields a circuit consisting of $(m+1)^2$ Toffoli gates: each individual product $\alpha_j \beta_{(i-j) \bmod (m+1)}$ corresponds to a single Toffoli gate. Adopting an observation from [31], we recognize that these $(m+1)^2$ Toffoli gates can be evaluated in linear depth: for fixed $(i-2j) \bmod (m+1)$, the Toffoli gates to compute the $m+1$ products $\alpha_j \beta_{(i-j) \bmod (m+1)}$ $(j = 0, \ldots, m)$ operate on disjoint wires. Consequently, we can evaluate these $m+1$ Toffoli gates in parallel, and iterating over all $m+1$ possible values for $(i-2j) \bmod (m+1)$, we obtain a multiplier of depth $m+1$. This establishes the following result, which for the special case $|\xi\rangle = |0\rangle$ yields a basic multiplier.

Proposition 3.1.1. *If a ghost-bit basis representation of $\mathbb{F}_{2^m}$ is available, the multiplication $|\alpha\rangle |\beta\rangle |\xi\rangle \mapsto |\alpha\rangle |\beta\rangle |\xi + \alpha\beta\rangle$ with $\alpha, \beta, \xi \in \mathbb{F}_{2^m}$ can be realized in depth $m+1$ with $m^2 + 2m + 1$ Toffoli gates.*

As a concrete example of a ghost-bit basis multiplier, let us apply the above proposition to the field with 16 elements.

Example 3.1.1. *Consider, from Example 2.2.2, the ghost-bit basis representation of* $\mathbb{F}_{2^4}$. *In this case, evaluating all terms* $\alpha_j\beta_{(i-j) \bmod 5}$ *in order for* $(i-2j) \bmod 5 = 0, 1, 2, 3, 4$ *yields a multiplier of depth* 5, *consisting of* $5 \cdot 5 = 25$ *Toffoli gates, as shown in Figure 3.1.*

Figure 3.1: A ghost-bit basis multiplier for $\alpha \cdot \beta \in \mathbb{F}_{2^4}$



Next, we consider the special case of computing products $\alpha \cdot \alpha^{2^j}$ with a fixed j, as occurring in the Itoh-Tsujii algorithm described in Section 2.2.4. This variant of our multiplier takes as input the ghost-bit basis representation $(\alpha_0, \ldots, \alpha_m) \in \mathbb{F}_2^{m+1}$ of some $\alpha \in \mathbb{F}_{2^m}$ and a $|0\rangle$-initialized $m+1$-bit register, in which the ghost-bit basis representation $(\gamma_0, \gamma_1, \ldots, \gamma_m)$ of $\gamma = \alpha \cdot \alpha^{2^j}$ will be stored. The total number of wires required is only $2 \cdot (m+1)$. As we are using a ghost-bit basis representation, squaring is a simple permutation, and more generally, exponentiation by 2^r corresponds to a permutation. In particular, we can obtain the ghost-bit basis representation of α^{2^r} from $(\alpha_0, \alpha_1, \ldots, \alpha_m)$ by reading out the individual entries in a different order.

Hence, the following result confirms that the saving of $m+1$ wires can be done without sacrificing the property of having linear depth.

Proposition 3.1.2. *If a ghost-bit basis for $\mathbb{F}_{2^m}$ is available, then for any fixed $r \in \{0,\ldots,m\}$ the multiplication $|\alpha\rangle\,|\xi\rangle \mapsto |\alpha\rangle\,|\xi + \alpha\cdot\alpha^{2^r}\rangle$ with $\alpha,\xi \in \mathbb{F}_{2^m}$ can be realized in depth $2m+2$ using m^2+m Toffoli and $m+1$ CNOT gates.*

Proof: Let $\alpha = \sum_{i=0}^{m} \alpha_i x^i + (x^{m+1}+1)$ be a ghost-bit basis representation for $\alpha \in \mathbb{F}_{2^m}$. Then Equation (2.3) yields $\alpha^{2^r} = \sum_{i=0}^{m} \alpha_{\pi^{-r}(i)} x^i + (x^{m+1}+1)$, and with Equation (2.4) we recognize the i^{th} coefficient of $\alpha\cdot\alpha^{2^r}$ as

$$\gamma_i = \sum_{j=0}^{m} \alpha_j \alpha_{\pi^{-r}((i-j) \bmod (m+1))} \quad (i=0,\ldots,m).$$

As applying π can be seen as doubling modulo $m+1$, applying π^{-r} translates into division by 2^r modulo $m+1$. We may assume that $2^r \neq 1 \bmod (m+1)$, as otherwise $r \in \{0,m\}$, and exponentiation with 2^r becomes the identity on $\mathbb{F}_{2^m}$. Then, for any fixed 'index sum' $\sigma \in \{0,\ldots,m\}$, there are exactly $m+1$ pairs $(i,j) \in \{0,\ldots,m\}^2$ satisfying

$$\pi^{-r}((i-j) \bmod (m+1)) + j = \sigma \bmod (m+1). \tag{3.1}$$

Namely, for each $i \in \{0,\ldots,m\}$ we obtain a unique corresponding $j \in \{0,\ldots,m\}$ by solving the linear equation

$$2^{-r}\cdot(i-j) + j = \sigma \bmod (m+1)$$

for j—at this we divide by $1-2^{-r} \pmod{m+1}$ which is possible as $2^r \neq 1$. The subsequent argument shows that we can compute the $m+1$ products $\alpha_j \alpha_{\pi^{-r}((i-j) \bmod (m+1))}$ for those (i,j)-pairs satisfying Equation (3.1) in depth 2. By arranging our circuit such that the values $\sigma = 0,\ldots,m$ are processed in order, we achieve the claimed overall depth of $2m+2$.

Suppose we have two products $\alpha_j \alpha_{\pi^{-r}((i-j) \bmod (m+1))}$ and $\alpha_{j'} \alpha_{\pi^{-r}((i'-j') \bmod (m+1))}$ satisfying

$$\pi^{-r}((i-j) \bmod (m+1)) + j = \sigma = \pi^{-r}((i'-j') \bmod (m+1)) + j',$$

then we may assume $j \neq j'$, as otherwise

$$\pi^{-r}((i-j) \bmod (m+1)) = \pi^{-r}((i'-j') \bmod (m+1)),$$

and there is nothing to show. Consequently, the two gates evaluating the two terms

$$\alpha_j \alpha_{\pi^{-r}((i-j) \bmod (m+1))} \text{ and } \alpha_{j'} \alpha_{\pi^{-r}((i'-j') \bmod (m+1))}$$

have different target bits. We can evaluate these two terms in parallel whenever the intersection

$$\{j, \pi^{-r}((i-j) \bmod (m+1))\} \cap \{j', \pi^{-r}((i'-j') \bmod (m+1)\}$$

is empty—in this case the corresponding gates operate on disjoint wires. To better understand the situation, let us define an undirected graph $\mathfrak{G}$ with vertex set $\mathbb{Z}/(m+1)$, so that vertex $i+(m+1)$ corresponds to the wire representing α_i. We connect two vertices, whenever they serve as control bits for the same gate; i. e., we include the edges

$$\{j \bmod (m+1), \pi^{-r}((i-j) \bmod (m+1)) \bmod (m+1)\}$$

for all $i, j \in \mathbb{Z}/(m+1)$ with $\pi^{-r}((i-j) \bmod (m+1)) + j = \sigma \bmod (m+1)$. In particular, we obtain exactly one self-loop ($j = \sigma/2 \bmod (m+1)$). Instead of using the above description of the edges, we can equivalently include all edges

$$\{j \bmod (m+1), \sigma - j \bmod (m+1)\}$$

for $j \in \mathbb{Z}/(m+1)$. Because $\sigma - (\sigma - j) = j \bmod (m+1)$, we see that the resulting graph $\mathfrak{G}$ consists of $m/2$ vertex pairs, each connected by two parallel edges, and one isolated point (namely $\sigma/2 \bmod (m+1)$) with a self-loop, corresponding to a CNOT. Consequently, two colors suffice to color the edges in such a way, that no neighboring edges share a color. Now all gates corresponding to an edge with the same color operate on disjoint wires and hence can be evaluated in parallel. □

To illustrate the "wire saving" offered by Proposition 3.1.2, let us again consider the field with 16 elements.

Example 3.1.2. *For $r = 2$, the permutation π^{-r} corresponds to a multiplication with $2^{-2} = -1 \bmod 5$, i. e., we have to find*

$$\gamma_i = \alpha_0\alpha_{-i \bmod 5} + \alpha_1\alpha_{(1-i) \bmod 5} + \alpha_2\alpha_{(2-i) \bmod 5} + \alpha_3\alpha_{(3-i) \bmod 5} + \alpha_4\alpha_{(4-i) \bmod 5}$$

for $i = 0, \ldots, 4$. Using the condition $2 \cdot j - i = \sigma \bmod 5$, each of the occurring 25 terms can be associated with a particular value of σ:

$\sigma = 0$**:** $\alpha_0\alpha_0$, $\alpha_1\alpha_4$, $\alpha_2\alpha_3$, $\alpha_3\alpha_2$, $\alpha_4\alpha_1$

$\sigma = 1$**:** $\alpha_0\alpha_1$, $\alpha_1\alpha_0$, $\alpha_2\alpha_4$, $\alpha_3\alpha_3$, $\alpha_4\alpha_2$

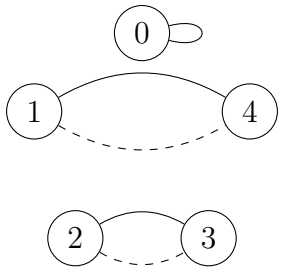
$\sigma = 2$**:** $\alpha_0\alpha_2$, $\alpha_1\alpha_1$, $\alpha_2\alpha_0$, $\alpha_3\alpha_4$, $\alpha_4\alpha_3$

$\sigma = 3$**:** $\alpha_0\alpha_3$, $\alpha_1\alpha_2$, $\alpha_2\alpha_1$, $\alpha_3\alpha_0$, $\alpha_4\alpha_4$

$\sigma = 4$**:** $\alpha_0\alpha_4$, $\alpha_1\alpha_3$, $\alpha_2\alpha_2$, $\alpha_3\alpha_1$, $\alpha_4\alpha_0$

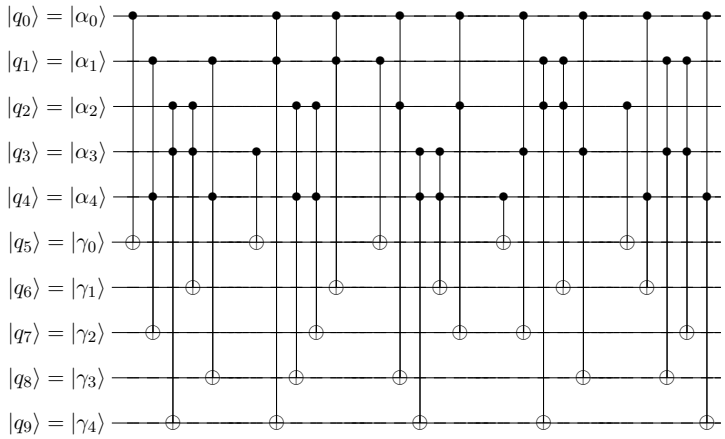
The resulting graph for $\sigma = 0$ is shown in Figure 3.2.

Figure 3.2: Graph representing the term for $\sigma = 0$ as described in Example 3.1.2. The 2-coloring of the edges—where the different line styles indicate the colors—translates into a depth 2 circuit for this term.



Each edge corresponds to one gate, and with the 2-coloring of the edges we obtain a depth 2 circuit for evaluating the terms associated with $\sigma = 0$ (and add them to the respective input/partial result γ_i). Applying a similar reasoning to the other σ-values, we obtain a circuit of depth 10 for implementing the map $|\alpha\rangle|\xi\rangle \mapsto |\alpha\rangle|\xi + \alpha \cdot \alpha^4\rangle$ for $\alpha, \xi \in \mathbb{F}_{2^4}$, as seen in Figure 3.3.

Figure 3.3: A ghost-bit basis multiplier for $\alpha \cdot \alpha^{2^2} \in \mathbb{F}_{2^4}$



3.1.2 Linear depth multiplication using a Gaussian normal basis

Assume $\mathbb{F}_{2^m}$ has a Gaussian normal basis of type t. Our multiplier takes as input the normal basis representations $(\alpha_0, \alpha_1, \ldots, \alpha_{m-1}) \in \mathbb{F}_2^m$ and $(\beta_0, \beta_1, \ldots, \beta_{m-1}) \in \mathbb{F}_2^m$ of two elements $\alpha, \beta \in \mathbb{F}_{2^m}$, along with a $|0\rangle$-initialized m-bit register, in which the normal basis representation $(\gamma_0, \gamma_1, \ldots, \gamma_{m-1})$ of $\gamma = \alpha \cdot \beta$ will be stored. Consequently, the total number of wires is $3m$. Each coefficient product $\alpha_j \beta_k$ in Equation (2.5) can be realized with a Toffoli gate, and so for a fixed $i \in \{0, \ldots, m-1\}$ we can compute γ_i with at most

$$\begin{cases} tm - 1 \text{ consecutive Toffoli gates} & \text{, if } t \text{ is even} \\ tm - 1 + 2 \cdot (m/2) = (t+1)m - 1 \text{ consecutive Toffoli gates} & \text{, if } t \text{ is odd} \end{cases}.$$

From this we immediately obtain an overall gate count of $(t + (t \bmod 2)) \cdot m^2 - m$ Toffoli gates for our normal basis multiplier. This multiplier can be realized in linear depth: fix an arbitrary $k \in \{1, \ldots, tm-1\}$ and two different positions $i, i' \in \{0, \ldots, m-1\}$ in the normal basis representation of the product $\gamma = \alpha \cdot \beta$. Then the Toffoli gates computing $\alpha_{F(k+1)+i}\beta_{F(p-k)+i}$ and $\alpha_{F(k+1)+i'}\beta_{F(p-k)+i'}$ operate on disjoint wire sets, as obviously

$$\begin{aligned} F(k+1)+i &\neq F(k+1)+i' \pmod m \text{ and} \\ F(p-k)+i &\neq F(p-k)+i' \pmod m. \end{aligned}$$

For odd t, we see analogously that $\alpha_{k-1+i}\beta_{k-1+\frac{m}{2}+i}$ can be calculated in parallel with $\alpha_{k-1+i'}\beta_{k-1+\frac{m}{2}+i'}$ for all $i \neq i'$, and $\alpha_{k-1+\frac{m}{2}+i}\beta_{k-1+i}$ can be calculated in parallel with $\alpha_{k-1+\frac{m}{2}+i'}\beta_{k-1+i'}$ for all $i \neq i'$, as summarized in the following result.

Proposition 3.1.3. *If a Gaussian normal basis of type t is available for $\mathbb{F}_{2^m}$, the multiplication $|\alpha\rangle|\beta\rangle|\xi\rangle \mapsto |\alpha\rangle|\beta\rangle|\xi + \alpha\beta\rangle$ of two field elements $\alpha, \beta \in \mathbb{F}_{2^m}$ can be realized in depth $(t + (t \bmod 2)) \cdot m - 1$ using $(t + (t \bmod 2)) \cdot m^2 - m$ Toffoli gates.*

As a concrete example of a Gaussian normal basis multiplier, let us apply the above proposition to the field with 32 elements.

Example 3.1.3. *Consider the type 2 Gaussian normal basis from Example 2.2.3. Here the product $\gamma = \alpha \cdot \beta$ of $\alpha, \beta \in \mathbb{F}_{2^5}$, is represented by $(\gamma_0, \ldots, \gamma_{m-1})$ with*

$$\begin{aligned}\gamma_i \;=\; & \alpha_{1+i}\beta_i + \alpha_{3+i}\beta_{1+i} + \alpha_{2+i}\beta_{3+i} + \alpha_{4+i}\beta_{2+i} + \alpha_{4+i}\beta_{4+i} + \\ & \alpha_{2+i}\beta_{4+i} + \alpha_{3+i}\beta_{2+i} + \alpha_{1+i}\beta_{3+i} + \alpha_i\beta_{1+i}.\end{aligned}$$

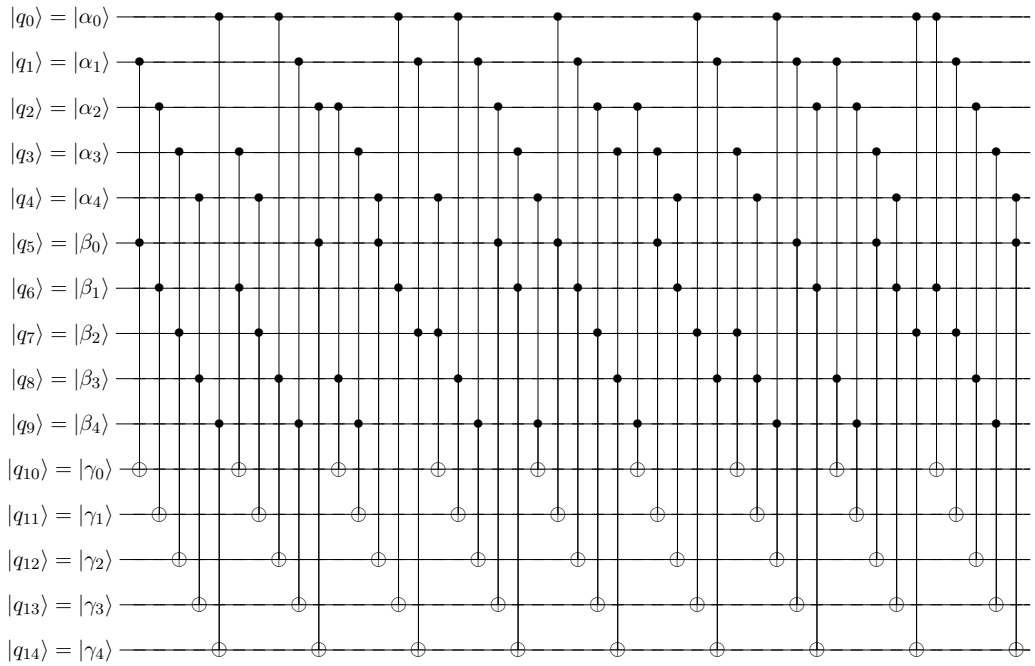
Implementing this summation term by term yields a normal basis multiplier for $\mathbb{F}_{2^5}$ comprised of $9 \cdot 5 = 45$ Toffoli gates and of total depth 9 (each term of the summation can be evaluated in parallel for $i = 0, \ldots, 4$), as seen in Figure 3.4.

Similarly, as in the case of a ghost-bit basis representation, it is possible to compute products of the form $\alpha \cdot \alpha^{2^r}$ in linear depth without having α^{2^r} represented as a separate input. Hence, the following result shows that the saving of m wires can be done without sacrificing the property of having linear depth.

Proposition 3.1.4. *If a Gaussian normal basis of type t is available for $\mathbb{F}_{2^m}$, for any fixed $r \in \{0, \ldots, m\}$ the multiplication $|\alpha\rangle|\xi\rangle \mapsto |\alpha\rangle|\xi + \alpha \cdot \alpha^{2^r}\rangle$ for $\alpha \in \mathbb{F}_{2^m}$ can be realized in depth $3 \cdot (t + (t \bmod 2)) \cdot m - 3$ using $(t + (t \bmod 2)) \cdot m^2 - m$ gates (CNOT or Toffoli).*

Proof: Using Equation (2.5) to calculate the product $\alpha \cdot \alpha^{2^j}$ again, the upper bound for the total number of gates remains unchanged. It could happen, however,

Figure 3.4: A Gaussian normal basis multiplier for $\alpha \cdot \beta \in \mathbb{F}_{2^5}$



that the control bits of a Toffoli gate end up on the same wire, so that instead of a Toffoli we obtain a CNOT gate.

To argue that the circuit depth grows at most by a factor of 3, we fix $k \in \{1, \ldots, tm-1\}$ arbitrary. Then $\beta_k = \alpha_{k-r}$, and we claim that all m terms

$$\alpha_{F(k+1)+i}\beta_{F(p-k)+i} = \alpha_{F(k+1)+i}\alpha_{F(p-k)-r+i} \quad (i = 0, \ldots, m-1) \tag{3.2}$$

can be calculated in parallel using depth at most 3.

Case $F(k+1) = F(p-k) - r \pmod{m}$**:** Here, instead of Toffoli gates, we have only CNOT gates operating on disjoint wires. Hence, all m terms can be computed at the same time, i., e., in depth 1.

Case $F(k+1) \neq F(p-k) - r \pmod{m}$**:** For $i \neq i'$, we can evaluate the terms

$$\alpha_{F(k+1)+i}\alpha_{F(p-k)-r+i} \text{ and } \alpha_{F(k+1)+i'}\alpha_{F(p-k)-r+i'}$$

in parallel whenever the two sets

$$\{F(k+1)+i, F(p-k)-r+i\} \text{ and } \{F(k+1)+i', F(p-k)-r+i'\}$$

have an empty intersection, meaning the two Toffoli gates operate on disjoint wires. We define an undirected graph $\mathfrak{G}$ with vertex set $\mathbb{Z}/(m)$—so vertex $i+(m)$ corresponds to the wire representing $\alpha_{i \pmod m}$—and edge set

$$E := \{\{F(k+1)+i \pmod m, F(p-k)-r+i \pmod m\} : i = 0, \ldots, m-1\};$$

i. e., each edge corresponds to one Toffoli gate. If we can find an edge coloring of this graph such that neighboring edges always have different colors, then all Toffoli gates corresponding to the same color can be calculated in parallel. We show that 3 colors will be sufficient, and hence a depth 3 circuit suffices to compute all the products in (3.2). For $\delta = F(p-k)-r-F(k+1)$, let $\langle\delta\rangle$ be the cyclic subgroup generated by $\delta+(m)$ in $\mathbb{Z}/(m)$, and let

$$\mathbb{Z}/(m) = G_1 \uplus \cdots \uplus G_t \tag{3.3}$$

be the decomposition of $\mathbb{Z}/(m)$ into $\langle\delta\rangle$-cosets. Rewriting the edge set E as

$$E = \{\{i \pmod m, i+\delta \pmod m\} | i \in \{0, \ldots, m-1\}\},$$

we see that the decompositon (3.3) actually yields a decomposition of the graph $\mathfrak{G}$—there are no edges between vertices in G_j and $G_{j'}$ if $j \neq j'$. Moreover, $\langle\delta\rangle$ is cyclic with generator $\delta+(m)$, so for each G_j, the subgraph of $\mathfrak{G}$ with vertex set G_j is a closed cycle on $\mathrm{ord}(\delta+(m))$ vertices. As such, we may alternatively color the edges in such a cycle red and blue. Then neighboring edges can only obtain the same color at the very last step when we try to close the cycle—this happens whenever $\mathrm{ord}(\delta+(m))$ is odd. Hence, for the last edge in a cycle, a third color may be needed. As there are no edges between the individual cycles, we have found the desired 3-coloring of E.

The above argument takes care of all even t-values, and for odd t-values the first of the summations in Equation (2.5) is taken care of as well.[1] To argue that for fixed

[1]For $\mathrm{ord}(\delta+(m)) = 2$ the sets G_j consist of two vertices, and we actually encounter graphs with a 2-coloring of the edges.

k the terms $\alpha_{k-1+i}\alpha_{k-1+\frac{m}{2}-r+i}$ $(i = 1, \ldots, m)$ and $\alpha_{k-1+\frac{m}{2}+i}\alpha_{k-1-r+i}$ $(i = 1, \ldots, m)$ can be computed in depth 3, we can use an analogous argument as above, replacing δ with $(m/2) - r$ and $(m/2) + r$, respectively. □

Example 3.1.4. *Continuing with the Gaussian normal basis representation of* $\mathbb{F}_{2^5}$ *from Example 3.1.3, let us consider the special case of a multiplication* $\gamma = \alpha \cdot \beta$ *where* $\beta = \alpha^{2^1}$*, i. e.,* $r = 1$*. Then we have* $\beta_k = \alpha_{k-1}$ *and Equation (2.6) can be rewritten as* $\gamma_i =$

$$\underline{\alpha_{1+i}\alpha_{4+i}}+\alpha_{3+i}\alpha_i+\alpha_{2+i}\alpha_{2+i}+\underline{\alpha_{4+i}\alpha_{1+i}}+\alpha_{4+i}\alpha_{3+i}+\alpha_{2+i}\alpha_{3+i}+\alpha_{3+i}\alpha_{1+i}+\alpha_{1+i}\alpha_{2+i}+\alpha_i\alpha_i.$$

In particular, the addition of the terms $\alpha_{2+i}\alpha_{2+i}$ *and* $\alpha_i\alpha_i$ *can be implemented with CNOT instead of Toffoli gates, fulfilling the condition* $F(k+1) = F(11-k)-1$ *as in the first case of the above proof. We also note that the underlined terms cancel each other, which yields a simplification of our circuit that is not reflected by the upper bounds in Proposition 3.1.4.*

Going through the remaining values for k *(for which* $F(k+1) \neq F(11-k)-1$ *and no cancellation occurs), we obtain the following values* $\delta = F(11-k)-1-F(k+1)$*:*

k	2	5	6	7	8
δ	-3	-1	1	-2	1

As $m = 5$ *is prime, each* $\delta + (5)$ *generates the complete additive group* $\mathbb{Z}/(5)$*, and so the graph* $\mathfrak{G}$ *is simply a closed cycle. For instance, consider* $k = 5$ *such that* $\delta = -1$*. Then the graph in Figure 3.5 is obtained, where a vertex labeled* i $(i = 0, \ldots, 4)$ *represents the residue class* $i + (5)$*, and different line styles indicate different colors.*

As shown in Figure 3.6, this 3-coloring translates into a quantum circuit of depth 3 to compute the terms $\alpha_{4+i}\alpha_{3+i}$ $(i = 0, \ldots, 4)$ *(and add them to the respective input/partial result* γ_i*).*

Figure 3.5: Graph corresponding to the cosets $\delta + (5)$ for $\delta = -1$ as described in Example 3.1.4. The 3-coloring of the edges—where the different line styles in the pentagon indicate the three different colors—translates into a depth 3 circuit.

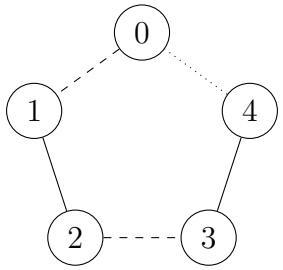
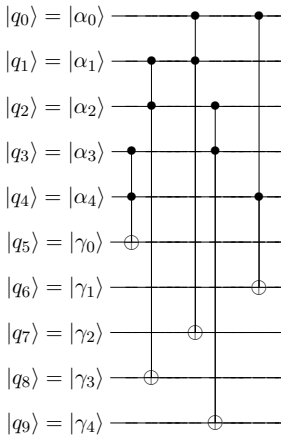


Figure 3.6: Part of a Gaussian normal basis multiplier for $\alpha \cdot \alpha^{2^1} \in \mathbb{F}_{2^5}$: computing the terms $\alpha_{4+i}\alpha_{3+i}$



3.2 INVERSION IN DEPTH O(M LOG(M)) USING THE ITOH-TSUJII ALGORITHM

With the linear depth multipliers from the previous section, we can now implement a depth $O(m \log(m))$ algorithm to invert field elements $\alpha \in \mathbb{F}_{2^m}^*$, if a Gaussian normal basis or ghost-bit basis representation is available.

The first part of the input is, respectively, an m or $(m+1)$-bit representation $|\alpha\rangle$ of the element $\alpha \in \mathbb{F}_{2^m}^*$ to be inverted.[2] Now, providing $\lfloor \log_2(m-1) \rfloor$ auxiliary registers

[2]The input $|0\rangle$ for $|\alpha\rangle$ results in the output $|0\rangle$ as "inverse."

that are initialized with $|0\rangle$, a sequence of $\lfloor \log_2(m-1) \rfloor$ consecutive multipliers can be used to calculate the values $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$ from Section 2.2.4—recall that $\beta_{2^0} = \alpha$. From Proposition 3.1.2 and Proposition 3.1.4, we obtain the following resource counts for this part of the inverter computation:

- If a ghost-bit basis representation of $\mathbb{F}_{2^m}$ is available, we can find all of

  $$\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$$

  in depth $\lfloor \log_2(m-1) \rfloor \cdot (2m+2)$ using $\lfloor \log_2(m-1) \rfloor \cdot (m^2+m)$ Toffoli and $\lfloor \log_2(m-1) \rfloor \cdot (m+1)$ CNOT gates. In doing so, $(1+\lfloor \log_2(m-1) \rfloor) \cdot (m+1)$ qubits suffice.

- Assume that a Gaussian normal basis representation of $\mathbb{F}_{2^m}$ is available. Then we can find all of $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$ in depth $\lfloor \log_2(m-1) \rfloor \cdot (3 \cdot (t+(t \bmod 2)) \cdot m - 3)$ using $\lfloor \log_2(m-1) \rfloor \cdot ((t+(t \bmod 2)) \cdot m^2 - m)$ gates (CNOT or Toffoli). In doing so, $(1+\lfloor \log_2(m-1) \rfloor) \cdot m$ qubits suffice.

At this point, our inverter has computed all of $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$ and stored each of these values in a separate set of wires. Next, we can use a sequence of $\mathrm{hw}(m-1)-1$ (general) multipliers, each obtaining an auxiliary input $|0\rangle$, to gather the actually needed values $\beta_{2^{k_1}}, \ldots, \beta_{2^{k_{\mathrm{hw}(m-1)}}}$ and form their product using Equation (2.7). All exponentiations of the form $\beta_j^{2^i}$ are free, in that a multiplier can just read out the coefficients of the respective β_j in permuted order to obtain the required input value. This is simply a permutation of the control bit positions. Consequently, we have the following resource counts:

- If a ghost-bit basis of $\mathbb{F}_{2^m}$ is available and given $|\beta_{2^{k_1}}\rangle, \ldots, |\beta_{2^{k_{\mathrm{hw}(m-1)}}}\rangle$, we can compute $|\beta_{m-1}\rangle$ in depth $(\mathrm{hw}(m-1)-1) \cdot (m+1)$ using $(\mathrm{hw}(m-1)-1) \cdot (m^2+2m+1)$ Toffoli gates. For the auxiliary inputs $|0\rangle$ respectively storing some intermediate results, $(\mathrm{hw}(m-1)-1) \cdot (m+1)$ qubits suffice.

- If a Gaussian normal basis of $\mathbb{F}_{2^m}$ is available and given $|\beta_{2^{k_1}}\rangle, \ldots, |\beta_{2^{k_{\mathrm{hw}(m-1)}}}\rangle$, we can compute $|\beta_{m-1}\rangle$ in depth $(\mathrm{hw}(m-1)-1)\cdot((t+(t \bmod 2))\cdot m-1)$ using $(\mathrm{hw}(m-1)-1)\cdot((t+(t \bmod 2))\cdot m^2-m)$ Toffoli gates. For the auxiliary inputs $|0\rangle$ respectively storing some intermediate results, $(\mathrm{hw}(m-1)-1)\cdot m$ qubits are needed.

The final squaring operation in the Itoh-Tsujii algorithm is again free, in that the last multiplier can simply write out the result in permuted order. In summary, we obtain the following estimate for a ghost-bit basis, where we double depth and gate count to account for the resources to "uncompute" auxiliary values—this is an upper bound, as the last multiplication actually does not have to be "undone."

Proposition 3.2.1. *If a ghost-bit basis for $\mathbb{F}_{2^m}$ is available, the inversion $|\alpha\rangle|0\rangle \mapsto |\alpha^{-1}\rangle|0\rangle$ can be implemented in depth $2\cdot\lfloor\log_2(m-1)\rfloor\cdot(2m+2)+2\cdot(\mathrm{hw}(m-1)-1)\cdot(m+1) = \mathrm{O}(m\log_2(m))$ and using $2\cdot\lfloor\log_2(m-1)\rfloor\cdot(m^2+m)+2\cdot(\mathrm{hw}(m-1)-1)\cdot(m^2+2m+1)$ Toffoli and $2\cdot\lfloor\log_2(m-1)\rfloor\cdot(m+1)$ CNOT gates. The inversion can be implemented with $(1+\lfloor\log_2(m-1)\rfloor)\cdot(m+1)+(\mathrm{hw}(m-1)-1)\cdot(m+1) = \mathrm{O}(m\log_2(m))$ qubits.*

Analogously, adding the respective bounds for the case of a Gaussian normal basis of type t yields the following estimate. If we consider t as constant, the depth of the resulting circuit is again in $\mathrm{O}(m\log_2(m))$.

Proposition 3.2.2. *If a Gaussian normal basis of type t for $\mathbb{F}_{2^m}$ is available, the inversion $|\alpha\rangle|0\rangle \mapsto |\alpha^{-1}\rangle|0\rangle$ can be implemented in depth $\lfloor\log_2(m-1)\rfloor\cdot(6\cdot(t+(t \bmod 2))\cdot m-6)+2\cdot(\mathrm{hw}(m-1)-1)\cdot((t+(t \bmod 2))\cdot m-1) = \mathrm{O}(m\log_2(m))$ using $2\cdot\lfloor\log_2(m-1)\rfloor\cdot((t+(t \bmod 2))\cdot m^2-m)+2\cdot(\mathrm{hw}(m-1)-1)\cdot((t+(t \bmod 2))\cdot m^2-m)$ gates (CNOT or Toffoli). The inversion can be implemented with $(1+\lfloor\log_2(m-1)\rfloor)\cdot m+(\mathrm{hw}(m-1)-1)\cdot m = \mathrm{O}(m\log_2(m))$ qubits.*

It is worth noting that if our extension degree m has the form $m = 2^n+1$, e. g., for m being a Fermat prime, the Hamming weight of $m-1$ is one; i. e., we can restrict

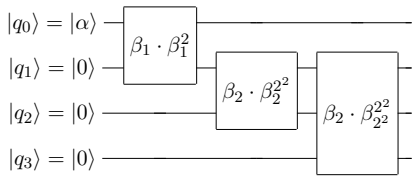
entirely to special multipliers as described in Proposition 3.1.2 and Proposition 3.1.4. As in the general case, the last multiplier can output the result $\beta_{2^{m-1}}$ in permuted order, so that the correct inverse $(\beta_{m-1})^2$ is obtained without the need to implement a squaring operation.

Avoiding such a special case, the following example illustrates the structure of the discussed inverter with an extension of degree 7, where a general multiplier with two arguments is brought into use.

Example 3.2.1. *Consider the field $\mathbb{F}_{2^7}$ discussed in Example 2.2.4, and assume a Gaussian normal basis representation is used. Then, to compute α^{-1} from an input $\alpha = \beta_{2^0} \in \mathbb{F}_{2^7}^*$, we can use two special multipliers as described in Proposition 3.1.4 to compute β_{2^1} and β_{2^2}. Interpreting the input wires in appropriately permuted order, one general multiplier suffices to compute $\beta_{2^2+2^1}$. In addition, writing the output in appropriately permuted order, the output of this multiplier is actually $\beta_{2^2+2^1}^2$.*

Representing an m-qubit input by a single wire, the structure of the resulting inverter in $\mathbb{F}_{2^7}^*$ is summarized below in Figure 3.7.

Figure 3.7: A ghost-bit or Gaussian normal basis inverter for $\alpha \in \mathbb{F}_{2^7}^*$



Finally, we obtain as direct consequence of Propositions 3.2.1 and 3.2.2 the following corollary which gives an upper bound on the number of T-gates to perform inversion in a binary finite field where a ghost-bit basis or a Gaussian normal basis representation is available. According to [36, Chapter 4.2], a Toffoli gate requires the use of 7 T-gates in a circuit of overall T-depth of 6.

Corollary 3.2.1. *If a ghost-bit basis for $\mathbb{F}_{2^m}$ is available, an inverter can be implemented with a T-depth of at most* $12 \cdot \lfloor \log_2(m-1) \rfloor \cdot (2m+2) + 12 \cdot (\mathrm{hw}(m-1)-1) \cdot (m+1)$

and using no more than $14\cdot\lfloor\log_2(m-1)\rfloor\cdot(m^2+m)+14\cdot(\mathrm{hw}(m-1)-1)\cdot(m^2+2m+1)$ *many* T*-gates.*

If a Gaussian normal basis of type t *for* $\mathbb{F}_{2^m}$ *is available, an inverter can be implemented with a* T*-depth of at most* $6\cdot\lfloor\log_2(m-1)\rfloor\cdot(6\cdot(t+(t \bmod 2))\cdot m-6)+(12\cdot\mathrm{hw}(m-1)-6)\cdot((t+(t \bmod 2))\cdot m-1)$ *using at most* $14\cdot\lfloor\log_2(m-1)\rfloor\cdot((t+(t \bmod 2))\cdot m^2-m)+14\cdot(\mathrm{hw}(m-1)-1)\cdot((t+(t \bmod 2))\cdot m^2-m)$ *many* T*-gates.*

CHAPTER 4
QUANTUM CIRCUITS FOR BINARY ELLIPTIC CURVE ARITHMETIC

4.1 FIXING A FINITE FIELD REPRESENTATION

Fast addition formulae for points on an elliptic curve over a finite binary field $\mathbb{F}_{2^m}$ aim at reducing the number of (expensive) $\mathbb{F}_{2^m}$-operations. The following operations are of particular interest:

Addition: Given $\alpha, \beta \in \mathbb{F}_{2^m}$, compute their sum $\alpha + \beta$.

Multiplication: Given $\alpha, \beta \in \mathbb{F}_{2^m}$, compute their product $\alpha \cdot \beta$.

Multiplication by a constant: For a fixed non-zero constant $\gamma \in \mathbb{F}_{2^m}^*$, on input $\alpha \in \mathbb{F}_{2^m}$, compute $\gamma \cdot \alpha$. The value γ, for example, could be a coefficient in the defining equation of an elliptic curve.

Squaring: Given $\alpha \in \mathbb{F}_{2^m}$, compute α^2.

If one is interested in a unique representation of curve points, then the inversion of $\mathbb{F}_{2^m}$-elements also comes into play.

Inversion: Given $\alpha \in \mathbb{F}_{2^m}^*$, find $\alpha^{-1} \in \mathbb{F}_{2^m}$.

The specific cost of each operation depends on how the field $\mathbb{F}_{2^m}$ is represented, and in the next two sections we look at three representations that have been considered in the literature on quantum circuits.

4.1.1 Polynomial basis representation

In a polynomial basis representation, $\mathbb{F}_{2^m}$ is identified with a quotient field $\mathbb{F}_2[x]/(f)$ where $f \in \mathbb{F}_2[x]$ is an irreducible polynomial of degree m. Each $\alpha \in \mathbb{F}_{2^m}$ is represented by the unique sequence $(\alpha_0, \ldots, \alpha_{m-1}) \in \mathbb{F}_2^m$ with $\alpha = \sum_{i=0}^{m-1} \alpha_i x^i + (f)$. In a quantum circuit, we store each coefficient α_i in a separate qubit. Quantum arithmetic in such a representation has been explored by a number of authors, including Beauregard et al. [7], Kaye and Zalka [26], and Maslov et al. [30]. For each of the four basic tasks mentioned above, the exact implementation complexity varies depending on the particular choice of f, and efficient circuits are available:

Addition: As addition is defined coefficient-wise, m CNOT gates are sufficient to derive the representation of $\alpha + \beta$ from those of α and β. These gates operate on disjoint wires and can be implemented in depth 1. To realize an addition $|\alpha\rangle |\beta\rangle |0\rangle \mapsto |\alpha\rangle |\beta\rangle |\alpha + \beta\rangle$ where the sum is stored in a separate register, we can first add $|\alpha\rangle$ to $|0\rangle$, followed by adding $|\beta\rangle$, i. e., $2m$ CNOT gates and depth 2 suffice. In particular, we do not need a single T-gate to implement $\mathbb{F}_{2^m}$-addition.

Multiplication: Building on a classical Mastrovito multiplier [32, 33, 39], in [30] a linear depth quantum circuit is presented which derives the product $\alpha \cdot \beta$ from $\alpha, \beta \in \mathbb{F}_{2^m}$. This circuit requires m^2 Toffoli gates and $m^2 - 1$ CNOT gates. In particular, the T-gate complexity of a full $\mathbb{F}_{2^m}$-multiplication is quite substantial.[1]

Multiplication by a constant: Fix $\gamma \in \mathbb{F}_{2^m}^*$. As multiplication with γ is $\mathbb{F}_2$-linear, invoking a general multiplier is not necessary. Instead, we can realize multiplication by γ as a matrix-vector multiplication with a suitable non-singular matrix Γ. An LUP-decomposition of Γ immediately yields a depth $2m$ circuit that is comprised of no more than $m^2 + m$ CNOTs. No Toffoli gates are needed.

[1] With the realization of [3], a Toffoli gate can be implemented without ancillae with seven T-gates (or $T^\dagger$-gates) in a circuit that has a T-depth of 3.

Squaring: No dedicated quantum circuit to implement the squaring map $|\alpha\rangle\,|0\rangle \mapsto |\alpha\rangle\,|\alpha^2\rangle$ has been proposed, but as squaring in $\mathbb{F}_{2^m}$ is $\mathbb{F}_2$-linear, it is enough to implement a matrix-vector multiplication in depth $2m$ using no more than $m \cdot (m+1) = m^2 + m$ CNOTs. No Toffoli gates are needed.

Summarizing, among the above mentioned four basic operations, only the general multiplication involves T-gates, and their number unfortunately is quadratic in the extension degree m. In cryptographic applications of elliptic curves, values of $m \geq 160$ are common. Hence, if we can save a general $\mathbb{F}_{2^m}$-multiplication at the expense of some additions, squarings or constant multiplications, this can be of great value for the implementor of a quantum circuit.

So far, our discussion has ignored the inversion operation. Prior to this research, the literature offered only a circuit with a cubic number of gates and quadratic depth [26], making the two representations discussed in the next section seemingly more attractive for inversion. However, in Section 4.1.3 below, we will show that both the cubic gate complexity and the quadratic depth of this operation can be avoided by adapting the inversion technique used in Section 3.2 to the polynomial basis setting.

4.1.2 Gaussian normal basis and ghost-bit basis representations

Aiming for a more efficient inversion algorithm, in Section 3.2 two field representations are considered that differ from the polynomial basis representation just discussed: a *ghost-bit basis* and a *Gaussian normal basis* representation. Technical details can be found in Section 3.2; so below, we restrict to looking at the cost of the relevant arithmetic operations:

Addition: With a Gaussian normal basis, addition can be performed in the same way as with a polynomial basis. If a ghost-bit basis is available, elements in $\mathbb{F}_{2^m}$ are represented with $m+1$ bits, resulting again in two approaches for the addition. One approach is to add $|\alpha\rangle$ to $|\beta\rangle$ yielding one additional CNOT gate

and a depth 1 circuit. The other approach is to add $|\alpha\rangle$ followed by $|\beta\rangle$ to $|0\rangle$ yielding two additional CNOT gates and a depth 2 circuit. Apart from these details, the addition operation is exactly the same as when using a polynomial basis representation.

Multiplication: If a ghost-bit basis is available, the multiplication $\alpha \cdot \beta$ of two field elements $\alpha, \beta \in \mathbb{F}_{2^m}$ can be realized in depth $m+1$ using $(m+1)^2$ Toffoli gates.

With a Gaussian normal basis of type t, a quantum circuit of depth $(t + (t \bmod 2)) \cdot m - 1$ involving $(t + (t \bmod 2)) \cdot m^2 - m$ Toffoli gates is available for multiplying two elements in $\mathbb{F}_{2^m}$.

Multiplication by a constant: Choosing the matrix Γ in accordance with the Gaussian normal basis or the ghost-bit basis, we can proceed as in the case of a polynomial basis. For a Gaussian normal basis this yields a circuit with m^2+m CNOTs, and as a result of the extra bit used in a ghost-bit basis, for the latter we obtain a quantum circuit comprised of $(m+1) \cdot (m+2) = m^2 + 3m + 2$ CNOT gates. No Toffolis are needed.

Squaring: This operation is free, since the square of a field element can be obtained by simply reading the coefficient vector in permuted order. Hence, no gates are required to implement the squaring operation and we require m respectively $m+1$ CNOTs, all operating in parallel, to implement the map $|\alpha\rangle\,|0\rangle \mapsto |\alpha\rangle\,|\alpha^2\rangle$.

Again, in terms of T-gate complexity, multiplication is the dominating operation, and the number of squaring operations in formulae for fast elliptic curve addition can be expected to be quite small. Consequently, using a polynomial basis representation looks preferable, even if the particular extension degree of interest affords a Gaussian normal basis of small type.

However, taking the computation of inverses into account—an operation that occurs in the derivation of a unique representation of a curve point—the situation seems

to become more involved: In Section 3.2 an inversion circuit of depth $\mathrm{O}(m \log_2 m)$ involving $\mathrm{O}(m^2 \log_2 m)$ gates has been presented. Compared to the quadratic depth and cubic gate complexity of the best published inversion circuit using a polynomial basis [26], this looks quite attractive. While [26] builds on Euclid's algorithm, Section 3.2 builds on a classical technique by Itoh and Tsujii [23], which exploits that an efficient squaring algorithm is available. As mentioned, in the case of a Gaussian normal basis or a ghost-bit basis representation, the squaring operations in a quantum circuit are actually free. To overcome the cubic gate complexity and quadratic depth requirements of inversion using a polynomial basis, we discuss the complexity of applying Itoh and Tsujii's algorithm with a polynomial basis in the next section.

4.1.3 Itoh-Tsujii inversion with a polynomial basis representation

Applying the Itoh-Tsujii algorithm as described in Section 2.2.4, we show that even with a polynomial basis, this approach is a very attractive alternative to Euclid's algorithm. Computing a value β_{i+j} from given values β_i, β_j by means of Equation (2.7) involves one multiplication and an exponentiation by a fixed power of 2. As mentioned in Section 4.1.1, the multiplication can be implemented with m^2 Toffolis plus $m^2 - 1$ CNOT gates in a quantum circuit of depth $\mathrm{O}(m)$. Differing from the situation in Section 3.2, the exponentiation with 2^i is not free, but as the map $\xi \mapsto \xi^{2^i}$ is $\mathbb{F}_2$-linear and bijective, we can implement it as a matrix-vector multiplication with a suitable non-singular $m \times m$ matrix having entries in $\mathbb{F}_2$. Thence, using an LUP-decomposition of this matrix, the needed exponentiation can be realized with $m^2 + m$ CNOT gates in depth $2m$. Summarizing, we see that in a polynomial basis representation, one evaluation of Equation (2.7) can be realized in depth $\mathrm{O}(m)$ using m^2 Toffolis and $2m^2 + m - 1$ CNOT gates.

Repeatedly applying Equation (2.7) with $i = j$ to find all of $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$ requires $\lfloor \log_2(m-1) \rfloor - 1$ evaluations of Equation (2.7); i. e., this step can be realized

in depth $\mathrm{O}(m \log_2 m)$ by means of $(\lfloor \log_2(m-1) \rfloor - 1) \cdot m^2$ Toffolis and $\mathrm{O}(m^2 \log_2 m)$ CNOT gates. Using Equation (2.7) to find

$$\beta_{2^{k_1}+2^{k_2}}, \beta_{2^{k_1}+2^{k_2}+2^{k_3}}, \ldots, \beta_{2^{k_1}+2^{k_2}+\cdots+2^{k_{\mathrm{hw}(n-1)}}} (= \beta_{m-1})$$

requires performing $\mathrm{hw}(m-1)-1$ evaluations of Equation (2.7) sequentially, and we obtain a depth of $\mathrm{O}(m \log_2 m)$, involving $(\mathrm{hw}(m-1)-1) \cdot m^2$ Toffolis and $\mathrm{O}(m^2 \log_2 m)$ CNOT gates. Computing $\alpha^{-1} = (\beta_{m-1})^2$ is just a matrix-vector multiplication with a suitable non-singular $m \times m$ matrix, and using an LUP-decomposition of the latter, a quantum circuit with no more than $n^2 + n$ CNOT gates can realize this squaring in depth $2m$.

To "uncompute" ancillae, we run the complete circuit—with exception of the final squaring—"backwards" and obtain the following:

Proposition 4.1.1. *In a polynomial basis representation, α^{-1}, the inverse of an element $\alpha \in \mathbb{F}_{2^m}$, can be computed in depth $\mathrm{O}(m \log_2 m)$ using $2 \cdot (\lfloor \log_2(m-1) \rfloor + \mathrm{hw}(m-1) - 2) \cdot m^2 = \mathrm{O}(m^2 \log_2 m)$ Toffolis and $\mathrm{O}(m^2 \log_2 m)$ CNOT gates. This includes the cost for cleaning up ancillae.*

Remark 4.1.1. *Organizing the computation of β_{m-1} in a tree structure, the circuit depth for this step can be reduced to $\mathrm{O}(m \log_2 \log_2 m)$. However, because we must repeatedly apply Equation (2.7) with $i = j$ to find all of $\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{k_1}}$, the overall depth of the inverter will still be bounded by $\mathrm{O}(m \log_2 m)$.*

Even though the squaring operation is not free, in terms of T-gate complexity, this inverter seems quite competitive to the ones presented in Section 3.2 for ghost-bit and Gaussian normal basis representations. Thus, for the remainder of our discussion on point addition, we assume that a polynomial basis representation of the underlying field $\mathbb{F}_{2^m}$ is used.

4.2 CHOOSING A CURVE REPRESENTATION: THE COST OF ADDING A FIXED POINT

Before looking at the task of implementing a general point addition $P_1 + P_2$, it is worthwhile to consider the special case when $P_1 \neq \mathcal{O} \neq P_2$, $P_1 \neq \pm P_2$, and P_2 is a fixed point. In a discrete logarithm computation as discussed in [26, 30], this is the only case needed; i. e., only the very last case of the addition law in Figure 2.5 needs to be taken into account. Still, when using affine coordinates, the addition law involves an inversion in $\mathbb{F}_{2^m}$ and as indicated by the discussion in Section 4.1, this inversion operation is typically (much) more expensive to implement than addition or multiplication in $\mathbb{F}_{2^m}$. Therefore, relying on a projective formulation of the group law is a natural choice when designing quantum circuits. In projective coordinates, each $(x, y) \in \mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^m}) \setminus \{\mathcal{O}\}$ is represented by a triple $(X, Y, Z) \in \mathbb{F}_{2^m}^3$ such that $X/Z = x$ and $Y/Z = y$, and $\mathcal{O}$ is represented by a triple $(0, Y, 0) \in \mathbb{F}_{2^m}^3$ with $Y \neq 0$. These triples are only unique up to multiplication with a non-zero element in $\mathbb{F}_{2^m}$. Maslov et al. [30] exploit this freedom to restrict the number of finite field inversion circuits in a discrete logarithm computation. In particular, they observe that as long as such a (non-unique) projective representation is sufficient, the addition of a constant curve point can be realized in linear depth.

To the best of our knowledge, prior to this research no detailed (gate-level) analysis of how to add a fixed point on an elliptic curve has been published. Subsequently we note that—even with a clever implementation of projective coordinates—the T-gate complexity of such a quantum circuit can be reduced substantially by passing to a different curve representation. As a welcome aside, it seems that the circuit depth can be brought down simultaneously.

4.2.1 Mixed addition with projective coordinates

For the fixed point that is to be added, one can assume an affine representation is available leaving no need to handle a general "Z-coordinate" for this operand. So using projective coordinates, a natural (non-trivial) way to implement the addition of a fixed point is to apply the *madd-2008-bl* formulae from [8]: with the curve parameter a_2 as in Equation (2.8) these formulae derive a projective representation (X_3, Y_3, Z_3) of $P_1 + P_2$ with twelve $\mathbb{F}_{2^m}$-multiplications, three of them having one operand fixed (namely, one operand is x_2, y_2 or a_2), seven $\mathbb{F}_{2^m}$-additions, and one squaring.

$$\begin{aligned}
A &= Y_1 + Z_1 \cdot y_2, \quad B = X_1 + Z_1 \cdot x_2, \quad AB = A + B, \\
C &= B^2, \qquad\qquad E = B \cdot C, \qquad\qquad F = (A \cdot AB + a_2 \cdot C) \cdot Z_1 + E, \\
\hline
X_3 &= B \cdot F, \\
Y_3 &= C \cdot (A \cdot X_1 + B \cdot Y_1) + AB \cdot F, \\
Z_3 &= E \cdot Z_1.
\end{aligned}$$

Translating these formulae one by one immediately yields a quantum circuit in which the number of Toffolis, respectively T-gates, is determined by the nine general $\mathbb{F}_{2^m}$-multiplications. To reduce the circuit depth, we can try to parallelize some of the computations. Adding some CNOT gates to create "work copies" of intermediate results, we can enable parallelization without increasing the number of T-gates. To characterize the complexity of the resulting quantum circuit, we write $D_M(m)$ for the depth of an $\mathbb{F}_{2^m}$-multiplier

$$|\alpha\rangle\,|\beta\rangle\,|\xi\rangle \mapsto |\alpha\rangle\,|\beta\rangle\,|\xi + \alpha\beta\rangle\,,$$

and $G_M(m)$ for the number of gates required to implement such a multiplier. Further, we write $D_M^T(m)$ for the T-depth of an $\mathbb{F}_{2^m}$-multiplier and $G_M^T(m)$ for the number of T-gates required to implement such a multiplier. We assume that $D_M(m)$, $G_M(m)$, $D_M^T(m)$, and $G_M^T(m)$ include the cost for cleaning up ancillae. Squaring operations and multiplications by a non-zero constant can be implemented with no more than

$m^2 + m$ CNOT gates in depth $2m$ each. As a functional composition of squarings and multiplications by a non-zero constant can be combined into a single invertible $\mathbb{F}_2$-linear map (through matrix multiplication), any fixed functional composition of squarings and non-zero constant multiplications can be implemented in depth $2m$ with $m^2 + m$ CNOT gates as well.

Proposition 4.2.1. *The point addition*

$$|X_1\rangle |Y_1\rangle |Z_1\rangle |0\rangle |0\rangle |0\rangle \longrightarrow |X_1\rangle |Y_1\rangle |Z_1\rangle |X_3\rangle |Y_3\rangle |Z_3\rangle$$

can be implemented in overall depth $6D_M(m)$ *plus* $8m + \mathrm{O}(1)$ *(the latter accounting for CNOT gates), and T-depth* $6D_M^T(m)$*. Further, a total of* $15G_M(m)$ *gates and* $8m^2 + \mathrm{O}(m)$ *CNOT gates suffice. The total number of T-gates is* $15G_M^T(m)$*. This includes the cost for cleaning up ancillae.*

Here (X_3, Y_3, Z_3) *is some projective representation of* $P_1 + P_2$ *and* $P_2 \in \mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^m})$ *a fixed point, represented with affine coordinates* (x_2, y_2)*.*

Proof: To implement the *madd-2008-bl* formulae we can proceed as follows:

1. Create a "work copy" Z_1' of Z_1 using m CNOT gates, all of which operate in parallel. Then compute $Z_1 \cdot y_2$ and $Z_1' \cdot x_2$ in parallel and store these values in separate ($|0\rangle$-initialized) registers, using $2 \cdot (m^2 + m)$ CNOT gates and depth $2m$.

2. Using $2m$ CNOT gates, all operating in parallel, add Y_1 to $Z_1 \cdot y_2$ and add X_1 to $Z_1' \cdot x_2$, so that those registers now hold A and B respectively. Using $2m$ additional CNOT gates and increasing the circuit depth by 2, we can also store $AB = A + B$ in a new ($|0\rangle$-initialized) register. Moreover, using $2m$ CNOT gates, we can in constant depth provide "work copies" A' of A and B' of B.

3. Using $m^2 + m$ CNOT gates, we can now compute $C = B^2$ in depth $2m$. If $a_2 \neq 0$, with no more than $m^2 + m$ additional CNOT gates we can in parallel determine $a_2 \cdot (B')^2$.

4. Using four multiplication circuits that operate in parallel, we can now compute $E = B \cdot C$, $A \cdot AB$, $A' \cdot X_1$ and $B' \cdot Y_1$ in depth $D_M(m)$, using $4 \cdot G_M(m)$ gates.

5. Next, using $\leq 2m$ CNOT gates that operate in parallel we can add $A' \cdot X_1$ to $B' \cdot Y_1$ and—if $a_2 \neq 0$—$A \cdot AB$ to $a_2 \cdot (B')^2$.

6. With three general $\mathbb{F}_{2^m}$-multipliers we can now compute $(A \cdot AB + a_2 \cdot (B')^2) \cdot Z_1'$, $C \cdot (A' \cdot X_1 + B' \cdot Y_1)$, $Z_3 = E \cdot Z_1$ and store these values in new registers. For this, depth $D_M(m)$ and $3 \cdot G_M(m)$ gates suffice.

7. By adding $(A \cdot AB + a_2 \cdot C) \cdot Z_1'$ to E we obtain the value F in depth 1—involving m CNOT gates. Increasing the depth by 1 and adding m more CNOT gates, we can also create a "work copy" F' of F.

8. Invoking two more multiplication circuits, we can obtain $X_3 = B \cdot F$ and $AB \cdot F'$ in depth $D_M(m)$ with $2 \cdot G_M(m)$ gates.

9. Finally, adding $AB \cdot F'$ to $C \cdot (A' \cdot X_1 + B' \cdot Y_1)$ yields Y_3, and this addition can be realized in depth 1 with m CNOT gates.

To clean up ancillae, the circuit is run backwards, excluding the final multiplications to compute $Z_3 = E \cdot Z_1$, $X_3 = B \cdot F$, the multiplication $C \cdot (A' \cdot X_1 + B' \cdot Y_1)$, and the final addition to compute Y_3. This increases the overall depth by $3D_M(m)$ plus $4m + \mathrm{O}(1)$ (the latter accounting for CNOT gates), the T-depth by $3D_M^T(m)$, the gate count by an additional $6G_M(m)$ plus $4m^2 + \mathrm{O}(m)$ (the latter accounting for CNOT gates), and the T-gate count by $6G_M^T(m)$. □

4.2.2 Mixed addition with a formula by Higuchi and Takagi

Building on earlier work by López and Dahab [28], in [22] Higuchi and Takagi suggest a method to add points on an elliptic curve, which requires fewer multiplications than the *madd-2008-bl* formulae we just discussed. Again, we consider the case of a point

addition $P_1 + P_2$ with $P_1 \neq \pm P_2$ and $P_1 \neq \mathcal{O} \neq P_2$, where P_2 is fixed. Instead of the usual projective coordinates (X, Y, Z) with $x = X/Z$ and $y = Y/Z$ satisfying Equation (2.8), Higuchi and Takagi choose a projective representation with $x = X/Z$ and $y = Y/Z^2$. The corresponding projective formulation of Equation (2.8) then becomes

$$Y^2 + XYZ = X^3Z + a_2X^2Z^2 + a_6Z^4,$$

and the identity element $\mathcal{O}$ is represented by $(X, 0, 0) \in \mathbb{F}_{2^m}^3$ with $X \in \mathbb{F}_{2^m}^*$ arbitrary. For adding a curve point P_1 represented in these coordinates by $(X_1, Y_1, Z_1) \in \mathbb{F}_{2^m}^3$ to a fixed curve point P_2 given by affine coordinates $(x_2, y_2) \in \mathbb{F}_{2^m}^2$, ten $\mathbb{F}_{2^m}$-multiplications along with nine $\mathbb{F}_{2^m}$-additions and three squarings suffice. In two of the ten multiplications one operand is constant:

$$\begin{aligned}
A &= x_2 \cdot Z_1, \quad B_1 = X_1^2, \qquad B_2 = A^2, \\
C &= X_1 + A, \quad D = B_1 + B_2, \quad E = y_2 \cdot Z_1^2, \\
F &= Y_1 + E, \quad G = F \cdot C, \\
\hline
Z_3 &= Z_1 \cdot D, \\
X_3 &= X_1 \cdot (E + B_2) + A \cdot (Y_1 + B_1), \\
Y_3 &= (X_1 \cdot G + Y_1 \cdot D) \cdot D + (G + Z_3) \cdot X_3.
\end{aligned}$$

Allowing an additional squaring, which does not affect the T-gate complexity, the formula for Y_3 can be rewritten as

$$Y_3 = X_1 \cdot D \cdot G + Y_1 \cdot D^2 + (G + Z_3) \cdot X_3. \tag{4.1}$$

This latter formulation is helpful in deriving a quantum circuit with fewer T-gates and a lower depth than the one in Proposition 4.2.1:

Proposition 4.2.2. *The point addition*

$$|X_1\rangle\,|Y_1\rangle\,|Z_1\rangle\,|0\rangle\,|0\rangle\,|0\rangle \longrightarrow |X_1\rangle\,|Y_1\rangle\,|Z_1\rangle\,|X_3\rangle\,|Y_3\rangle\,|Z_3\rangle$$

can be implemented in overall depth $4D_M(m)$ *plus* $4m + \mathrm{O}(1)$ *(the latter being CNOT gates), and* T*-depth* $4D_M^T(m)$*. Further, a total of* $13G_M(m)$ *gates and* $12m^2 + \mathrm{O}(m)$ *CNOT gates suffice. The total number of* T*-gates is* $13G_M^T(m)$*. This includes the cost for cleaning up ancillae.*

Here (X_3, Y_3, Z_3) *is some projective representation of* $P_1 + P_2$ *as used by Higuchi and Takagi, and* P_2 *is a fixed curve point that is represented with affine coordinates* (x_2, y_2)*.*

Proof: To implement the point addition formulae by Higuchi and Takagi we can proceed as follows:

1. Using $4m$ CNOT gates, in depth 2 we create "work copies" X_1' of X_1 as well as Z_1', Z_1'' and Z_1''' of Z_1.

2. With no more than $4 \cdot (m^2 + m)$ CNOT gates, implement the matrix-vector multiplications to compute $A = x_2 \cdot Z_1$, $B_1 = X_1^2$, $B_2 = (x_2 \cdot Z_1')^2$ and $E = y_2 \cdot (Z_1'')^2$ which can be performed in parallel in depth $2m$. To be able to compute D^2, using $2 \cdot (m^2 + m)$ CNOT gates, we also compute in parallel $B_1^2 = (X_1')^4$ and $B_2^2 = (x_2 \cdot Z_1''')^4$.

3. Using $\mathrm{O}(m)$ CNOT gates and constant depth we can now store $C = X_1 + A$, $D = B_1 + B_2$, a "work copy" D' of D, and $F = Y_1 + E$ in separate registers. Moreover, maintaining constant depth and with a linear number of CNOT gates, we can also store $E + B_2$, $Y_1 + B_1$, and $D^2 = B_1^2 + B_2^2$; the latter three values will be used for computing X_3 and Y_3 respectively.

4. Now, using six general $\mathbb{F}_{2^m}$-multipliers, we can in parallel compute $G = F \cdot C$, $Z_3 = Z_1 \cdot D$, $X_1 \cdot (E + B_2)$, $A \cdot (Y_1 + B_1)$, $X_1' \cdot D'$, and $Y_1 \cdot D^2$. For this, $6 \cdot G_M(m)$ gates and depth $D_M(m)$ suffice.

5. At this point, $\mathrm{O}(m)$ CNOT gates and constant depth are adequate to compute

$X_3 = X_1 \cdot (E + B_2) + A \cdot (Y_1 + B_1)$ and $G + Z_3$ and store these values in new registers.

6. With two more multipliers that operate in parallel, $(X_1' \cdot D') \cdot G$ and $(G + Z_3) \cdot X_3$ can be computed. Using $2 \cdot G_M(m)$ gates, this can be accomplished in depth $D_M(m)$.

7. Finally, using $\mathrm{O}(m)$ CNOT gates and depth 2, with Equation (4.1) we can compute $Y_3 = X_1 \cdot D' \cdot G + Y_1 \cdot D^2 + (G + Z_3) \cdot X_3$.

To clean ancillae, we run the circuit backwards, with the exception of the the final additions to compute Y_3 and X_3 and the multipliers to compute $Z_3 = Z_1 \cdot D$, $(G + Z_3) \cdot X_3$ and $A \cdot (Y_1 + B_1)$. This increases the overall depth by $2D_M(m)$ plus $2m + \mathrm{O}(1)$ (the latter accounting for CNOT gates), the T-depth by $2D_M^T(m)$, the gate count by an additional $5G_M(m)$ plus $6m^2 + \mathrm{O}(m)$ (the latter accounting for CNOT gates), and the T-gate count by $5G_M^T(m)$. □

Comparing Proposition 4.2.1 and Proposition 4.2.2, we see that passing from the usual projective representation to the one used by Higuchi and Takagi results in a significant saving in the total number of T-gates while reducing the circuit depth and T-depth. Hence, replacing the usual projective addition in the quadratic depth solution for the discrete logarithm problem in [30] with the addition discussed in this section is an attractive implementation option.

4.3 IMPLEMENTING A GENERAL POINT ADDITION USING EDWARDS CURVES

In view of the case distinctions in the addition law in Figure 2.5, implementing a quantum circuit that properly handles all cases of a point addition appears to be a somewhat burdensome task: in addition to the "generic case" $P_1 \neq \pm P_2$ (with P_2 not being fixed) and $P_1 \neq \mathcal{O} \neq P_2$, we have to implement a doubling formula ($P_1 = P_2$),

making sure that the identity element is handled properly ($P_1 = -P_2$, $P_1 = \mathcal{O}$ or $P_2 = \mathcal{O}$). It is important to note here that testing the branching conditions in Figure 2.5 comes at a certain cost when working with inversion-free arithmetic as just discussed. With projective coordinates as described in Section 4.2.1, let $(X_1, Y_1, Z_1) \in \mathbb{F}_{2^m}^3$ and $(X_2, Y_2, Z_2) \in \mathbb{F}_{2^m}^3$ be representations of two curve points P_1, P_2 different from the identity. Checking if these two points satisfy

$$\underbrace{X_1/Z_1}_{x_1} = \underbrace{X_2/Z_2}_{x_2} \; (\iff X_1 Z_2 = X_2 Z_1)$$

requires two $\mathbb{F}_{2^m}$-multiplications—not taking into account additional gates that may be needed to clean up ancillae.

Working with a different representation of elliptic curves offers an elegant alternative to dealing with the case distinctions in Figure 2.5: In [9], Bernstein et al. discuss a representation of ordinary elliptic curves over $\mathbb{F}_{2^m}$ which affords a *complete* addition law; i. e., the addition of any two curve points is handled with the very same formula. For $m \geq 3$ (which is especially safe to assume in cryptographic applications), each ordinary elliptic curve is birationally equivalent to such a *complete binary Edwards curve* [9]. Working with the complete binary Edwards curve defined in Section 2.3.3, the identity element of a such a curve is $(0, 0) \in \mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^m})$, and for *any* two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ in $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^m})$, their sum is $P_3 = (x_3, y_3)$ with

$$\begin{aligned} x_3 &= \frac{d_1(x_1 + x_2) + d_2(x_1 + y_1)(x_2 + y_2) + (x_1 + x_1^2)(x_2(y_1 + y_2 + 1) + y_1 y_2)}{d_1 + (x_1 + x_1^2)(x_2 + y_2)} \text{ and} \\ y_3 &= \frac{d_1(y_1 + y_2) + d_2(x_1 + y_1)(x_2 + y_2) + (y_1 + y_1^2)(y_2(x_1 + x_2 + 1) + x_1 x_2)}{d_1 + (y_1 + y_1^2)(x_2 + y_2)}. \end{aligned}$$

Similar to working with a short Weierstrass form, one can pass to projective coordinates to avoid costly inversions. In [9] an explicit addition formula is given to compute a representation (X_3, Y_3, Z_3) of the sum of two points on a complete binary Edwards curve, represented projectively as (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2). The formula involves 21 general multiplications in $\mathbb{F}_{2^m}$, three multiplications by the parameter d_1, one

multiplication by the parameter d_2, 15 additions of $\mathbb{F}_{2^m}$-elements, and one squaring:

$$\begin{aligned}
W_1 &= X_1 + Y_1, & W_2 &= X_2 + Y_2, & A &= X_1 \cdot (X_1 + Z_1), \\
B &= Y_1 \cdot (Y_1 + Z_1), & C &= Z_1 \cdot Z_2, & D &= W_2 \cdot Z_2, \\
E &= d_1 C^2, & H &= (d_1 Z_2 + d_2 W_2) \cdot W_1 \cdot C, & I &= d_1 Z_1 \cdot C, \\
U &= E + A \cdot D, & V &= E + B \cdot D, & S &= U \cdot V, \\
X_3 &= S \cdot Y_1 + (H + X_2 \cdot (I + A \cdot (Y_2 + Z_2))) \cdot V \cdot Z_1, \\
Y_3 &= S \cdot X_1 + (H + Y_2 \cdot (I + B \cdot (X_2 + Z_2))) \cdot U \cdot Z_1, \\
Z_3 &= S \cdot Z_1.
\end{aligned}$$

These formulae can be translated into a quantum circuit for adding arbitrary (variable) curve points:

Proposition 4.3.1. *Denote by (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) projective representations of two (not necessarily distinct) points $P_1, P_2 \in E_{\mathrm{B},d_1,d_2}$. Then the point addition*

$$|X_1\rangle |Y_1\rangle |Z_1\rangle |X_2\rangle |Y_2\rangle |Z_2\rangle |0\rangle |0\rangle |0\rangle \longrightarrow |X_1\rangle |Y_1\rangle |Z_1\rangle |X_2\rangle |Y_2\rangle |Z_2\rangle |X_3\rangle |Y_3\rangle |Z_3\rangle$$

can be implemented in overall depth $5D_M(m) + 4\max(D_M(m), 2m) + \mathrm{O}(1)$, where the argument $2m$ of $\max(\cdot)$ as well as the $\mathrm{O}(1)$ reflect CNOT gates only, and T-depth $9D_M^T(m)$. Further, a total of $39G_M(m)$ plus $8m^2 + \mathrm{O}(m)$ CNOT gates suffice. The total number of T-gates is $39G_M^T(m)$. At this, (X_3, Y_3, Z_3) is a projective representation of $P_1 + P_2$. This includes the cost for cleaning up ancillae.

Proof: To implement the above addition formulae, we proceed as follows:

1. Compute in parallel the values W_1, W_2 as well as $X_1 + Z_1$ and $Y_1 + Z_1$, $Y_2 + Z_2$, and $X_2 + Z_2$ from the input values $X_1, Y_1, Z_1, X_2, Y_2, Z_2$—this can be done in constant depth using $\mathrm{O}(m)$ CNOT gates. In addition we use (depth 1) additions to $|0\rangle$ to create "work copies" W_2' of W_2, Z_1' of Z_1, and Z_2', Z_2'' of Z_2 using $3m$ CNOT gates.

2. Using four general $\mathbb{F}_{2^m}$-multipliers and two matrix vector multiplications, compute in parallel the values A, B, C, $D = W_2 \cdot Z_2'$, along with $d_1 Z_2''$ and $d_2 W_2'$. As all involved multipliers operate on disjoint sets of wires, this can be done in depth $\max(D_M(m), 2m)$ using no more than $4G_M(m)$ plus $2 \cdot (m^2 + m)$ gates (the latter accounting for CNOT gates).

3. Compute (in preparation for computing H) the value $d_1 Z_2'' + d_2 W_2'$ and create "work copies" A' of A, B' of B, C' of C, and D' of D using $\mathrm{O}(m)$ CNOT gates and constant depth.

4. Using five general $\mathbb{F}_{2^m}$-multipliers and two matrix vector multiplications, compute in parallel the values $E = d_1 C^2$, $W_1 \cdot C'$, $A \cdot D$, $B \cdot D'$, $A' \cdot (Y_2 + Z_2)$, $B' \cdot (X_2 + Z_2)$ and $d_1 Z_1$. This can be done in depth $\max(D_M(m), 2m)$ with no more than $5G_M(m)$ plus $2 \cdot (m^2 + m)$ gates (the latter accounting for CNOT gates).

5. Compute U and V and create "work copies" U' of U and V' of V in constant depth using $\mathrm{O}(m)$ CNOT gates.

6. Using five general $\mathbb{F}_{2^m}$-multipliers, find H, I, S, $U' Z_1'$ and $V' Z_1$ using $5G_M(m)$ gates in depth $D_M(m)$.

7. Compute $I + A \cdot (Y_2 + Z_2)$ and $I + B' \cdot (X_2 + Z_2)$ in constant depth using $\mathrm{O}(m)$ CNOT gates. Moreover, generate a "work copy" S' of S using n CNOT gates and maintaining constant depth.

8. Using four general $\mathbb{F}_{2^m}$-multipliers, compute in parallel $X_2 \cdot (I + A \cdot (Y_2 + Z_2))$ and $Y_2 \cdot (I + B(\cdot X_2 + Z_2))$, SX_1 and $S'Y_1$, in depth $D_M(m)$ using $4G_M(m)$ gates.

9. Using $\mathrm{O}(m)$ CNOT gates, compute $H + X_2 \cdot (I + A \cdot (Y_2 + Z_2))$ and $H + Y_2 \cdot (I + B \cdot (X_2 + Z_2))$ in depth 2.

10. Multiply $H + X_2 \cdot (I + A \cdot (Y_2 + Z_2))$ with $V'Z_1$, $H + Y_2 \cdot (I + B \cdot (X_2 + Z_2))$ with $U'Z_1'$, and compute $Z_3 = S \cdot Z_1$. This can be done using $3G_M(m)$ gates in depth $D_M(m)$.

11. Compute X_3 by adding $S'Y_1$ to $(H + X_2 \cdot (I + A \cdot (Y_2 + Z_2))) \cdot V'Z_1$ and Y_3 by adding SX_1 to $(H + Y_2 \cdot (I + B \cdot (X_2 + Z_2))) \cdot U'Z_1'$ in depth 1 using $\mathrm{O}(m)$ CNOT gates.

The above circuit has depth $3D_M(m) + 2\max(D_M(m), 2m) + \mathrm{O}(1)$ with the argument $2m$ of $\max(\cdot)$ as well as the $\mathrm{O}(1)$ originating in CNOT gates. The number of gates is bounded by $21G_M(m)$ plus $4m^2 + \mathrm{O}(m)$ CNOTs. "Uncomputing" auxiliary qubits by running the circuit backwards—with the exception of the multiplications $Z_3 = S \cdot Z_1$, $H + Y_2 \cdot (I + B' \cdot (X_2 + Z_2)) \cdot U'Z_1'$, $H + X_2 \cdot (I + A \cdot (Y_2 + Z_2)) \cdot V'Z_1$, and the final additions to compute X_3 and Y_3—yields the desired bound. □

Making use of the (linear-depth and polynomial-size) multiplication circuits in Section 3.1, for asymptotic purposes we obtain the following corollary from the above proposition.

Corollary 4.3.1. *Two points on an Edwards curve in projective representation can be added in linear depth with a polynomial-size quantum circuit.*

Proof: This follows immediately from the multiplier architectures described in Section 3.1, which have linear depth and involve only a polynomial number of gates.□

CHAPTER 5
A QUANTUM CIRCUIT FOR SERPENT

In this chapter, we discuss a design and the necessary resources to implement a quantum circuit which realizes SERPENT, a finalist block cipher candidate for the Advanced Encryption Standard (AES).

Our design is based on the cipher description given in [4] and is a 32-round SPN operating on a block size of 128 bits. The cipher initiates with a permutation of the plaintext, followed by 32 rounds of key mixing/S-boxes/linear transformations, and ends with a final permutation yielding the cipertext.

5.1 THE KEY SCHEDULE

SERPENT uses a 256-bit key K but can accept smaller key sizes. If the key length is less than 256 bits, it is first padded by adding a '1' followed by a string of '0's to achieve the desired length. Next, the key K must be expanded to 33 round keys $K_0, ..., K_{32}$, each 128-bits, in the following way (which is taken from [4]):

1. Write K as eight 32-bit words $w_{-8}, w_{-7}, ..., w_{-2}, w_{-1}$.

2. Generate 132 intermediate (pre)keys $w_0, ..., w_{131}$ using the recurrence

$$w_r = (w_{r-8} \oplus w_{r-5} \oplus w_{r-3} \oplus w_{r-1} \oplus \phi \oplus r) <<< 11 \tag{5.1}$$

   for $0 \leq r < 132$, where $<<<$ is a rotation operation as described in Section 5.3.1, and ϕ is the fractional part of the golden ratio.

3. Generate 33 round keys K_i, each 128-bits, from the intermediate (pre)keys w_r by running through the S-boxes as in Figure 5.1.

Figure 5.1: Using S-boxes to transform intermediate (pre)keys into round keys

$$
\begin{aligned}
K_0 &:= S_3(w_0, w_1, w_2, w_3) \\
K_1 &:= S_2(w_4, w_5, w_6, w_7) \\
K_2 &:= S_1(w_8, w_9, w_{10}, w_{11}) \\
K_3 &:= S_0(w_{12}, w_{13}, w_{14}, w_{15}) \\
K_4 &:= S_7(w_{16}, w_{17}, w_{18}, w_{19}) \\
&\vdots \\
K_{31} &:= S_4(w_{124}, w_{125}, w_{126}, w_{127}) \\
K_{32} &:= S_3(w_{128}, w_{129}, w_{130}, w_{131})
\end{aligned}
$$

We let B_i and B_{i+1} represent the input and output of the ith round of SERPENT, respectively. Hence, B_0 is the initial input (an initial permutation IP of the plaintext) and B_{32} is the final output (the ciphertext after a final permutation FP, which is the inverse of IP). Both IP and FP can be found in Appendix A of [4]. During all but the last round, $B_i \oplus K_i$ is computed, run through S-box S_i, and run through a linear transformation resulting in the next round's input, B_{i+1}. In the last round, the linear transformation is replaced with an $\oplus$ with K_{32}.

Proposition 5.1.1. *In one round of SERPENT, the generation of intermediate (pre)keys using Equation 5.1 can be realized on a quantum circuit with 512 CNOTs and* $80 + \sum_{r=4i}^{4i+3} hw(r)$ *NOTs, where* i *is the round number and* $hw(r)$ *is the Hamming weight of* r*.*

Proof: Each 32-bit word w_r in Equation 5.1 requires an $\oplus$ with four previously stored words, an $\oplus$ with ϕ as defined above, an $\oplus$ with r, and a rotation by 11. To $\oplus$ with the four previously stored words requires 128 CNOTs. This will be repeated

four times in each round totaling 512 CNOTs. To realize $\oplus$ with ϕ and $\oplus$ with i, we calculate the Hamming weight of each value and we will need this number of NOTs. As $\phi = 10011110001101110111100110111001$, the $hw(\phi) = 20$. This will be repeated four times totaling 80 NOTs. For each round i, we have the $\oplus$ of four consecutive numbers $r = 4i, 4i+1, 4i+2, 4i+3$. Hence, we have a total sum per round of $\sum_{r=4i}^{4i+3} hw(r)$ NOTs. Lastly, we have a rotation by 11, which is free. Hence, the total number of CNOTs and NOTs required are 512 and $80 + \sum_{r=4i}^{4i+3} hw(r)$, respectively. □

In designing our circuit, we will be able to clear and re-use wires for calculating the intermediate (pre)keys. We begin this process between rounds two and three, and then run it before every round.

Proposition 5.1.2. *After round i and before round $i+1$ of SERPENT, wires can be cleared and re-used with 512 CNOTs and the same number of NOTs as needed in Proposition 5.1.1 for calculating (pre)keys in round $i-2$.*

Proof: We can think of this as running the circuit backwards to "undo" operations and set four 32-bit words back to $|0\rangle$. As we are clearing calculations after round i that occurred originally in round $i-2$, it is clear that the same number of CNOTs and NOTs will be needed to "undo" $w_{4(i-2)}, w_{4(i-2)+1}, w_{4(i-2)+2}, w_{4(i-2)+3}$. □

5.2 THE S-BOXES

We make use of the S-boxes $S_0, ..., S_7$, and their inverses, shown in Figure 5.2, which is taken directly from the Appendix in [4].

Each S-box represents a four-bit permutation and we apply 32 of these in parallel to affect 128 bits, or four 32-bit words. Considering NOT, CNOT, and Toffoli gates, we have 28 generators on a group of permutations: 4 NOT, 12 CNOT, and 12 Toffoli. Each of these gates can be seen as a permutation (of order 2) on four bits, and we can

Figure 5.2: S-boxes $S_0, ..., S_7$ and their inverses

S_0	:=	3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12
S_1	:=	15	12	2	7	9	0	5	10	1	11	14	8	6	13	3	4
S_2	:=	8	6	7	9	3	12	10	15	13	3	14	4	0	11	5	2
S_3	:=	0	15	11	8	12	9	6	3	13	1	2	4	10	7	5	14
S_4	:=	1	15	8	3	12	0	11	6	2	5	4	10	9	14	7	13
S_5	:=	15	5	2	11	4	10	9	12	0	3	14	8	13	6	7	1
S_6	:=	7	2	12	5	8	4	6	11	14	9	1	15	13	3	10	0
S_7	:=	1	13	15	0	14	8	2	11	7	4	12	10	9	3	5	6
$InvS_0$	:=	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
$InvS_1$	:=	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0
$InvS_2$	:=	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7
$InvS_3$	:=	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
$InvS_4$	:=	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
$InvS_5$	:=	8	15	2	9	4	1	13	14	11	6	5	3	7	12	10	0
$InvS_6$	:=	15	10	1	13	5	3	6	0	4	9	14	7	2	12	9	11
$InvS_7$	:=	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

reduce the task of expressing the S-box as a quantum circuit to the task of expressing the permutation corresponding to each of the $S_0, \ldots, S_7$ as a word in this particular generating set. According to [17], each S-box can be realized using a maximum of 55 Toffoli, CNOT, and NOT gates.

5.2.1 Permutation parity

Our S-boxes consist of four even and four odd permutations. The even permutations can be done in place while the odd permutations require an additional wire for storing intermediate results [29]. S_0, S_1, S_2, S_6, and their inverses are the even permutations while S_3, S_4, S_5, S_7 and their inverses are the odd permutations.

Proposition 5.2.1. *The passing of four 32-bit words through an even S-box during a round of SERPENT can be realized on a quantum circuit using 128 qubits and at most a combination of 1,760 NOT, CNOT, and Toffoli gates.*

Proof: The following describes operations and resources necessary to pass four 32-bit words through an *even* S-box:

1. Four words of 32-bits each requires 128 qubits.

2. Each S-box takes four bits as input and outputs four bits. Therefore, for a 32-bit word, eight copies of the S-box are needed and these are computed in parallel.

3. As each S-box can be realized using at maximum a combination of 55 NOT, CNOT, and Toffoli gates, 440 total gates will be needed for one 32-bit word.

Hence, we can realize the passing of four 32-bit words through an *even* S-box during a round of SERPENT with a total of 128 qubits and 1,760 NOT, CNOT, and Toffoli gates. □

Proposition 5.2.2. *The passing of four 32-bit words through an odd S-box during a round of SERPENT can be realized on a quantum circuit using 160 qubits, at most a combination of 1,760 NOT, CNOT, and Toffoli gates, and a maximum of 32 NOT gates for clearing out and re-using the additional wire necessary for storing intermediate results.*

Proof: The following describes operations and resources necessary to pass four 32-bit words through an *odd* S-box:

1. Four words of 32-bits each requires 128 qubits.

2. Each S-box takes five bits as input and outputs five bits, with the fifth bit initialized to $|0\rangle$. Therefore, for a 32-bit word, eight copies of the S-box and eight additional wires are needed.

3. As each S-box can be realized using at maximum a combination of 55 NOT, CNOT, and Toffoli gates, 440 total gates will be needed for one 32-bit word.

4. Then, for one 32-bit word, clearing out and re-using the additional 8 wires responsible for storing the intermediate results of each S-box, we need at most eight NOT gates to reset any wire carrying a 1 to 0.

Hence, we can realize the passing of four 32-bit words through an *odd* S-box during a round of SERPENT with a total of 160 qubits, a combination of 1,760 NOT, CNOT, and Toffoli gates, and a maximum of 32 additional NOT gates. □

5.3 THE LINEAR TRANSFORMATION

The four 32-bit words X_0, X_1, X_2, X_3 are mixed by linear transformation following the procedure in Figure 5.3, which is taken from [4]. The linear transformation involves rotate and shift operations, denoted by $<<<$ and $<<$, respectively.

5.3.1 The rotation operation

The rotation operation on a quantum circuit is free as we only need to read the wires in a different order. For any word X_i of length n and $0 \leq i < n$, we describe a rotation by r as

$$x_i \longrightarrow x_{i+r(\bmod n)}$$

Figure 5.3: Mixing four 32-bit words with a linear transformation

$$
\begin{aligned}
X_0, X_1, X_2, X_3 &:= S_i(B_i \oplus K_i) \\
X_0 &:= X_0 <<< 13 \\
X_2 &:= X_2 <<< 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 << 3) \\
X_1 &:= X_1 <<< 1 \\
X_3 &:= X_3 <<< 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 << 7) \\
X_0 &:= X_0 <<< 5 \\
X_2 &:= X_2 <<< 22 \\
B_{i+1} &:= X_0, X_1, X_2, X_3
\end{aligned}
$$

Example 5.3.1 (Rotation)**.** *For a 6-bit word* $X_0 := x_0x_1x_2x_3x_4x_5$*, the rotation* $X_0 <<< 3 := x_3x_4x_5x_0x_1x_2$.

5.3.2 The shift operation

For any word X_i of length n and $0 \leq i < n$, we describe a shift by s as

$$
x_i = \begin{cases} x_{i+s}, \text{ if } i + s < n \\ 0 \quad , \text{otherwise} \end{cases} \tag{5.2}
$$

Hence, the shift operation of each X_i on a quantum circuit can be realized using $i - s$ CNOTs.

Example 5.3.2 (Shift)**.** *For a 6-bit word* $X_0 := x_0x_1x_2x_3x_4x_5$*, the shift* $X_0 << 3 := x_3x_4x_5000$.

Proposition 5.3.1. *In one round of SERPENT, the mixing of four 32-bit words with a linear transformation as in Figure 5.3 can be realized on a quantum circuit using 128 qubits and 246 CNOTs.*

Proof: The following describes operations and resources necessary for one round of a linear transformation operation on each of the four 32-bit words:

1. Four words of 32-bits each requires 128 qubits.

2. X_0 requires a rotation by 13, an $\oplus$ with X_1, an $\oplus$ with X_3, and a rotation by 5. As the rotations are free, the resources necessary are 64 CNOTs.

3. X_1 requires an $\oplus$ with X_0, an $\oplus$ with X_2, and a rotation by 1. As the rotations are free, the necessary resources are 64 CNOTs.

4. X_2 requires a rotation by 3, an $\oplus$ with X_3, and a shift by 7. As the rotations are free, the necessary resources are 57 CNOTs; 32 CNOTs for the $\oplus$ with X_3 and $i - s = 32 - 7 = 25$ CNOTs for the shift by 7.

5. X_3 requires an $\oplus$ with X_2, a shift by 3, a rotation by 7, and a rotation by 22. As the rotations are free, the necessary resources are 61 CNOTs; 32 CNOTs for the $\oplus$ with X_2 and $i - s = 32 - 3 = 29$ CNOTs for the shift by 3.

Hence, we can realize one round of a linear transformation with a total of 128 qubits and 246 CNOTs. □

5.4 THE SERPENT QUANTUM CIRCUIT

We now give the resource estimates for this quantum circuit.

Proposition 5.4.1. *The encryption of 128-bits*

$$|PT\rangle \longrightarrow |CT\rangle$$

where PT represents plaintext and CT represents the resulting ciphertext, can be implemented using:

1. *800 qubits*
2. *Linear computations: 45,642 CNOTs and 6,192 NOTs*
3. *Non-Linear (S-box) computations: a maximum combination of 172,480 gates and 1,600 NOTs*

This includes the cost of cleaning up ancillae.

Proof: Our quantum circuit for SERPENT takes as input 128-bit plaintext PT, 256-bit key K, and 416 qubits initialized to $|0\rangle$ (384 for use in calculating the intermediate (pre)keys and 32 for the odd S-boxes). To implement the above encryption circuit, we proceed as follows:

1. The plaintext PT is run through an initial permutation IP which is free as we just read the circuit from different wires.
2. For all 32 rounds, (pre)keys are determined as described in Section 5.1, Proposition 5.1.1, and this requires $512 \cdot 32 = 16,384$ CNOTs and $3,008$ NOTs.
3. For all 32 rounds, the round key K_i is calculated by running (pre)keys through an S-box, as in Figure 5.1, and by Proposition 5.2.1 and Proposition 5.2.2 this requires a maximum combination of $3520 \cdot 32 = 112,640$ gates. The maximum number of NOTs is determined by whether the S-box is even or odd. If it is odd, a maximum of 32 NOTs is required for each S-box. Using Figure 5.4, there are 32 odd S-boxes requiring a maximum of $32 \cdot 32 = 1,024$ NOTs.

4. For all 32 rounds, $B_i \oplus K_i$ is calculated, where B_i is the input for round i, and then run through S-box S_i. $B_i \oplus K_i$ requires 128 CNOTS each round for a total of $128 \cdot 32 = 4,096$ CNOTs. By Proposition 5.2.1 and Proposition 5.2.2, the S-box requires a maximum combination of $1,760 \cdot 32 = 56,320$ gates. The maximum number of NOTs is determined by whether the S-box is even or odd. If it is odd, a maximum of 32 NOTs is required for each S-box. Using Figure 5.4, there are 16 odd S-boxes requiring a maximum of $16 \cdot 32 = 512$ NOTs.

5. For the first 31 rounds, $S_i(B_i \oplus K_i)$ is mixed by a linear transformation as described in Figure 5.3, Proposition 5.3.1, and this requires $246 \cdot 31 = 7,626$ CNOTs.

6. Starting with the third round, intermediate (pre)keys must be cleared out so ancillae can be re-used. Through the 32^{nd} round, by Proposition 5.1.2 this requires $512 \cdot 30 = 15,360$ CNOTs and 2,804 NOTs.

7. After 32 rounds, in what we call round 33, we complete one final (pre)key requiring 512 CNOTs and 88 NOTs. Then, one final round key K_{32} is needed which means one final (odd) S-box requiring a maximum combination of 1,760 gates and 32 NOTs. In place of the linear transformation, we calculate $B_{32} \oplus K_{32}$ with 128 CNOTs.

8. For cleaning the ancillae, we first run the inverse of the final (odd) S-box used for a maximum combination of 1,760 gates and 32 NOTs per Proposition 5.2.2 and then clear the last three rounds of (pre)keys using $512 \cdot 3 = 1,536$ CNOTs and 292 NOTs per Proposition 5.1.2.

9. Last, we run the output B_{32} through a final permutation which is free as we just read the circuit from different wires.

Hence, we can realize SERPENT as a quantum circuit using 800 qubits, 45,642 CNOTs and 6,192 NOTs for linear computations and a maximum combination of 172,480 gates and 1,600 NOTs for non-linear computations. □

Figure 5.4: S-box parity for 32 rounds of SERPENT. With O = Odd and E = Even, from left to right, we list the S-box associated with K_i, then the S-box associated with $B_i \oplus K_i$, and finally the inverse of the S-box associated with K_i.

1	OEO	2	EEE	3	EEE
4	EOE	5	OOO	6	EOE
7	OEO	8	OOO	9	OEO
10	EEE	11	EEE	12	EOE
13	OOO	14	EOE	15	OEO
16	OOO	17	OEO	18	EEE
19	EEE	20	EOE	21	OOO
22	EOE	23	OEO	24	OOO
25	OEO	26	EEE	27	EEE
28	EOE	29	OOO	30	EOE
31	OEO	32	OOO		

CHAPTER 6
COMPARISONS AND CONCLUSIONS

The discussion in Chapter 3 demonstrates that the use of representations of finite fields, other than a polynomial basis, can enable efficient and elegant quantum circuits for realizing binary finite field arithmetic. Table 6.1 gives a brief asymptotic comparison of the circuit depth of the representations discussed in comparison to a polynomial basis representation. For a Gaussian normal basis representation the exact depth increases when the type t gets larger, but for cryptographic purposes a value of $t = 10$ is already unusually high, and small values like $t = 2$ or $t = 4$ are more typical; here, we consider t as a (small) constant.

Table 6.1: Circuit depth of $\mathbb{F}_{2^m}$-operations for different representations

	Addition	Mult.	Inversion
polynomial basis [7, 26, 31]	O(1)	O(m)	ext. Euclidean alg.: O(m^2)
ghost-bit basis	O(1)	O(m)	Itoh-Tsujii alg.: O($m \log(m)$)
Gaussian normal basis	O(1)	O(m)	Itoh-Tsujii alg.: O($m \log(m)$)

Table 6.2 gives an asymptotic comparison for the number of gates involved. Again, for Gaussian normal bases we consider the type t as a (small) constant.

Overall, a main feature of the presentations considered here is the convenient implementation of inversion in $\mathbb{F}_{2^m}$: having available a "free" squaring operation, the discussed technique by Itoh and Tsujii offers a viable alternative to Euclid's algorithm. It appears worthwhile to further explore the potential of different finite field

Table 6.2: Number of gates when implementing $\mathbb{F}_{2^m}$-operations for different representations

	Addition	Multiplication	Inversion
polynomial basis [7, 26, 31]	$\mathrm{O}(m)$	$\mathrm{O}(m^2)$	$\mathrm{O}(m^3)$
ghost-bit basis	$\mathrm{O}(m)$	$\mathrm{O}(m^2)$	$\mathrm{O}(m^2 \log(m))$
Gaussian normal basis	$\mathrm{O}(m)$	$\mathrm{O}(m^2)$	$\mathrm{O}(m^2 \log(m))$

representations for deriving quantum circuits that can, e. g., be used in connection with Shor's algorithm. From a cryptographic point of view, binary fields certainly play a prominent role, but, e.g., the discussion of *Optimal Extension Fields* by Bailey and Paar [5] illustrates that finite fields of larger characteristic are of cryptographic interest as well, as they can facilitate efficient (classical) implementations. Exploring different representations of finite fields with odd characteristic appears to be a worthwhile endeavor for future work.

The circuits for binary elliptic curve arithmetic we have presented in Chapter 4 are not "optimal" yet, but they gave ample evidence that incorporating results from the classic elliptic curve literature in quantum circuit design is worthwhile: it is possible to bring down the number of T-gates that need to be protected against errors and it is possible to reduce the overall circuit depth and T-depth. Our results have stimulated follow-up work on the design of efficient quantum circuits for elliptic curve arithmetic in [11]. For adequately evaluating the cryptanalytic potential of quantum computers, this appears to be a fruitful and important research avenue.

The circuit and resource estimates for SERPENT we have presented in Chapter 5 give an idea of the complexity needed to perform a key attack using Grover's algorithm. We hope to generate interest in follow-up work targeting the exact numbers and types of gates needed for implementing the non-linear S-box section of the circuit. Of specific interest, an exact count of T-gates needed and the resulting T-depth of the circuit would be useful.

In this thesis, we have designed circuits which we believe make significant contributions to both asymmetric and symmetric cryptography.

BIBLIOGRAPHY

[1] Brittanney Amento, Martin Rötteler, and Rainer Steinwandt. Efficient quantum circuits for binary elliptic curve arithmetic: reducing t-gate complexity. *Quantum Information & Computation*, 13(7-8):631–644, 2013.

[2] Brittanney Amento, Martin Rötteler, and Rainer Steinwandt. Quantum binary field inversion: improved circuit depth via choice of basis representation. *Quantum Information & Computation*, 13(1-2):116–134, 2013.

[3] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 32(6):818–830, June 2013.

[4] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the advanced encryption standard.

[5] D. V. Bailey and C. Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 472–485. Springer, 1998.

[6] J. N. de Beaudrap, R. Cleve, and J. Watrous. Sharp Quantum versus Classical Query Complexity Separations. *Algorithmica*, 34(4):449–461, 2002.

[7] S. Beauregard, G. Brassard, and J. M. Fernandez. Quantum Arithmetic on Galois Fields. arXiv:quant-ph/0301163v1, January 2003. Available at `http://arxiv.org/abs/quant-ph/0301163v1`.

[8] D. J. Bernstein and T. Lange. Explicit-formulas database. `http://www.hyperelliptic.org/EFD/index.html`.

[9] D. J. Bernstein, T. Lange, and R. R. Farashahi. Binary Edwards Curves. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 244–265. International Association for Cryptologic Research, Springer, 2008.

[10] S. Bravyi and A. Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A*, 71:022316, 2005.

[11] Parshuram Budhathoki and Rainer Steinwandt. Automatic synthesis of quantum circuits for point addition on ordinary binary elliptic curves. *Quantum Information Processing*, 14(1):201–216, 2015.

[12] A. M. Childs, L. J. Schulman, and U. V. Vazirani. Quantum algorithms for hidden nonlinear structures. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 395–404. IEEE Computer Society, 2007.

[13] R. Dahab, D. Hankerson, F. Hu, M. Long, J. López, and A. Menezes. Software Multiplication Using Gaussian Normal Bases. *IEEE Transactions on Computers*, 55(8), 2006.

[14] W. van Dam, S. Hallgren, and L. Ip. Quantum algorithms for some hidden shift problems. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 489–498, 2003. Available at `http://arxiv.org/abs/quant-ph/0211140v1`.

[15] A. G. Fowler, A. M. Stephens, and P. Groszkowski. High threshold universal quantum computation on the surface code. *Phys. Rev. A*, 80:052312, 2009.

[16] W. Geiselmann and H. Lukhaub. Redundant Representation of Finite Fields. In K. Kim, editor, *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 2001.

[17] M. Grassl. private communication, 2016.

[18] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to AES: quantum resource estimates. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, pages 29–43, 2016.

[19] Jozef Gruska. *Quantum Computing*. McGraw-Hill International (UK) Limited, 1999.

[20] J. Guajardo. Itoh-Tsujii Inversion Algorithm. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 650–653. Springer, second edition, 2011.

[21] S. Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, pages 653–658, 2002.

[22] A. Higuchi and N. Takagi. A fast addition algorithm for elliptic curve arithmetic using projective coordinates. *Information Processing Letters*, 76:101–103, 2000.

[23] T. Itoh and S. Tsujii. Structure of parallel multipliers for a class of fields $GF(2^m)$. *Information and Computation*, 83:21–40, 1989.

[24] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[25] D. Jungnickel. *Finite Fields: Structure and Arithmetics*. Wissenschaftsverlag, 1993.

[26] P. Kaye and C. Zalka. Optimized quantum implementation of elliptic curve arithmetic over binary fields. arXiv:quant-ph/0407095v1, July 2004. Available at `http://arxiv.org/abs/quant-ph/0407095v1`.

[27] A. Y. Kitaev. Quantum computations: algorithms and error correction. *Russian Math. Surveys*, 52(6):1191–1249, 1997.

[28] J. López and R. Dahab. Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography – SAC'98*, volume 1556 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 1999.

[29] Igor Markov. An introduction to reversible circuits. In *In Proceedings of the 12th International Workshop on Logic and Synthesis*, pages 318–319, 2003.

[30] D. Maslov, J. Mathew, D. Cheung, and D. K. Pradhan. An $O(m^2)$-depth quantum algorithm for the elliptic curve discrete logarithm problem over GF(2^m). *Quantum Information & Computation*, 9(7):610–621, 2009.

[31] D. Maslov, J. Mathew, D. Cheung, and D. K. Pradhan. On the Design and Optimization of a Quantum Polynomial-Time Attack on Elliptic Curve Cryptography. arXiv:0710.1093v2, February 2009. Available at `http://arxiv.org/abs/0710.1093v2`.

[32] E. D. Mastrovito. VLSI designs for multiplication over finite fields $GF(2^m)$. In T. Mora, editor, *Proceedings of the Sixth Symposium on Applied Algebra, Algebraic Algorithms and Error Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 297–309. Springer, 1988.

[33] E. D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Linköping University, Linköping, Sweden, 1991.

[34] C. Moore, D. Rockmore, A. Russell, and L. J. Schulman. The power of strong Fourier Sampling: Quantum Algorithms for Affine Groups and Hidden Shifts. *SIAM Journal on Computing*, 37(3):938–958, 2007.

[35] National Institute of Standards and Technology, Gaithersburg, MD 20899-8900. *FIPS PUB 186-3. Federal Information Processing Standard Publication. Digital Signature Standard (DSS)*, June 2009. Available at `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

[36] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[37] M. Ozols, M. Roetteler, and J. Roland. Quantum rejection sampling. In *Proceedings of the 3rd ACM conference on Innovations in Theoretical Computer Science (ITCS'12)*, pages 290–308, 2012.

[38] B. W. Reichardt. Quantum universality by state distillation. *Quantum Inf. Comput.*, 9:1030–1052, 2009.

[39] A. Reyhani-Masoleh and M. A. Hasan. Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$. *IEEE Transactions on Computers*, 53(8):945–959, 2004.

[40] F. Rodríguez-Henríquez, N. A. Saqib, and N. Cruz-Cortés. A Fast Implementation of Multiplicative Inversion over GF(2^m). In *International Symposium on Information Technology: Coding and Computing (ITCC 2005)*, volume 1, pages 574–579. IEEE Computer Society, 2005.

[41] M. Rötteler. Quantum algorithms for highly non-linear Boolean functions. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 448–457, 2010. Available at `http://arxiv.org/abs/0811.3208v2`.

[42] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[43] J. H. Silverman. Fast Multiplication in Finite Fields GF(2^N). In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 1999.

[44] J. A. Solinas. An Improved Algorithm for Arithmetic on a Family of Elliptic Curves. In B. S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 1997.

[45] N. Takagi, J. Yoshiki, and K. Takagi. A Fast Algorithm for Multiplicative Inversion in $GF(2^m)$ Using Normal Basis. *IEEE Transactions on Computers*, 50(5):394–398, 2001.