Professur für Sichere intelligente Systeme
Professorship of Secure Intelligent Systems

Post-Quantum Cryptography in Zcash: Security Analysis and Migration Strategies

Bachelorarbeit von
Bachelor Thesis of

Hannes Hacker

Prüfer/Supervisor
Prof. Dr.-Ing. Elif Bilge Kavun

Betreuer/Advisors
Shekoufeh Neisarian, M.Sc.

---

23. Juni 2025

June 23, 2025

Contents

Abstract

Mark Horvath [Hor], an analyst for quantum technologies, predicts the arrival of a cryptographically relevant quantum computer in 2034. These quantum computers can potentially solve mathematical problems modern cryptography relies on, which endangers especially applications fundamentally built on cryptography, like blockchain. This thesis proposes post-quantum secure alternatives for cryptographical functions embedded in the blockchain network Zcash [Hop+24] in the form of models and prototypes. We use the concept of Super-singular Elliptic Curves with Unknown Endomorphism Ring (SECUER) 4.2.3 to implement a Multi-Party-Computation (MPC) protocol and the Falcon algorithm [Fou+20] as a substitute for digital signatures, which employ Elliptic Curve Cryptography (ECC) [Wikd]. The benchmarking results 7 show that Falcon is suitable for supplanting algorithms, functional- and performance-wise, similar to the Elliptic Curve Digital Signature Algorithm (ECDSA) 4.1.1. Throughout the thesis, several issues with libraries and implementations have been encountered, manually resolved, and are contributed to the respective repositories: the ECDSA library "python-ecdsa" [tls], the Falcon library "falcon.py" [Pre] and the SECUER protocol implementation [Bas+22b].

Acknowledgments

I would first like to thank my thesis supervisor, Prof. Dr.-Ing. Elif Bilge Kavun, and advisor, Shekoufeh Neisarian, for giving me the freedom to explore this topic on my own while still providing sophisticated guidelines with elaborate correction to achieve a successful result and improving my skills and understanding in scientific projects.

Further, I thank my parents, Mr. and Mrs. Hacker, and my sister, Ms. Hacker, for their constant support, encouragement and facilitating myself to focus and thrive on my passion for blockchain and computer science; without you none of this would have been possible.

List of Figures

List of Tables

1 Introduction

On the 23^{rd} of October 2016, Wilcox *et al.* [Wil16] held a live-streamed event - "the Ceremony" 6.1. This was the beginning of a network called "Zcash" [Hop+24], which describes a blockchain aiming to provide economic freedom and privacy [Foua] by utilizing cryptographic technologies and decentralization in several areas [Hop+24]. Therefore, Zero-Knowledge Proofs (ZKP) control and restrict the amount of promulgated information to accomplish properties like authenticity and integrity. The original authors employed "Zero-knowledge Succinct Non-Interactive Arguments of Knowledge" (zk-SNARKs) 4.2.1 which require this "ceremony" as a trusted setup for the creation of public parameters. They can be compared to a public and private key pair in Public-Key-Cryptography (PKC) 4.1. Wilcox *et al.* [Wil16] constructed this ceremony to ensure security in different interconnected ways. Utilizing Multi-Party Computation (MPC) to compute independent shards of the key pair by different entities and geographic locations, the public key was assembled, and the respective creators deleted the private key shard. These parties, like John Dobbertin, who later exposed himself as Edward Snowden, have either been public the whole time, amicably exposed later on, or are still only known under a pseudonym. Other protection mechanisms included the employment of air-gapped systems or the buying and subsequent destruction of dedicated computers. Finally, the security of this setup can be summarized in one citation [Wil16] by Wilcox

et al. "as long as at least one of the participants successfully deletes their private key shard, then the toxic waste (private key) is impossible for anyone to reconstruct".

It can be construed that the people involved with Zcash are focusing on cutting-edge cryptography, but all of this effort and achievements could expire soon. This date is called the "Q-Day" and sets the moment when a cryptographically relevant quantum computer is available, which could potentially render modern cryptosystems obsolete. A currently known algorithm is the Rivest-Shamir-Adleman (RSA) [RSA78], which is based on the computational infeasibility of the factorization of large prime numbers for classical computers. Shor *et al.* [Sho97] found methods that could be used to solve those problems efficiently, thereby compromising the security of the respective cryptographic systems. The consequences cannot be described completely, but this situation potentially endangers digital communication's confidentiality, authenticity, and integrity for people and other entities. Researchers, other experts, and organizations are working to provide solutions to this problem, labelled as Post-Quantum Cryptography (PQC).

Besides, National Institute of Standards and Technology (NIST), considered as one of the "go-to" organizations for standards in the cryptographic space, features a challenge for research on finding new algorithms for key exchange and digital signatures [ST25] resulting in technologies currently considered secure against the quantum threat. Nevertheless, while this is a promising sign for a post-quantum secure future, it is still a long way to go. Not only key exchange and digital signature but also ZK technologies are considered prone to the computing power of quantum computers, as shown by Hopwood *et al.* [Hop].

2 Related Work

This thesis is based on the official Zcash specification by Hopwood *et al.* [Hop+24] and describes the theoretical and technical parts of the blockchain network. One of its node implementations, "zcashd", is publicly available on GitHub [Comf] and represents the fundamental implementations we investigated.

While the pool of publications intersecting quantum computers and Zcash is very insignificant, we utilize Yang *et al.* [Zeb+24], Kearney *et al.* [KP21] and Hopwood D. [Hop] e.g., displayed in Table 3.1, to recognize the proneness of the employed digital signatures to attacks by quantum computers.

Therefore, we focus on research about PQC 4.1 and specifically on algorithms for digital signatures. For this purpose, we examine the following research. The NIST carried out a competition [ST25] to determine post-quantum secure candidates, and Marchsreiter D. [Mar24] developed and evaluated a post-quantum secure blockchain by using those candidates. Combining these research results and obtaining information on the original Falcon specification [Fou+20] by Fouque *et al.*, we chose the Falcon algorithm for digital signatures to apply in this project. This decision rests not exclusively on Falcon's worthwhile performance, identified by Marchsreiter D. [Mar24], but also on the possibility of utilizing a public key recovery mode, specified in the original publication by Fouque *et al.* [Fou+20]. The usage of this mode is one of the firsts, according to Marchsreiter D.

Further, Kearney *et al.* [KP21] stated that the public parameters created for Zcash [Wil16] are vulnerable to the quantum threat, that leads us to apply the research of Basso *et al.* [Bas+22c] which introduced an MPC protocol that delivers post-quantum secure and statistically provable Proofs of Knowledge (PoK) utilizing SECUER 4.2.3. Other research focuses on the Zero-Knowledge Scale Transparent Arguments of Knowledge (zk-STARKs), which are considered secure against cryptographically relevant quantum computer attacks. Additionally, Arade *et al.* [AP25] describes the mathematical principles and practical usage of zk-STARKs.

Overall, the novelty of this thesis is built on the usage of cutting-edge technology like Falcon with public key recovery mode [Fou+20] and the SECUER protocol [Bas+22c] while focusing on use-cases like Zcash [Hop+24].

3 Motivation & Contributions

This work investigates the proneness of cryptography-based areas in the Zcash blockchain network [Hop+24] to the abilities of cryptographically relevant quantum computers. Zcash is divided into different protocols with different functionality provided by cryptographic technologies. The rising power of quantum computers poses a potential threat to many of those technologies since they build on mathematical problems that are probably not computationally infeasible for a sophisticated quantum computer. Hopwood D. presented at the ZCONVI 2025 [Hop] a Table 3.1 with all cryptographic functions in the Zcash codebase [Comf] considered prone to the quantum threat.

We take this Table 3.1 as a foundation for this research and eventually decided to focus on the following three primitives for a closer look:

- zk-SNARK ceremony [Wil16]
- ECDSA 4.1.1 in the transparent protocol
- EdDSA 7.5 in the Sprout protocol

This work intends to provide thought-provoking impulses by delivering models and prototypes to upgrade these areas to PQC 4.1. While we build this project on the foundation

Table 3.1: All cryptographic functions in Zcash which are prone to quantum computers.

Cryptographic problem	Relied on for	Primitives used
DL on secp256k1	**Transparent:** Spend authentication, BIP 32 non-hardened derivation	Signatures (ECDSA)
DL on Ed25519	**Sprout:** Spend authentication, Privacy	Signatures (EdDSA), encryption (ECIES)
DL on Jubjub	**Sapling:** Spend authentication, Privacy, Diversified address unlinkability	Signatures (RedDSA), encryption (ECIES)
DL on Pallas	**Orchard:** Spend authentication, Privacy, Diversified address unlinkability	Signatures (RedDSA), encryption (ECIES)
AGM on BLS12-381 G	**Sprout** and **Sapling** Balance	Proofs (Goth16)
DL on Vesta	**Orchard** Balance	Proofs(Halo2)

of the Zcash functionality, the results could be used in other systems, too, e.g., in "Bitcoin Core" [Teab], as the transparent protocol is mostly the remains from Zcash's Bitcoin fork [Comf]. After looking at different post-quantum secure alternatives, we decide to investigate the following specific technologies.

The concept of SECUER with a statistically proveable PoK is used and processed in an MPC by Basso *et al.* [Bas+22c]. The publication sketched a ceremony realized by Git repositories that had not yet been implemented. For the digital signatures, EdDSA and ECDSA, the Falcon [Fou+20] by Fouque *et al.* is selected to investigate the suitability as an alternative.

Over the course of this work, we discovered several issues in used technologies:

- Formatting errors while building the C implementation of the SECUER protocol [Bas+22a] on an M3 MacBook chip (used in section "The Ceremony" 6.1)
- Missing public key recovery mode in the selected Python library by Thomas Prest [Pre] like sketched in the Falcon specification [Fou+20] (used in section "Transparent Protocol" 6.2)

- Missing functionality for the hash functions of PyCryptodome [Eij] in the selected Python library for ECDSA and EdDSA [tls] (used in section "Sprout Protocol" 6.3)

These issues are resolved manually, and we attempt to contribute them to the respective Open-Source GitHub repositories while avoiding breaking changes and providing modular source code that can be potentially extended in the future.

Mainly, this thesis produces reusable prototypes and respective benchmark scripts for comparison of the digital signatures and a sample setup for the SECUER MPC protocol [Bas+22c] that could be extended or used as a foundation for other applications. The insight into these technologies can be used for real-world applications and further research. All resulting implementations can be inspected on this GitHub repository.

4 Terms & Technologies

4.1 Cryptography

Cryptography [Wikb] is a measure to provide confidentiality, authenticity, or integrity for digital systems. These properties can be achieved by technologies based on mathematically hard problems assumed to be computationally infeasible for classical computers.

The PKC [Wikh] is an example of a modern cryptographic method. This method creates a key pair for every participating entity. This pair consists of public and private keys, which are coupled mathematically. The public key can be open to the general public, but the private key has to be kept secret. Encryption and digital signatures utilize these principles, further described in the following scenarios.

- Party A wants to send a message to party B over an insecure channel. A intends to encrypt the message by utilizing the above-described key pair. A takes B's public key, encrypts the message, and sends it to B. B can use his private key to decrypt the received message.

- Party A wants to publish some information but wants to ensure everybody can verify that A published this information and no one else. Hereby, a digital signature

is created by A's key pair. A uses his private key to create the digital signature and publishes it with the message. Everybody who knows the public key of A can now verify the digital signature and check if the message is authentic.

PQC [Wikg] uses mathematically hard problems which are assumed to be computationally infeasible for classical and quantum computers. NIST [ST25] published and tested algorithms for encryption and digital signatures assumed to be post-quantum secure.

4.1.1 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA [Wikk] is an algorithm for creating and verifying digital signatures. Derivable from its name, it takes advantage of ECC [Wikd], and finite fields. The dependency on the discrete logarithm problem renders ECDSA prone to attacks by a cryptographically relevant quantum computer.

Edwards-curve Digital Signature Algorithm (EdDSA) [Wikk] is another algorithm that provides digital signatures on elliptic curves. Although there are some similarities with ECDSA, its properties and construction vary. Examples include that EdDSA uses special elliptic curves, called "twisted Edwards curves," and lacks a public-key recovery ability, in contrast to ECDSA.

4.1.2 Hashing Algorithms

Hashing algorithms [21] are mathematical functions taking an input x and returning an output y with a fixed size. The resulting outputs have different associated input values but no effective measures to reverse the map. This property, labeled as "pre-image resistance", obtains the denomination "one-way functions".

Figure 4.1: Illustration of a Σ protocol

4.2 Zero-Knowledge (ZK)

ZKPs [Wik25b] are cryptographic protocols to prove the correctness of statements without revealing unnecessary information. While different forms with different properties and constructions of these proofs exist, the Σ protocol in Fig. 4.1 can be considered intuitive to understand.

4.2.1 Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs)

zk-SNARKs' [Foub] constructions discard the interaction necessary in the Σ protocol in Fig. 4.1. Therefore, the prover creates the proof and sends it directly to the verifier to be accepted or declined. Another special property of zk-SNARKs is "succinctness," which qualifies the proof as short.

4.2.2 The Ceremony

"The Ceremony" [Foub] creates a reference string, shared between prover and verifier, that constitutes the public parameters for the protocol. Compromised but successfully

verified proofs could occur in case of an access to the underlying randomness. In contrast, Halo2 [Come] resembles a ZKP without the necessity of a trusted setup [Foub].

4.2.3 Super-singular Elliptic Curves with Unknown Endomorphism Ring (SECUER) protocol

Basso *et al.* [Bas+22c] created a protocol for the distributed generation of SECUER, vertices in an isogeny graph for the construction of PoK and a type of elliptic curves. With each of n parties creating a proof, the protocol finishes successfully in a SECUER tip curve if at least one party was honest. Basso *et al.* [Bas+22c] provided tools for proof creation and verification [Bas+22a] to be employed, such as in the suggested real-world application 6.1.

4.3 Blockchain

Blockchain [Wik25a] is a term coined for a Distributed Ledger Technology (DLT), which is considered decentralized, distributed, and immutable. The data is stored in blocks chained together by sequential hashes. Fundamentally structured in a Peer-to-Peer (P2P) architecture, the peers contain a copy of this chain of blocks (or "ledger") utilized in the consensus algorithm to agree on one version. These blocks are created by solving cryptography challenges like described for Bitcoin 4.3.2 and digital signatures 4.1 are utilized to provide authenticity for user actions.

4.3.1 Zcash

Zcash [Hop+24], forked from Bitcoin 4.3.2 and gone live in 2016 [Wil16], is a blockchain 4.3 employing different protocols. The transparent protocol is pseudo-anonymous like Bitcoin 4.3.2, but Sprout, Sapling, and Orchard are using ZK technologies 4.2 like zk-SNARKs 4.2.1. These "shielded" protocols use ZK to provide privacy for the participants [Fouc]. "zcashd" [Comf] is a node implementation of this blockchain 4.3.

4.3.2 Bitcoin

Bitcoin was released on the 1^{st} of October 2008 by the unknown entity "Satoshi Nakamoto" as a white paper [Nak18]. Classified as a blockchain network, it aimed to diminish double-spending but evolved into the current, most valuable cryptocurrency [Coi]. It specified Proof-of-Work (PoW) as its cryptographical challenge to solve 4.3, which denotes the nodes have to compete against each other to update an additional value of the upcoming block. To continue the hash-based chain, the hash value of the latest block is included in the next one to be hashed over again. The additional value (or "nonce") must be changed until the current block's hash starts with a specific number of *zeros* to be valid. The validity of this process rests upon the "pre-image resistance" 4.1.2 and the forced use of CPU power to solve this challenge. "Bitcoin Core" [Teab] is a node implementation of this blockchain 4.3.

5 Methods

In the "Motivation" 3 section, we point out that we focus on the ceremony 6.1 for zk-SNARKs 4.2.1 and on the digital signatures ECDSA 4.1.1 and EdDSA 4.1.1. We aim to use a similar methodology for each focus area described below. This work was created on a "Ubuntu Server 24.04.2 ARM" virtual machine running on a MacBook Air with a M3 chip (Apple Silicon).

We start researching the usage of the respective focus area in the Zcash environment. This results in obtaining information on the libraries and functions implemented in "zcashd" [Comf] and understanding the ceremony for the zk-SNARKs' [Wil16] public parameters.

After that, we look for post-quantum secure alternatives in these areas like the Python library implementing Falcon by Prest T. [Pre] or the SECUER protocol by Basso *et al.* [Bas+22c] which proposes a ceremony potentially being utilized as public parameters in future ZK 4.2 technologies. To find an appropriate alternative digital signature algorithm, this publication inspects the winners of the NIST competition [ST25] to find tested and accepted algorithms that provide sufficient security and usability in PQC.

Finally, we build standalone prototypes on these technologies, which hold all the necessary functions for displaying the usage in Zcash [Hop+24]. Subsequent tests verify the

correct functionality of the implementation and provide information on performance for comparison. For the prototypes of the digital signatures, we look for Python libraries, too, since we decided to go all the way with this programming language.

6 Implementation

6.1 The Ceremony

The ceremony 6.1, which was held in 2016 for the "Sprout 1.0" protocol of the Zcash blockchain, created the public parameters for zk-SNARKs 4.2.1. Therefore, Wilcox *et al.* [Wil16] created and executed an MPC protocol, besides other physical and privacy-based measures. Six parties participated in the protocol, and every party was responsible for creating a shard of the private and public keys. The public key shards were assembled at the end, but the shards of the private key had to be deleted by the respective party. As long as one party honestly deleted its private key shard, the protocol was successful, and the private key cannot be assembled afterward.

Not all zk-SNARKs [Wikf] are assumed to be secure against quantum computer attacks. However, Basso *et al.* [Bas+22c] proposed an MPC protocol utilizing SECUER 4.2.3 with similar properties, such as only one party has to be honest. SECUER is considered post-quantum secure and could be used in future ZK systems while keeping the resistance against a cryptographically relevant quantum computer. The proposition contains information on a feasible Git structure, which has been realized and extended Fig. 6.1 and is detailed in the following sections.

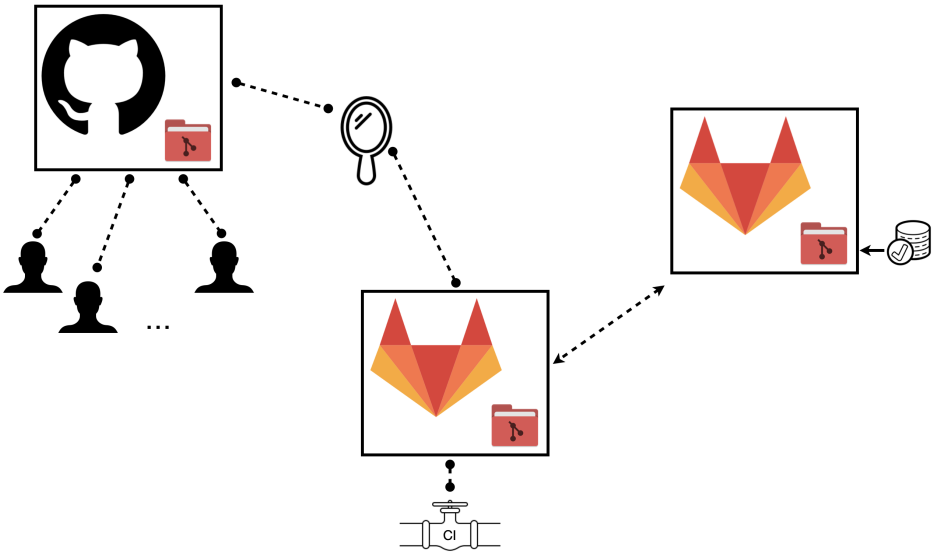

Figure 6.1: Architecture of the Multi-Party SECUER Protocol

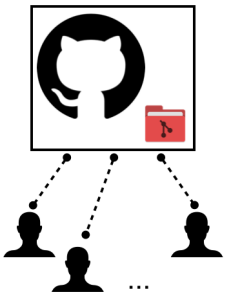

Figure 6.2: GitHub repository for user access

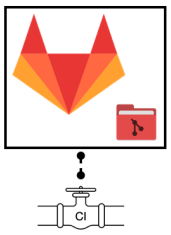

Figure 6.3: Gitlab repository for CI pipeline

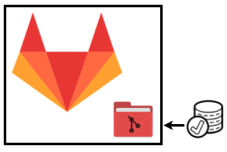

Figure 6.4: Gitlab repository for secure storage

6.1.1 Participants' Access on GitHub

The GitHub repository in Fig. 6.2 represents the access point for the protocol participants. It contains the tools [Bas+22a] provided by Basso *et al.* [Bas+22c] for proof creation and verification to enable the participants for honest contribution. Since this project was done on a MacBook Air with the M3 chip, the option "ARCH=M1" was included to build the tools by CMake [Wika] for the ARM architecture. The resulting formatting errors are fixed locally and contributed to the original repository [Bas+22a].

Three new directories are added to the repository:

- *curves*: holds all verified curves
- *proofs*: holds all verified proofs
- *new_additions*: holds participant's input

Based on Basso *et al.* [Bas+22c], we constitute that the first proof and respective curve 4.2.3 must be placed by the initiator of the protocol into the *curves* and *proofs* directory. The j-invariant (i.e. *–initial* option of the proving tool) or another curve can be selected for this cause. This repository has to be configured to only allow contribution to the main branch by pull requests which trigger the Continuous Integration (CI) pipeline on the mirrored Gitlab repository 6.1.2.

6.1.2 CI Pipeline on Gitlab

The mirroring Gitlab repository in Fig. 6.3 runs a CI pipeline, triggered by pull requests on the GitHub repository in Fig. 6.1 to check the protocol properties specified by Basso *et al.* [Bas+22c] and other security properties specified e.g., in Section 6.1.

The pipeline consists of these scripts written in Bash and stored in the third Gitlab repository 6.1.3 in Fig. 6.4:

1. Check if the directory structure is correct and none of the already verified directories have been changed manually
2. Check if the metadata of the new additions is correct
3. Check if the provided proof and curve belong together and change the GitHub repository accordingly

The pipeline sequentially executes these jobs respective to the stages: *structure*, *new_additions*, and *verification*, and is configured in `.gitlab-ci.yml`. The verification is displayed in the code sample below.

```
check_verify:
  stage: verification
  before_script:
    - apk add --no-cache --upgrade git
    - apk add --no-cache --upgrade make
    - apk add --no-cache --upgrade clang
  image: alpine:latest
  rules:
    # Stage will only be executed in pull request
    - if: $CI_PIPELINE_SOURCE == "external_pull_request_event"
  script:
    - git clone https://oauth2:${AUTOMATION_ACCESS_TOKEN}
      @git.fim.uni-passau.de/hacker/secuer-ceremony-secure.git
    - git clone https://oauth2:${ACCESS_GITHUB_TOKEN}
```

```
      @github.com/zekure-hacker/SECUER_ceremony.git
    - sh secuer-ceremony-secure/test_verify.sh
```

This repository provides the CI functionality, connects the other repositories, and stores the access token in variables. This decoupling enables us to manage the access control of all repositories separately. The GitHub repository 6.1.1 could be accessible for the relevant group of participants, and this repository can be set up to be accessible for entities responsible for the CI functionality.

6.1.3 Secure Value Storage on Gitlab

The last repository in Fig. 6.4 acts as a storage of restricted resources, which extends the decoupling of the access control 6.1.2. Hash values 4.1.2 over the directories *curves* and *proofs* of the GitHub repository 6.1.1 are included to track the latest verified state and signal manual changes as well as a trusted instance of the verification tool 4.2.3 for a valid verification check 6.1.3. Besides, this thesis implements three scripts to power the stages of the CI pipeline 6.1.2 and contains them in this repository.

Script for checking the directories' structure

This script checks if each of the directories described 6.1.1 exists in the pull request like:

```
if [ ! -e "c-impl/curves" ]; then
  exit 1;
fi
```

The above-mentioned hash values are accessed by the script for the integrity of the latest verified *curves* and *proofs* in the GitHub repository 6.1.1. These values are created by consecutive applications of a hashing software which employs "SHA256" [Wiki] and updated whenever there is a new verified state 6.1.3.

```
hash_curves_input=$(sha256sum c-impl/curves/* | sha256sum | head -c64)
hash_curves_reference=$(cat secuer-ceremony-secure/hashes/curves.txt)

if [ ! $hash_curves_input = $hash_curves_reference ]; then
  exit 2;
fi
```

Script for checking the validity of the additions

This script checks the number and denomination of the committed files and starting point of the provided isogeny walk [Bas+22c]. Only *none* or two newly added files are allowed, and file names have to follow these rules: If the last valid curve is called `curve2.txt`, then the new addition has to be named `curve3.txt`. Otherwise, the pull request will be declined. This is repeated on the proof file, too.

The `curve` file contains one line for the start and one line for the end of the isogeny walk 4.2.3, and therefore, the script checks if the committed file starts on the last verified curve, hold in the GitHub repository 6.1.1.

```
tip_curve_name=$(ls c-impl/curves | tail -n1)
addition_starting_curve=$(cat c-impl/new_additions/$curve_additions
                                | head -n1)
current_tip_curve=$(cat c-impl/curves/$tip_curve_name | tail -n1)
```

```
if [ $addition_starting_curve = $current_tip_curve ]; then
  exit 1;
fi
```

Script for verifying the added proof and curve

This script checks if newly added proof verifies the respective curve using the trusted instance of the verification tool 6.1.3.

```
./secuer-ceremony-secure/c-impl/verify_434
    < SECUER_ceremony/c-impl/new_additions/$proof_additions
    > secuer-ceremony-secure/testcurve.txt

input_file="SECUER_ceremony/c-impl/new_additions/$curve_additions"
input_hash=$(sha256sum "$input_file" | head -c64)
reference_file="secuer-ceremony-secure/testcurve.txt"
reference_hash=$(sha256sum "$reference_file" | head -c64)
rm secuer-ceremony-secure/testcurve.txt
if [ ! $input_hash = $reference_hash ]; then
  exit 1
fi
```

If the verification is successful, the *new_additions* directory 6.1.1 is cleared, the verified proof and curve are added to the respective directories 6.1.1 and the hash values for the validity of the directories in the GitHub repository 6.1.3 are updated accordingly.

6.2 Transparent Protocol

Zcash [Hop+24] maintains one transparent protocol which remains from the "Bitcoin Core" fork [Wikl] and three shielded protocols which employ privacy-enhancing technologies like ZK [Wik25b]. We investigate the implementation of the transparent protocol by looking at the codebase [Comf] and found `key.cpp` [Comb] and `pubkey.cpp` [Comd] holds all the necessary functions for the ECDSA 4.1.1 digital signatures. It shows that the library "libsecp256k1" [Comc] is deployed, already hinting at the relevant ECDSA parameter "secp256k1". The mentioned code files reveal the following functions:

- *secp256k1_ec_pubkey_create*: the creation of the public key and the key pair
- *secp256k1_ecdsa_sign*: the creation of a signature
- *secp256k1_ecdsa_sign_recoverable*: the creation of a signature with public key recovery functionality
- *secp256k1_ecdsa_recover*: recovery of the public key of a signature
- *secp256k1_ecdsa_verify*: verification of signature

The shown list strips down the functionality of the code files to the fundamental usage of the respective library by focusing on the core elements and disregarding encoding or parsing functions.

In this thesis, it is decided to use the "python-ecdsa" [tls] library to depict the functions shown above, stay consistent in terms of programming languages, and help with comparability e.g., in Section 7. The prototype below describes wrapper functions employing the "python-ecdsa" library.

```
import ecdsa
private = PrivateKey()
public = PublicKey(private.key)
```

This code generates a public/private key pair of ECDSA 4.1.1.

```
private.sign(message)
```

```
public.verify(signature, message)
```

Signing/verifying a message integrates the same library function in both wrapper functions designed for signing/verifying and signing/verifying with recovery because there is no difference in ECDSA, unlike Falcon.

```
ecdsa.VerifyingKey.from_public_key_recovery(
    signature, message, curve=ecdsa.SECP256k1
    )
```

This codes recovers the public key out of an ECDSA signature with no additional information.

These functions form a model of the ECDSA functionality in Zcash by implementing functions for key pair generation, signing and signing recoverable, verifying and verifying recoverable, and public key recovery. Changing the ECDSA library to the Falcon implementation of Prest T. [Pre] generates a model of a respective Falcon functionality to be compared e.g., in Section 7.

The information on the employed ECDSA 4.1.1 library "libsecp256k1" [Comc] makes use of this recovery functionality. Therefore, it is intriguing to keep this ability since it

aids blockchain networks in reducing the size of transactions by retaining the public key that can be recovered directly from the signature. For a post-quantum secure alternative to ECDSA, we decide to go with the Falcon algorithm [Fou+20] in its 512-bit version by considering Marchsreiter D. [Mar24] and the sketched-out description of a public key recovery mode in the Falcon specification [Fou+20].

We elaborated the Falcon implementation of Thomas Prest [Pre] as an appropriate library but we faced issues with it because it lacks a public key recovery mode. Hence, we had to manually implement the public key recovery mode, which is sketched in the Falcon specification [Fou+20] because it is not integrated into the mentioned implementation. A usual Falcon signature [Fou+20] consists of a salt r and a polynomial s_2. In the signing process, a hash value c is created from the message m and the salt r and the private key enables to compute the short polynomials s_1 and s_2 such that:

$$s_1 + s_2 * h \equiv c \mod q \tag{6.1}$$

where h labels the public key and q is 12289 describing the modulus of the algorithm. s_1 is not included in the signature because it is recalculated in the verification process by utilizing h and s_2. The public-key recovery mode makes use of the following equation

$$h = s_2^{-1}(HashToPoint(r||m, q, n) - s_1) \tag{6.2}$$

By n being the bits of the secret key, which is 512 in this thesis. This facilitates us to cut out the recalculation of the verify function but demands to add a check of invertibility on s_2 in the sign function and adding s_1 to the signature.

The following code snippet executes the check:

```
neutral_polynomial = [1, 0, 0, ...]
ntt_s2 = ntt(s[1])
ntt_inv_s2 = []

for i in range(0, len(ntt_s2)):
    ntt_inv_s2.append(inv_mod_q[ntt_s2[i]])

inv_s2 = intt(ntt_inv)

if not mul_zq(s[1], inv_s2) == neutral_polynomial:
    continue
```

Existing functions for Number Theoretic Transform (NTT) aid on arithmetics of polynomials and a list of inverses save the respective calculations. A new sign function for recoverability outlines these concepts. The recover function implements the public-key equation 6.2 by using the provided NTT functions for subtracting and dividing. As described above, a new verify function compresses this process to the minimum. The public-key recovery mode 6.2 reduces the total size of the public key and signature by around 15 %. Overall, the added functions are *sign_recoverable*, *verify_recoverable*, and *recover*, which can be plugged into the respective functions of the wrapper model.

6.3 Sprout Protocol

The above-described methodology e.g., in Section 5, is used to create another Python model like seen in the section before 6.2 but on the functionality of the Sprout protocol. Therefore, we looked at the "zcashd" implementation [Comf] again to find functions related to the Ed25519 [Wikc] algorithm. The results are written as a Rust implementation [Coma].

By only implementing these functions:

- *generateKeyPair*
- *sign*
- *verify*

the Ed25519 functions are a subset of the functions e.g., in the Section 6.2, and follow almost the same methodology. To replicate a Python module of this functionality, the same library "python-ecdsa" [tls] is used like in the transparent protocol.

```
private = PrivateKey()
public = PublicKey(private)
```

Generating the key pair is implemented as before, but the private key has to be created by adding its randomness.

```
start = random.randbytes(32)
privkey = eddsa.PrivateKey(eddsa.generator_ed25519, start)

privkey.sign(message)
pubkey.verify(message, signature)
```

The signing and verification functions are similar to the transparent protocol. Ed25519 does not have a public-key recovery mode, which would be an issue in this protocol if it did have this mode because it would compromise the intended privacy feature. The alternative Falcon prototype could be copied e.g., from the Section 6.2, without the recovery features.

7 Results & Discussion

This thesis results in three prototypes of the SECUER distributed generation with Git 6.1 and the ECDSA 6.2 and EdDSA 6.3 models, which can be considered as starting points for research on real-world applications. During this thesis, we resolved issues with the selected libraries and implementations. After aligning them to the original projects, they will be contributed to the respective repositories on GitHub.

Firstly, we faced formatting issues when building the SECUER implementation [Bas+22a] on an Apple M3 architecture. Secondly, the missing public key recovery mode in the Falcon implementation of Prest T. [Pre] e.g., pictured in the Section 6.2. We will write functionality tests for this additional code according to the existing ones in the repository. After adjusting the coding style accordingly, a pull request is created. Lastly, a contribution was created for the ECDSA Python library [tls]. This library was not accommodating the hash functions of the "PyCryptodome" library [Eij], which were necessary for the performance benchmarks below. Prest *et al.* [Pre] used the SHAKE256 [Wikj] algorithm which is of the type XOF [Wike]. PyCryptodome implements three other of those XOF-hash algorithms, observerd in Tables 7.3 or 7.4. Thus, this work adjusted the ECDSA library to include hash functions implemented in PyCryptodome.

We developed benchmarking scripts for the digital signatures prototypes 6 to compare their performance. Starting with the transparent protocol 6.2, the comparison is outlined

Table 7.1: libsecp256k1

Function	Avg (us)
sign	16.40
recover	22.70
verify	21.30

Table 7.2: Transparent ECDSA default SHA-1

Function	Avg (us)
generateKeyPair	3,606.68
sign	3,377.48
signRecoverable	3,405.19
recover	48,688.67
verify	13,440.62
verifyRecoverable	13,483.86

between ECDSA 4.1.1 and Falcon [Fou+20]. Table 7.1 shows the performance of Zcash's ECDSA library [Comc] and Table 7.2 the performance of the Python library [tls] we employed for the respective prototype 6.2. The optimized C-library, which is used in the "zcashd" node codebase [Comf] and also in the "Bitcoin Core" codebase [Teaa] outperforms the Python library "python-ecdsa" [tls] in all metrics which leads to our suggestions to interpret the following results relative to each other and not absolute. This work compares Python implementation to Python implementation by changing the underlying algorithm since implementations in programming languages like C are generally faster than in Python.

Table 7.3 and Table 7.4 hold the performance values of the respective prototypes for the transparent protocol 6.2. The recoverable functions of Table 7.3 and their respective function without recoverability features call the same library functions. They were just included in the prototype to have a consistent table structure. The mentioned tables show Falcon's significantly higher execution time when generating a key pair, which can be neglected because key pairs are only generated once per wallet on blockchain networks. Additionally, Falcon's signing function is slower than that of ECDSA. While this is a

negative point, it can be relativized as digital signatures are only created once by the participants every time they send a transaction. However, all blockchain network nodes verify these signatures to check the authenticity of the transactions and the integrity of the blockchain. Consequently, it can be stated that the verification operation has significantly more impact on the performance than the signing operation since it is executed much more often. Furthermore, "zcashd" [Comf] and "Bitcoin Core" [Teab] are employing the public-key recovery mode; thus, the recover function will be called for every verification to extract and check against the public key.

Taking that into account, one can make the following calculation:

```
time for recoverable signature
+ n * (time for recovering public key from signature
        + verifying the signature with the recovered public key)
```

Using the values from Table 7.3 and Table 7.4, we can see the following.

$$\begin{aligned} 3,436.05 + n * (51,568.12 + 13,688.37) &> 13,039.47 + n * (2,432.30 + 496.24) \\ \Leftrightarrow n * (51,568.12 + 13,688.37) &> 9,603.42 + n * (2,432.30 + 496.24) \end{aligned} \tag{7.1}$$

Even for $n = 1$ (one verification), the ECDSA implementation [tls] is much slower than the Falcon [Pre] implementation, not to mention that Falcon provides post-quantum secure properties. Besides, no significant performance impact was detected between the different hash functions [Wike].

The Sprout protocol 6.3 is using the Ed25519 algorithm 7.5, which does not have public-key recovery mode, however outperforms ECDSA 7.3 and Falcon 7.4 in every metric.

Table 7.3: Transparent ECDSA in us

Function	SHAKE256	TurboSHAKE256	cSHAKE256	KangarooTwelve
generateKeyPair	3,642.28	3,412.98	3,623.05	3,561.23
sign	3,436.05	3,290.02	3,401.28	3,384.80
signRecoverable	3,451.94	3,271.53	3,391.53	3,366.98
recover	51,568.12	50,361.61	51,352.64	51,186.69
verify	13,691.62	13,379.89	13,444.9	13,365.02
verifyRecoverable	13,688.37	13,555.28	13,414.14	13,368.04

Table 7.4: Transparent Falcon-512 in us

Function	SHAKE256	TurboSHAKE256	cSHAKE256	KangarooTwelve
generateKeyPair	1,702,893.86	1,469,102.25	1,750,183.59	1,572,846.81
sign	10,283.75	10,216.36	10,341.93	10,327.24
signRecoverable	13,039.47	13,638.10	13,777.6	13,387.82
recover	2,432.30	2,401.06	2,407.71	2,429.77
verify	2,256.12	2,232.61	2,240.59	2,262.21
verifyRecoverable	496.24	496.92	498.56	496.50

Table 7.5: Sprout Ed25519 benchmark

Function	Avg (us)
generateKeyPair	326.63
sign	254.39
verify	1,395.66

Finally, this work shows that Falcon (in its 512-bit version) is a promising candidate for replacing ECDSA in blockchain networks in terms of performance and security. This work created independent models so that the results could be related to blockchains other than Zcash, such as Bitcoin. The transparent protocol remains from the "Bitcoin Core" fork [Wikl], and therefore, Falcon can also be seen as an alternative for ECDSA in this context. Nevertheless, this thesis selected Python implementations that have not been optimized for blockchain networks, so these benchmarking results have to be referred to carefully.

8 Conclusion & Future Work

This thesis aims to be interpreted as a starting point to demonstrate feasible directions for upgrading cryptographic technologies employed in blockchain networks to alternatives of PQC. Our independent and modular prototypes can be considered for blockchain networks other than Zcash. For instance, while focusing on Zcash, we investigated the "transparent protocol" 6.2 and the results of replacing ECDSA by Falcon can also be considered for Bitcoin. Nevertheless, quite different technologies applying these cryptographic functions could benefit from this thesis.

Overall, we showed that there are promising technologies to be utilized for future protection against quantum computers. Many of those are still in a theoretical and abstract form and need improvement to be used in real-life applications like the SECUER MPC 6.1, which works as intended but lacks usability because of the Git infrarstructure. While this work achieved positive results, they cannot be directly applied, i.e., in the Zcash codesbase [Comf]. The transformation in terms of programming language, encoding and fitting into the code structure, adjustment of coding style, and testing the functionality should be considered as a future work. We also mentioned that this could be applied to other blockchain implementations like "Bitcoin Core" [Teab] since they partially use similar codebases as from the fork 6.2. Another area we advise to further the research is post-quantum secure ZKPs 4.2 like the ceremony using SECUER 6.1. Ancillary ZK

technologies and their frameworks are continuously evolving and becoming more usable. However, since the cryptographically relevant quantum computer is an emerging threat, these frameworks and technologies should also be transferred to post-quantum secure alternatives.

Bibliography

[AP25] Madhuri S. Arade and Nitin N. Pise. "ZK-STARK: Mathematical Foundations and Applications in Blockchain Supply Chain Privacy." In: Cybern. Inf. Technol. 25.1 (Mar. 2025), pp. 3–18. ISSN: 1314-4081. DOI: 10.2478/cait-2025-0001. URL: https://doi.org/10.2478/cait-2025-0001 (cit. on p. 4).

[Bas+22a] Andrea Basso et al. C-Implementation from "Supersingular Curves You Can Trust". 2022. URL: https://github.com/trusted-isogenies/SECUER-pok/tree/main/c-impl (cit. on pp. 6, 11, 17, 28).

[Bas+22b] Andrea Basso et al. Implementations from "Supersingular Curves You Can Trust". 2022. URL: https://github.com/trusted-isogenies/SECUER-pok/ (cit. on p. iv).

[Bas+22c] Andrea Basso et al. Supersingular Curves You Can Trust. Cryptology ePrint Archive, Paper 2022/1469. 2022. URL: https://eprint.iacr.org/2022/1469 (cit. on pp. 4, 6, 7, 11, 13, 15, 17, 20).

[Coi] Coinmarketcap. Cryptocurrency Prices, Charts And Market Capitalizations. URL: https://coinmarketcap.com (visited on 06/13/2025) (cit. on p. 12).

[Coma] Electric Coin Company. ed25519.rs of zcashd. (Visited on 05/29/2025) (cit. on p. 26).

[Comb] Electric Coin Company. key.cpp of zcashd. (Visited on 05/24/2025) (cit. on p. 22).

[Comc] Electric Coin Company. libsecp256k1 of zcashd. (Visited on 05/24/2025) (cit. on pp. 22, 23, 29).

[Comd] Electric Coin Company. pubkey.cpp of zcashd. (Visited on 05/24/2025) (cit. on p. 22).

[Come] Electric Coin Company. The halo2 Book. URL: `https://zcash.github.io/halo2/concepts.html` (visited on 06/13/2025) (cit. on p. 11).

[Comf] Electric Coin Company. Zcash - Internet Money. URL: `https://github.com/zcash/zcash` (visited on 05/22/2025) (cit. on pp. 3, 5, 6, 12, 13, 22, 26, 29, 30, 33).

[Eij] Helder Eijs. Python library "PyCryptodome". URL: `https://www.pycryptodome.org` (visited on 05/31/2025) (cit. on pp. 7, 28).

[Foua] Zcash Foundation. Homepage of the Zcash Foundation. (Visited on 06/10/2025) (cit. on p. 1).

[Foub] Zcash Foundation. WHAT ARE ZK-SNARKS? URL: `https://z.cash/learn/what-are-zk-snarks/` (visited on 06/06/2025) (cit. on pp. 10, 11).

[Fouc] Zcash Foundation. WHAT IS THE DIFFERENCE BETWEEN SHIELDED ... URL: `https://z.cash/learn/what-is-the-difference-between-shielded-and-transparent-zcash/` (visited on 06/13/2025) (cit. on p. 12).

[Fou+20] Pierre-Alain Fouque et al. Falcon: Fast-Fourier Lattice-based Compact Signatures. 2020. URL: `https://falcon-sign.info/falcon.pdf` (visited on 05/22/2025) (cit. on pp. iv, 3, 4, 6, 24, 29).

[Hop] Daira-Emma Hopwood. Post-Quantum Zcash with Daira-Emma Hopwood - ZconIV. (Visited on 06/10/2025) (cit. on pp. 2, 3, 5).

[Hop+24] Daira-Emma Hopwood et al. Zcash Protocol Specification. 2024 (cit. on pp. iv, 1, 3–5, 12, 13, 22).

[Hor] Mark Horvath. Begin Transitioning to Post-Quantum Cryptography Now. (Visited on 06/06/2025) (cit. on p. iv).

[KP21] Joseph J. Kearney and Carlos A. Perez-Delgado. "Vulnerability of blockchain technologies to quantum attacks." In: Array 10 (2021), p. 100065. ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2021.100065. URL: https://www.sciencedirect.com/science/article/pii/S2590005621000138 (cit. on pp. 3, 4).

[Mar24] Dominik Marchsreiter. Towards Quantum-Safe Blockchain: Exploration of PQC ... Cryptology ePrint Archive, Paper 2024/1178. 2024. URL: https://eprint.iacr.org/2024/1178 (cit. on pp. 3, 24).

[Nak18] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2018. URL: https://bitcoin.org/bitcoin.pdf (cit. on p. 12).

[Pre] Thomas Prest. A python implementation of the signature scheme Falcon. (Visited on 05/24/2025) (cit. on pp. iv, 6, 13, 23, 24, 28, 30).

[RSA78] Ronald Linn Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems." In: Commun. ACM (1978). DOI: 10.1145/359340.359342 (cit. on p. 2).

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." In: SIAM Journal on Computing 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172 (cit. on p. 2).

[ST25] National Institute of Standards and Technology. Post-Quantum Cryptography. 2025. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms (visited on 05/22/2025) (cit. on pp. 2, 3, 9, 13).

[Teaa] Bitcoin Core Team. Bitcoin Core integration/staging tree. (Visited on 05/31/2025) (cit. on p. 29).

[Teab] Bitcoin Core Team. libsecp256k1 of Bitcoin. (Visited on 05/31/2025) (cit. on pp. 6, 12, 30, 33).

[tls] tlsfuzzer. pure-python ECDSA signature/verification and ECDH key agreement. (Visited on 05/25/2025) (cit. on pp. iv, 7, 22, 26, 28–30).

[21] What Is a Hash Function in Cryptography? A Beginner's Guide. 2021. URL: https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide/ (visited on 06/06/2025) (cit. on p. 9).

[Wik25a] Wikipedia. Blockchain. 2025. URL: https://en.wikipedia.org/wiki/Blockchain (visited on 05/04/2025) (cit. on p. 11).

[Wik25b] Wikipedia. Zero-knowledge proof. 2025. URL: https://en.wikipedia.org/wiki/Zero-knowledge_proof (visited on 05/29/2025) (cit. on pp. 10, 22).

[Wika] Wikipedia. CMake. URL: https://en.wikipedia.org/wiki/CMake (visited on 06/14/2025) (cit. on p. 17).

[Wikb] Wikipedia. Cryptography. URL: https://en.wikipedia.org/wiki/Cryptography (visited on 06/10/2025) (cit. on p. 8).

[Wikc] Wikipedia. EdDSA. URL: https://en.wikipedia.org/wiki/EdDSA (visited on 06/11/2025) (cit. on p. 26).

[Wikd] Wikipedia. Elliptic-curve cryptography. URL: https://en.wikipedia.org/wiki/Elliptic-curve_cryptography (visited on 06/11/2025) (cit. on pp. iv, 9).

[Wike] Wikipedia. Extendable-output function. URL: https://en.wikipedia.org/wiki/Extendable-output_function (visited on 06/14/2025) (cit. on pp. 28, 30).

[Wikf] Wikipedia. Non-interactive zero-knowledge proof. URL: https://en.wikipedia.org/wiki/Non-interactive_zero-knowledge_proof (visited on 06/14/2025) (cit. on p. 15).

[Wikg] Wikipedia. Post-quantum cryptography. URL: https://en.wikipedia.org/wiki/Post-quantum_cryptography (visited on 06/18/2025) (cit. on p. 9).

[Wikh] Wikipedia. Public-key cryptography. URL: https://en.wikipedia.org/wiki/Public-key_cryptography (visited on 06/10/2025) (cit. on p. 8).

[Wiki] Wikipedia. SHA-2. URL: https://en.wikipedia.org/wiki/SHA-2 (visited on 06/14/2025) (cit. on p. 20).

[Wikj] Wikipedia. SHA-3. URL: https://en.wikipedia.org/wiki/SHA-3 (visited on 06/14/2025) (cit. on p. 28).

[Wikk] Wikpedia. Elliptic Curve Digital Signature Algorithm. URL: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorith (visited on 06/11/2025) (cit. on p. 9).

[Wikl] Wikpedia. Zcash. URL: https://en.wikipedia.org/wiki/Zcash (visited on 06/23/2025) (cit. on pp. 22, 32).

[Wil16] Zooko Wilcox. The Design of the Ceremony. 2016. URL: https://electriccoin.co/blog/the-design-of-the-ceremony/ (visited on 05/09/2025) (cit. on pp. 1, 4, 5, 12, 13, 15).

[Zeb+24] Yang Zebo et al. "A Survey and Comparison of Post-Quantum and Quantum Blockchains." In: IEEE Communications Surveys Tutorials 26.2 (2024), pp. 967–1002. DOI: 10.1109/COMST.2023.3325761 (cit. on p. 3).

Eidesstattliche Erklärung / Declaration

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeitin gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

I declare that this thesis is a presentation of original work and I am the sole author. Work submitted for this research degree at the University of Passau has not formed part of any other degree either at the University of Passau or at another establishment. All sources are acknowledged as references.

Passau, 23. Juni 2025

Author