

PQMZ: Formally Verified Falcon-PKR for Post-Quantum Cryptographic Migration in Zcash

Hannes Hacker
Barkhausen Institut
Dresden, Germany

Sara Zain
Barkhausen Institut
Dresden, Germany

Shekoufeh Neisarian
University of Passau
Passau, Germany

Elif Bilge Kavun
Barkhausen Institut
Dresden, Germany
Technische Universität Dresden
Dresden, Germany

Abstract

Migrating blockchain systems to Post-Quantum Cryptography (PQC) requires not only replacing classical primitives, but also analysing transformations that change how signatures are represented and verified. We study Falcon-PKR, a public-key-recovery variant of Falcon, motivated by Zcash-like transparent transaction workflows. Falcon-PKR replaces explicit storage of the full Falcon public key with a compact stored digest and reconstructs a candidate public key from the signature during verification. In our construction, this reduces the combined representation size of public key and signature by approximately 15%, but it also introduces a distinct security case in which verification depends on a reconstructed key derived from adversarially supplied signature data.

We address this with a machine-checked EasyCrypt formalisation in the Random Oracle Model (ROM). We mechanically verify an EUF-CMA reduction showing that the security of Falcon-PKR is bounded by the security of standard Falcon together with collision resistance of a domain-separated public-key hash. The analysis therefore treats both assumptions explicitly and does not claim a proof of Falcon itself. We also machine-check correctness for honest signing under an abstract Falcon signing interface. We implement Falcon-PKR for Falcon-512 and Falcon-1024 in Python and C, and evaluate runtime and size against ECDSA and selected Post-Quantum (PQ) alternatives, exposing explicit verification and storage trade-offs. As a complementary case study, we implement and analyse an extended SECUER-based Multi-Party Computation (MPC) ceremony architecture. Together, the two case studies address both transaction-signature and ceremony-infrastructure layers of PQ blockchain migration.

CCS Concepts

• **Security and privacy** → **Privacy-preserving protocols**; • **Software and its engineering** → *Formal software verification*.

Keywords

Post-Quantum Cryptography (PQC), Digital signature, Falcon Public-Key Recovery (PKR), Formal verification, EasyCrypt

ACM Reference Format:

Hannes Hacker, Shekoufeh Neisarian, Sara Zain, and Elif Bilge Kavun. 2018. PQMZ: Formally Verified Falcon-PKR for Post-Quantum Cryptographic Migration in Zcash. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 15 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

The transition to Post-Quantum Cryptography (PQC) is increasingly critical for systems that rely on long-lived public-key cryptography. Large-scale Quantum Computers (QCs) would undermine widely deployed public-key primitives based on integer factorisation and discrete logarithms via Shor's algorithm [47]. This threatens schemes such as Rivest–Shamir–Adleman (RSA) [42] and Elliptic Curve Digital Signature Algorithm (ECDSA) [24], motivating the development and standardisation of Post-Quantum (PQ) key-establishment mechanisms and digital-signature schemes, including the National Institute of Standards and Technology (NIST) PQC standardisation effort [33].

Blockchain systems provide a particularly relevant setting for this transition. Their cryptographic operations are executed at scale, while transaction representation and verification costs directly affect system efficiency. Migrating a blockchain from classical to PQC therefore requires not only secure PQ primitives, but also careful consideration of their concrete representation, verification procedures, and any transformations introduced to integrate them into existing systems. In this work, we use Zcash [23] as a motivating case study. Zcash relies on classical signatures in its transparent transaction protocol, and on a cryptographic ceremony to generate setup parameters for many shielded proofs. PQ migration of a Zcash-like system therefore requires addressing two distinct layers: the transaction-signature layer and the ceremony-infrastructure layer. We study both layers: Falcon-Public-Key Recovery (PKR) targets the signature layer, while our SECUER-based Multi-Party

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, Woodstock, NY

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
<https://doi.org/XXXXXXX.XXXXXXX>

Computation (MPC) case study targets ceremony deployment and operation.

Our primary focus is Falcon [20], a lattice-based PQ signature scheme. We investigate a PKR transformation, which we call Falcon-PKR, in which the Falcon public-key polynomial is not stored explicitly; instead, a compact public-key digest is stored, and the public-key polynomial is reconstructed from the signature during verification. In our implementation results, this replacement reduces the combined size of public key and signature by about 15% for Falcon. This transformation replaces the explicit public-key polynomial with a compact digest, but it also changes a security-critical part of verification: standard Falcon verification checks a signature against a given public key, whereas PKR verification first reconstructs a candidate public key from the signature and then checks its digest against the stored value.

This change raises a security question that cannot be answered by benchmarking alone: does the PKR transformation preserve the Existential Unforgeability under Chosen-Message Attack (EUF-CMA) security of standard Falcon? In a forgery attempt, the signature is adversarially generated, so the reconstructed key is not automatically guaranteed to equal the honestly generated key. The PKR verifier accepts a reconstructed key only if its public-key hash matches the hash of the honestly generated key.

Consequently, a security proof must explicitly account for the case where the reconstructed key differs from the honestly generated key and relate this event to a cryptographic assumption, while also showing that when the two keys coincide, the remaining verification relation is equivalent to that of standard Falcon.

We address this question through a machine-checked, game-based formalisation in EasyCrypt [7]. In the Random Oracle Model (ROM), we machine-checked an EUF-CMA security reduction for Falcon-PKR showing that a successful PKR forgery can be transformed into either a forgery against standard Falcon or a collision in the domain-separated public-key hash. Thus, the security of the PKR transformation reduces to the EUF-CMA security of the underlying Falcon scheme together with an explicit collision-resistance assumption for the additional public-key hash. We separately mechanically verify a correctness experiment for honest signing. Under the properties exposed by the abstract Falcon signing interface, PKR reconstructs the original public key and the resulting PKR signature satisfies the conditions checked by the PKR verifier. The formalisation therefore establishes the security and correctness of the PKR transformation at the abstract Falcon interface, rather than mechanically verifying the concrete low-level Falcon implementation.

We complement the formal analysis with concrete implementations of Falcon-PKR for Falcon-512 and Falcon-1024 in Python and C, including integration with the PQClean implementation [25]. We evaluate the resulting construction in terms of key and signature sizes, signing and verification performance, and compare Falcon and Falcon-PKR with classical schemes relevant to blockchain applications such as ECDSA, as well as with selected PQ alternatives. These experiments quantify the practical effect of PKR and provide the implementation-level context for the formal security analysis.

As a second, complementary case study, we investigate the Supersingular Elliptic Curves with Unknown Endomorphism Rings

(SECUER)-based MPC protocol proposed by Basso et al. [8]. We implement and evaluate an extended architecture, analyse its security-relevant attack surface, and study its practical performance. Together, the two case studies examine complementary aspects of cryptographic migration in blockchain systems: adapting a PQ signature primitive to reduce representation overhead, and evaluating cryptographic infrastructure relevant to protocol deployment.

Contributions. This paper makes the following contributions:

- We implement a PKR mode for Falcon-512 and Falcon-1024 in Python and C, including integration with the PQClean implementation [25].
- We formalise Falcon-PKR in EasyCrypt and mechanically verify an EUF-CMA security reduction in the ROM. The reduction relates Falcon-PKR security to the EUF-CMA security of standard Falcon and the collision resistance of a domain-separated public-key hash.
- We mechanically verify a correctness experiment for Falcon-PKR, showing that, for an honest signing output satisfying the properties exposed by the abstract Falcon signing interface, PKR reconstructs the original public key and the resulting signature satisfies the conditions required for PKR verification.
- We evaluate Falcon and Falcon-PKR for Falcon-512 and Falcon-1024 in terms of key and signature sizes and performance, and compare their characteristics with classical schemes relevant to blockchain applications such as ECDSA, as well as with PQ alternatives.
- We extend and evaluate the SECUER-based MPC implementation proposed by Basso et al. [8], including its practical performance. Besides, we analyse its security-relevant attack surface and argue that the implementation reduces it sufficiently to be considered for practical use.
- We identify and address several implementation issues encountered during the study and contribute corresponding fixes to the evaluated open-source projects where possible.

Paper organization. Sections 2 and 3 present related work and background knowledge. Section 4 describes the SECUER-based MPC implementation and Falcon-PKR. Section 5 presents the machine-checked security and correctness analysis of Falcon-PKR in EasyCrypt. Section 6 presents the experimental evaluation. Finally, Section 7 concludes the paper.

2 Related Work

Our work is related to research on PQ migration of blockchain systems, PKR for digital signatures, formal verification of PQC, and cryptographic ceremony infrastructure.

PQC for blockchains. The migration of blockchain systems to PQC has been studied from both system and cryptographic perspectives. Several works survey PQ blockchain designs and discuss requirements for quantum-resistant cryptocurrencies [28, 38, 45]. Other works investigate the integration of PQ primitives into blockchain platforms, including Hyperledger Fabric and Ethereum, or consider PQ key management and consensus mechanisms [27, 40, 41]. These works primarily address system-level migration and architectural considerations. Our work instead studies a concrete cryptographic transformation relevant to such a migration: PKR for

a PQ signature scheme, together with its implementation, experimental evaluation, and machine-checked security analysis. More closely related to our setting, a quantum-secure blockchain architecture incorporating PKR for Falcon is proposed in [29], while Falcon-PKR is subsequently studied experimentally in a blockchain context in [29]. These works establish the practical motivation for using PKR with Falcon and investigate its implementation and performance. Neither work provides a formal security argument for the PKR transformation: there is no reduction showing that a successful PKR forgery implies a break of standard Falcon EUF-CMA security or a collision in the public-key hash, nor is there an analysis of the security case that arises when the reconstructed key differs from the signer’s actual key. Our work addresses this formal gap by providing a machine-checked security reduction and correctness analysis of Falcon-PKR. More precisely, we show that every successful Falcon-PKR forgery is accounted for by either a standard Falcon EUF-CMA forgery or a collision for the domain-separated public-key hash.

Public-Key Recovery (PKR) and signatures. PKR is well established for classical signature schemes such as ECDSA [24], where a public key can be reconstructed from a signature together with recovery information. The Falcon specification describes a key-recovery mode in which the public key is replaced by a hash of the public-key polynomial and the polynomial is recovered from the signature during verification; this mode is presented as an optional construction [20]. The Python implementation of Falcon by Prest, which we use as one of our implementation baselines, exposes the standard key-generation, signing, and verification interfaces but does not expose a key-recovery interface [37]. Falcon-PKR has been proposed and experimentally evaluated for blockchain applications [29]. Their work motivates and evaluates the construction concretely. Our focus is complementary: we formally analyse whether the PKR transformation preserves EUF-CMA security and mechanically verify the corresponding security reduction and correctness properties.

In particular, PKR verification involves reconstructing a candidate public key from the signature and verifying that its digest matches the stored value. Consequently, security cannot be established from implementation correctness or benchmarking alone: a security argument must account for the possibility that the reconstructed key differs from the honestly generated key. Our formalisation addresses this gap by relating such a discrepancy to the collision resistance of the domain-separated public-key hash and otherwise reducing successful PKR forgery to standard Falcon EUF-CMA security. Related formal work analyses message-recovery signatures in the ideal-cipher model [12], which recovers message data rather than public-key material. The structural difference is significant: in message-recovery schemes, the recovered value is the message itself, verified against a fixed set or by re-signing, whereas in Falcon-PKR, the recovered value is the public-key polynomial, which is subsequently used to evaluate the verification equation. This introduces a separate collision-resistance obligation for H_{pk} that has no counterpart in message-recovery security arguments, and the proof technique of [12] does not apply. To the best of our knowledge, no prior work provides a machine-checked EUF-CMA security analysis of a PKR transformation for a PQ signature scheme.

Formal verification of PQC. Machine-checked security proofs have been developed for several PQ primitives, including Dilithium in EasyCrypt [4], a mechanically verified tight security proof for SPHINCS+ [6], and formal verification of PQ implementations such as Kyber [2]. These works demonstrate the feasibility of mechanically checking security arguments and implementations for modern PQ constructions. In particular, Barbosa et al.’s Dilithium formalisation [4] establishes the game-hopping methodology in probabilistic Relational Hoare Logic (pRHL) that our work builds upon, using byequiv judgments to certify game transitions and Hoare judgments for single-game probability bounds. However, the specific proof obligations arising from Falcon-PKR—reasoning over a public key reconstructed at verification time, a separate collision-resistance game for H_{pk} , and the equivalence between PKR and standard verification via an intermediate reference verifier—have no direct precedent in prior EasyCrypt PQC formalisations. Rather, we formalise the Falcon-PKR transformation in EasyCrypt and mechanically verify its EUF-CMA security reduction in the ROM, treating the EUF-CMA security of standard Falcon as a black-box assumption and without reproving the security of the underlying Falcon construction or mechanically verifying its concrete implementation. The reduction explicitly accounts for the additional collision-resistance assumption introduced by the domain-separated public-key hash used by PKR. We mechanically verify the correctness of honestly generated signatures separately, under the properties exposed by the abstract Falcon signing interface. This positioning is complementary to recent formal analyses of Falcon and conservatively modified variants [19], which study the security of Falcon itself, whereas our result concerns the security-preserving property of the PKR transformation relative to the stated security assumption for standard Falcon.

Cryptographic ceremony infrastructure. Cryptographic setup ceremonies are central to privacy-preserving blockchain systems based on Zero-Knowledge Proof (ZKP) systems. Prior work has studied ceremony designs and specification issues for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (ZK-SNARK) setup [9, 36]. Basso et al. [8] provide a theoretically secure protocol for distributed SECUER setup, but do not address concrete deployment or the implementation-level attack surface of a real ceremony workflow. We complement this work with an extended Git-based SECUER architecture, analysing its security-relevant attack surface and evaluating its practical performance in a blockchain-oriented setting. This complements our Falcon-PKR analysis: the former addresses secure cryptographic setup, while the latter addresses transaction authentication.

3 Preliminaries

This section introduces the cryptographic and blockchain concepts needed for the remainder of the paper.

3.1 Digital Signatures

A digital signature scheme consists of key-generation, signing, and verification algorithms. Its standard security notion is EUF-CMA, which captures an adversary’s inability to produce a valid signature on a fresh message after obtaining signatures on messages of its choice [46]. Formally, the EUF-CMA game proceeds as follows:

a challenger generates a public and private key pair $(pk, sk) \leftarrow \text{KeyGen}()$ and gives pk to adversary $\mathcal{A}$; $\mathcal{A}$ may adaptively query a signing oracle on messages of its choice, and let Q denote the set of messages queried at least once; finally $\mathcal{A}$ outputs a candidate forgery (m^*, σ^*) , where m^* is the target message and σ^* is the candidate signature, and wins if $\text{Verify}(pk, m^*, \sigma^*) = \text{true}$ and $m^* \notin Q$. This is the security notion used in our formal analysis of Falcon-PKR. Classical schemes such as ECDSA rely on the hardness of elliptic-curve discrete logarithms and are therefore vulnerable to large-scale QCs running Shor's algorithm [47]. We consider ECDSA as a classical baseline in our experimental evaluation.

ECDSA additionally supports PKR, allowing a verifier to reconstruct a public key from a signature together with recovery information. This functionality motivates analogous PKR mechanisms for PQ signatures. Falcon-PKR differs from ECDSA recovery in that the recovered object is a Falcon public-key polynomial, whose consistency with the signer-associated key is checked using a public-key digest.

3.2 Falcon

Falcon is a lattice-based PQ digital signature scheme selected by NIST for standardisation [20, 26]. It is characterised by relatively compact public keys and signatures and is designed for efficient software implementations. Falcon provides multiple parameter sets; our implementation and evaluation consider Falcon-512 and Falcon-1024.

At a high level, a standard Falcon signature is the pair (r, s_2) , where r is a uniformly random salt and $s_2 \in \mathbb{Z}_q[x]/\phi$ is a short polynomial with $\phi = (x^n + 1)$ and $q = 12289$. The verifier recomputes $c = H(r||m)$, where m is the message, via the hash-to-point function H , derives $s_1 = c - s_2 \cdot h$, where h is the Falcon public key polynomial, and accepts if and only if $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$, where $\lfloor \beta^2 \rfloor$ is a fixed bound per security level (Falcon-512: $\lfloor \beta^2 \rfloor = 34,034,726$; Falcon-1024: $\lfloor \beta^2 \rfloor = 70,265,242$ [20]). Note that s_1 is *not* transmitted; it is derived by the verifier from the Falcon public key polynomial h , the signature component s_2 , and the random oracle. The public key is the Falcon polynomial h . Falcon-PKR modifies the signature representation and verification flow; its precise construction is presented in Section 3.3.

3.3 Public-Key Recovery (PKR)

In a conventional signature scheme, verification is performed using an explicit public key. A PKR mechanism instead derives a candidate public key from the signature and uses additional information to determine whether the recovered key is the intended one. In Falcon-PKR, the signer-associated value is a digest $H_{pk}(h)$ of the Falcon public-key polynomial h .

Remark 1 (Notation). Throughout this paper, H denotes the *hash-to-point* function (`HashToPoint` in the Falcon specification [20]), modelled as a lazy-sampled random oracle that maps $(r||m)$ to a polynomial in $\mathbb{Z}_q[x]/\phi$. H_{pk} denotes the *public-key hash*, a separate deterministic function mapping a public-key polynomial to a fixed-size digest, assumed to be collision-resistant. These two functions have different domains, different ranges, and play distinct cryptographic roles; they must never be conflated.

The function H_{pk} is *domain-separated* from the hash-to-point oracle H : a distinct domain-separation prefix is used in the hash invocation, ensuring that the collision-resistance assumption for H_{pk} can be applied independently of the random-oracle modelling of H and that the two assumptions do not circularly depend on each other. Given a signature, the verifier reconstructs a candidate polynomial h_{rec} and accepts only if the recovered key satisfies the Falcon verification relation and $H_{pk}(h_{rec}) = H_{pk}(h)$. This introduces a security case that is absent from standard Falcon verification. For an honestly generated public key polynomial h , an adversarially generated signature may reconstruct a different polynomial h_{rec} . If $h_{rec} \neq h$ nevertheless passes the digest check, then h and h_{rec} constitute a collision for H_{pk} . This observation underlies the collision-resistance component of our formal security reduction: the EUF-CMA advantage of Falcon-PKR is bounded by the EUF-CMA advantage of standard Falcon together with the collision-resistance advantage against H_{pk} . In the formalisation, h is drawn from the Falcon key-generation distribution over polynomials in $\mathbb{Z}_q[x]/\phi$, and the domain of H_{pk} is the same set of public-key polynomials. The separation between these domains is necessary for the collision-resistance assumption to be stated independently.

3.4 Machine-Checked Game-Based Proofs

Modern cryptographic security proofs are structured as *security games* [48]: an adversary interacts with a challenger over a sequence of oracle queries and tries to win a defined objective (e.g., forge a signature). Security is established by showing that any adversary winning the new game can be converted—via an efficient *reduction*—into an adversary winning a game already known to be hard. EasyCrypt [7] is a proof assistant for game-based cryptographic security proofs: security games are written as probabilistic programs and game hops are verified by a type-checker using a probabilistic relational Hoare logic, eliminating proof gaps and unchecked reasoning steps that commonly arise in informal pen-and-paper arguments. Concretely, each game hop is expressed as a pRHL judgment $\text{equiv}[G_1 \sim G_2 : \Phi \Rightarrow \Psi]$, where Φ is a relational precondition over the initial states of G_1 and G_2 , and Ψ is the relational postcondition that must hold after both games execute, certified by the EasyCrypt type-checker, and single-game probability bounds are expressed as Hoare judgments $\text{hoare}[G : \Phi \Rightarrow \Psi]$. For game outcomes, we use the conventional EasyCrypt-style notation $\text{Pr}[G : \text{res}]$, where res denotes the Boolean return value of game G . In relational judgments, subscripts indicate the run: res_1 and res_2 refer to the outputs of the left and right programs, respectively. The formalisation reasons about the abstract Falcon signing and verification interface. The PKR transformation introduces proof obligations beyond those of the standard Falcon EUF-CMA game. In particular, the formalisation must account for the public key reconstructed during verification, its binding to the stored public-key digest, and the relation between PKR verification and standard Falcon verification. The resulting security reduction and correctness argument are developed in Section 5.

3.5 Zero-Knowledge Proofs (ZKPs) and ZK-SNARKs

ZKPs allow a prover to convince a verifier about knowledge of a secret without revealing the secret. [43]. ZK-SNARKs provide succinct proofs that can be verified efficiently and are used by blockchain systems for privacy-preserving transactions.

Basso et al. [8] propose an MPC protocol for jointly generating super-singular elliptic-curve parameters with an unknown endomorphism ring. The protocol is designed to avoid reliance on a single participant in the generation of parameters potentially used by subsequent cryptographic protocols.

Our work implements and evaluates an extended architecture based on this protocol. We treat this component as a complementary case study: the Falcon-PKR part of the paper studies a PQ signature transformation and its formal security, whereas the SECUR part examines the implementation and practical characteristics of cryptographic ceremony infrastructure.

3.6 Blockchain Systems

Blockchains are distributed ledgers in which transactions are recorded in an ordered sequence of blocks and authenticated using cryptographic mechanisms [15]. Digital signatures are commonly used to authenticate transactions and other user-authorized actions. For PQ migration, the size of cryptographic objects and the cost of signing and verification are important practical considerations. These considerations motivate our experimental evaluation of Falcon and Falcon-PKR alongside classical and other PQ signature schemes.

Zcash is a cryptocurrency that provides privacy-preserving transactions using ZKPs [23]. It supports both transparent and shielded transaction mechanisms. Transparent transactions use conventional public-key signatures, whereas shielded transactions employ ZKPs to protect transaction information.

We use Zcash as a motivating case study for considering the practical implications of PQ migration in a blockchain setting. In particular, it provides a concrete context for studying both signature-related representation costs and cryptographic ceremony infrastructure. Our implementations and experiments do not constitute an integration of Falcon-PKR or the SECUR-based architecture into the live Zcash protocol.

4 Implementation

This section describes the concrete artefacts used to evaluate and operationalise the two case studies of this paper. We first specify and implement Falcon-PKR with emphasis on security-critical checks required by the formal analysis, and then present the SECUR deployment architecture and its attack-surface hardening decisions.

4.1 Falcon-PKR

Falcon (Section 3.2) operates over the ring $\mathbb{Z}_q[x]/(x^n + 1)$, $n \in \{512, 1024\}$, corresponding to NIST security levels 1 and 5 respectively [34]. The private key is a tuple (f, g, F, G) of short polynomials satisfying $fG - gF = q \bmod \phi$ with f invertible; the public key is $h = gf^{-1} \bmod (\phi, q)$. Signing and verification follow the construction in Section 3.2. The PKR mode (Section 3.3) replaces the stored polynomial h with the digest $H_{pk}(h)$ and augments the signature to the triple (r, s_1, s_2) . Two algorithm changes are required relative

to standard Falcon: (i) the signing loop must iterate until s_2 is invertible; and (ii) the verifier must compute the recovered-key digest $H_{pk}(h_{rec})$ and compare it to the stored $H_{pk}(h)$. Key generation is unchanged. Both changes correspond directly to proof obligations made explicit in the EasyCrypt formalisation (Section 5).

On the one hand, we adapt the signing algorithm (Alg. 1) to include s_1 and to iterate until s_2 is invertible. The signing func-

Algorithm 1 *Sign_recoverable*($\mathbf{m}, \mathbf{sk}, \lfloor \beta^2 \rfloor$) - builds on [20]

Require: A message $\mathbf{m}$, a secret key $\mathbf{sk}$, a bound $\lfloor \beta^2 \rfloor$

Ensure: A recoverable signature $\mathbf{sig}$ of $\mathbf{m}$

```

 $\mathbf{r} \leftarrow \{0, 1\}^{320}$  uniformly
 $\mathbf{c} \leftarrow \text{HashToPoint}(\mathbf{r} \parallel \mathbf{m}, q, n)$ 
 $\mathbf{t} \leftarrow (-\frac{1}{q} \text{FFT}(\mathbf{c}) \odot \text{FFT}(\mathbf{F}), \frac{1}{q} \text{FFT}(\mathbf{c}) \odot \text{FFT}(\mathbf{f}))$ 
do
  do
     $\mathbf{z} \leftarrow \text{ffSampling}_n(\mathbf{t}, T)$ 
     $\mathbf{s} = (\mathbf{t} - \mathbf{z})\hat{\mathbf{B}}$ 
  while  $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$ 
     $(s_1, s_2) \leftarrow \text{invFFT}(\mathbf{s})$ 
     $\mathbf{s} \leftarrow \text{Compress}(s_1 \parallel s_2, 8 \cdot \text{sbytelenrec} - 328)$ 
  while  $\mathbf{s} = \perp$  or  $s_2$  is not invertible  $\bmod (\phi, q)$ 
return  $\mathbf{sig} = (\mathbf{r}, \mathbf{s})$ 
```

tion includes s_1 in the recoverable encoding. Sampling is repeated until the norm bound is satisfied, compression succeeds, and s_2 is invertible $\bmod (\phi, q)$.

On the other hand, we adapt the verification algorithm (Alg. 2) with validity checks and recovered-key digest comparison. The

Algorithm 2 *Verify_recoverable*($\mathbf{m}, \mathbf{sig}, H_{pk}(\mathbf{h}), \lfloor \beta^2 \rfloor$) - builds on [20]

Require: Message $\mathbf{m}$, signature $\mathbf{sig} = (\mathbf{r}, \mathbf{s})$, stored digest $H_{pk}(\mathbf{h})$, bound $\lfloor \beta^2 \rfloor$

Ensure: accept or reject

```

 $(s_1, s_2) \leftarrow \text{Decompress}(\mathbf{s}, 8 \cdot \text{sbytelenrec} - 328)$ 
if  $s_1 = \perp$  or  $s_2 = \perp$  then reject ▷ Reject invalid encodings
end if
if  $s_2$  is not invertible  $\bmod (\phi, q)$  then reject ▷ Recovery requires invertible  $s_2$ 
end if
 $h_{rec} \leftarrow s_2^{-1}(\text{HashToPoint}(\mathbf{r} \parallel \mathbf{m}, q, n) - s_1)$ 
 $pk_{rec} \leftarrow H_{pk}(h_{rec})$ 
if  $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$  and  $pk_{rec} = H_{pk}(h)$  then accept
else reject
end if
```

input value representing the public key $pk = h$ is replaced by a collision-resistant digest $H_{pk}(h)$ (distinct from the hash-to-point oracle H ; see the notation remark in Section 3.3). Verification first validates the decoding result and the invertibility of s_2 . It then computes the recovered-key digest $H_{pk}(h_{rec})$, where

$$h_{rec} = s_2^{-1}(\text{HashToPoint}(\mathbf{r} \parallel \mathbf{m}, q, n) - s_1).$$

Since s_1 is transmitted in the recoverable signature, no additional re-computation of s_1 from h is needed. Finally, the signature is accepted

if and only if (s_1, s_2) satisfies the bound and $H_{pk}(h_{rec}) = H_{pk}(h)$. Despite the fact that the size of the signature doubles, the consequence of storing a collision-resistant digest instead of the full public key polynomial reduces their combined storage costs by 15% [20].

For the actual implementation, we selected the Python library of Prest [37] and the C library of PQClean [25]. Fortunately, both libraries provide functionality for the Number Theoretic Transform (NTT) to do arithmetic calculations directly on the polynomials in $\mathbb{Z}_q[x]/(\phi)$ and check if a polynomial p is invertible if and only if $NTT(p)$ does not contain a coefficient with value zero [20]. Consequently, the algorithmic pseudocode and the outlined adjustments can be implemented straightforwardly in both libraries.

4.2 SECUER

The MPC protocol [8] is designed to generate a super-singular elliptic curve with an unknown endomorphism ring, named SECUER. A new Proof-of-Knowledge (PoK) is described in the protocol such that a party i can prove its knowledge of a secret isogeny $\phi : E_{i-1} \rightarrow E_i$ between two SECUER E_i and E_{i-1} to another party without revealing the secret isogeny. The PoK protocol is based on the SIDH-based Σ -protocol [18] and its subsequent developments [17], and is used to construct a distributed SECUER setup protocol. While the underlying isogeny-based cryptography relates to the hardness of the endomorphism ring problem [35], the security of the protocol is proved in [8] using the Simplified Universal Composability (SUC) framework [13] in the real or ideal world paradigm.

The protocol starts with a SECUER E_0 , and each party i adds another SECUER E_i to the sequence when they have found an isogeny from E_{i-1} to E_i . Hereby, only the sequence of PoK and SECUER is publicly known. The protocol will work correctly and output a SECUER E_t under the assumption that at least one party is honest.

4.2.1 Realisation and attack surface. This section addresses the implementation of each property specified in the distributed SE-CUER protocol [8] in a real-world deployment, including its benefits for the protocol’s security, as well as the extension we applied to improve the protocol.

We designed an architecture (Fig. 1) with three distinct structured repositories: a GitHub repository for participants’ access (left) and two repositories on a self-hosted GitLab instance for executing checks on commits and storing sensitive resources. For conciseness, the three repositories are referred to as LR for the left repository, CR for the center repository, and RR for the right repository. In our architecture, LR allows the participant to commit its generated proof and corresponding curve. The combination of open-source tools and an open-proof format, together with a publicly accessible repository, meets the requirements for transparency, openness, trustworthiness, and a low barrier to entry of the original protocol [8]. Besides, the architecture relies entirely on GitHub and GitLab infrastructure to mitigate and outsource resilience against Sybil and Denial-of-Service (DoS) attacks.

After a party commits as a Pull Request (PR), CR mirrors LR and checks if the committed proof and curve correspond to a valid isogeny between the latest valid curve and the committed one.

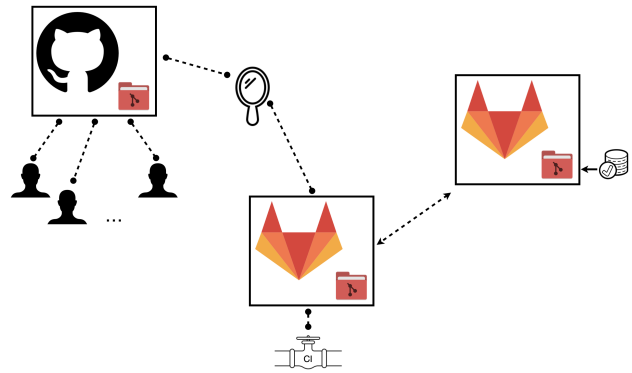


Figure 1: Architecture of the Multi-Party Computation (MPC) protocol utilizing SECUER; access point GitHub repository (left), mirroring Continuous Integration (CI) GitLab repository (center), and secure storage GitLab repository (right).

Therefore, we query a trusted instance of the verification tool [8] starting from the latest valid (“tip”) curve and confirm the absence of errors. Additionally, the execution environment (CR) compares the query output with the committed curve to ensure the correctness of the ongoing sequence of curves and proofs stored in LR.

A successful execution of all configured checks merges the PR and adds a new valid proof and curve to the sequence of proofs and curves in LR, respectively. The PR is rejected otherwise. This portion of the implementation already wraps up the basic functionality of the distributed SECUER setup protocol and can be related to its proven security. Despite that, theory and practice diverge, and the implementation requires further checks to establish a sufficient degree of security.

Since the infrastructure is built on Git, we have to consider platform-specific properties when selecting suitable security measures. A PR and repositories must not rewrite the already verified sequence of proofs and curves, and committing and executing the checks have to be separated [8].

We addressed this issue with the three-part architecture (Fig. 1). LR operates as a public “access point” for participants to send their PRs, while the other two repositories are private. Consequently, parties accessing LR cannot access the other two. Besides, the management of each repository should be distributed among different responsible entities, leading to decentralization and distributing the risk of a full compromise among at least three entities. While CR is only used to mirror LR and execute checks, RR securely stores sensitive resources such as the scripts for the validity checks, the SHA-256 [32] hashes to ensure that the sequence of proofs and curves has not been modified, and a trusted instance of the verification tool [8] to perform proof verification. In case of a valid PR, the resources are updated accordingly. Since CR, which accesses LR and RR only via preset tokens, and RR are private, the remaining attack vector is potentially mostly from the GitHub and GitLab infrastructure.

The SECUER protocol does not explicitly address the requirement that a PR must not change the sequence of proofs and curves, nor does it consider the potential security implications of other

files stored in LR. These include the configuration file for the Continuous Integration (CI) pipeline used to check PRs in CR, as well as potentially ambiguous behaviour that could go undetected in GitHub repositories. The most straightforward approach to this attack vector is to replace the hash values for the sequence of proofs and curves in RR with one hash value over each file and each directory in LR to ensure that not even one bit in a PR is unverified under the assumption of collision resistance of SHA-256 [32].

In summary, the architecture implements the distributed SE-CUER setup protocol proposed for real-world deployment [8]. It applies and extends the proposed security measures, such that the remaining attack vectors are more likely to be contained within the SHA-256 hash function and the GitHub and GitLab infrastructure used.

5 Formal Security Analysis of Falcon-PKR

The benchmarking results in Section 6 establish that Falcon-PKR outperforms standard Falcon verification and achieves a 15% size reduction, making it an attractive candidate for Zcash’s transparent protocol [23]. Benchmarks alone, however, cannot answer whether the PKR transform is *safe* to deploy: altering how a public key is stored and how a signature is verified can introduce subtle attack surfaces even when individual operations appear correct. This section provides a machine-verified answer by formally proving two complementary properties of our Falcon-PKR design—*security* and *correctness*—using the EasyCrypt proof assistant [7]. Together, these proofs provide a machine-checked security and correctness analysis of the PKR transformation at the stated cryptographic and algebraic abstraction level.

5.1 Security Goals and Modelling Assumptions

Our goal is to prove that Falcon-PKR satisfies EUF-CMA [22], the standard security notion for digital signatures used in blockchain protocols. Informally, EUF-CMA states that no computationally bounded adversary—even one that can adaptively request signatures on arbitrary messages—can produce a valid signature on any message it did not previously query. In the Zcash [23] context, breaking EUF-CMA would allow an attacker to forge a transaction signature without the private key, enabling coin theft or identity impersonation on the transparent protocol.

We model the HashToPoint function $H(r||m) \rightarrow c$ as a lazy-sampled random oracle. For each previously unseen input, the oracle samples a fresh value from the specified challenge distribution and stores it; subsequent queries to the same input return the stored value. This provides the consistent random-function abstraction used in the Falcon security analysis [20].

The security reduction relies on two cryptographic assumptions:

- (1) **EUF-CMA security of standard Falcon.** The base sign is a NIST-standardised security claim for Falcon-512 (NIST Level 1) and Falcon-1024 (NIST Level 5) [20, 30].
- (2) **The public-key hash H_{pk} is collision-resistant.** It is computationally infeasible to find two distinct Falcon public-key polynomials $h_1 \neq h_2$ that hash to the same digest under H_{pk} . Critically, H_{pk} must be *domain-separated* from the HashToPoint function used in signing: if the same hash were reused for both purposes, the security argument would be circular

(see Section 6.1). In the Python and C implementations, H_{pk} is instantiated with a separate SHAKE-256 [14] call using a distinct domain-separation prefix.

The formalisation models the ring $\mathbb{Z}_q[x]/\phi$ abstractly through its algebraic operations, with no axiom fixing the degree n . The security and correctness theorems therefore apply uniformly to both Falcon-512 ($n = 512$, NIST Level 1) and Falcon-1024 ($n = 1024$, NIST Level 5) without requiring separate proofs for each parameter set. Implementation details such as NTT or Fast Fourier Transform (FFT) arithmetic are not part of the abstract interface and do not appear in the proof. The central challenge distinguishing this proof from standard Falcon security is that PKR verification does not operate on the stored public key directly, but on a key h_{rec} reconstructed from the signature at verification time. Collision resistance establishes that $h_{rec} = h$ except in the collision event; the intermediate reference-verifier game $G_{mid-ref}$ then connects verification under the recovered key to the standard Falcon verification relation.

5.2 Security Theorem

Theorem 1 (EUF-CMA security reduction for Falcon-PKR). *For any adversary $\mathcal{A}$ against Falcon-PKR in the ROM, its EUF-CMA advantage is bounded by*

$$\text{Adv}_{\text{PKR}}^{\text{EUF-CMA}}(\mathcal{A}) \leq \text{Adv}_{\text{Std}}^{\text{EUF-CMA}}(\text{Red}_\sigma(\mathcal{A})) + \text{Adv}_{H_{pk}}^{\text{CR}}(\text{Red}_{\text{CR}}(\mathcal{A})), \quad (1)$$

where Red_σ transforms a Falcon-PKR adversary into a standard Falcon EUF-CMA adversary, and Red_{CR} transforms it into a collision-finding adversary against H_{pk} . The bound is established by mechanically checked game transformations and reductions in EasyCrypt.

The proof first instruments the Falcon-PKR game with the event $\text{badCR} \iff h_{rec} \neq h \wedge H_{pk}(h_{rec}) = H_{pk}(h)$, where h is the honestly generated public-key polynomial and h_{rec} is the key reconstructed from the forgery. The conjunction explicitly captures the collision condition for the public-key hash: two distinct polynomials map to the same H_{pk} digest. We then split the winning event into cases based on whether badCR occurs (see Figure 2). On the badCR branch, the two distinct polynomials h and h_{rec} have the same H_{pk} digest. Consequently, the forged signature directly yields a collision for H_{pk} , which gives the second term in Theorem 1. On the $\neg\text{badCR}$ branch, an accepting public-key hash check implies $h_{rec} = h$. The PKR verification relation can then be connected to standard Falcon verification through the intermediate reference-verifier game and the algebraic equivalence

$$s_2^{-1}(c - s_1) = h \iff s_1 + s_2 \cdot h = c,$$

where $c = H(r||m)$. The reduction Red_σ uses the standard Falcon signing oracle to simulate the PKR signing oracle and outputs the pair (r, s_2) from a successful PKR forgery as a standard Falcon forgery. This yields the first term in Theorem 1.

Thus, every successful Falcon-PKR forgery is accounted for by one of two events: either it gives a standard Falcon forgery, or it gives a collision in H_{pk} . Consequently, assuming EUF-CMA security of standard Falcon in the stated model and collision resistance of H_{pk} , the bound implies EUF-CMA security of Falcon-PKR. The theorem therefore isolates the additional security assumption introduced by PKR: collision resistance of H_{pk} . It does not independently

establish the concrete security level of the underlying Falcon instantiation.

5.3 Correctness Guarantee

Security preservation concerns adversarially generated signatures; we separately verify the correctness of the PKR transformation for honest signing.

Theorem 2 (Correctness of Falcon-PKR). *For every message m , an honestly generated Falcon key pair and the corresponding PKR signing procedure produce a signature that is accepted by the PKR verification procedure. Moreover, for every signing output satisfying $s_1 + s_2 \cdot h = H(r||m)$ with invertible s_2 , PKR reconstructs the original public key:*

$$\text{RECOVER}(r, m, s_1, s_2) = s_2^{-1}(H(r||m) - s_1) = h.$$

The proof is established compositionally in EasyCrypt. The underlying Falcon signing interface specifies the required postcondition of an honest signing output: s_2 is invertible, the signing relation holds, and the resulting signature satisfies the Falcon norm bound. Together with the construction of s_1 in the PKR signing procedure, these interface properties establish $s_1 + s_2 \cdot h = H(r||m)$ and the conditions required by the PKR verifier. PKR then follows directly from the signing equation. Writing $c = H(r||m)$ and substituting $s_1 = c - s_2 \cdot h$, we obtain $s_2^{-1}(c - s_1) = s_2^{-1}(c - (c - s_2 \cdot h)) = h$, where invertibility of s_2 justifies the use of s_2^{-1} . The PKR transformation leaves s_1 and s_2 unchanged and therefore preserves the signature norm. Once recovery reconstructs h , the signing relation and norm bound consequently satisfy the conditions checked by the PKR verifier, which accepts the honestly generated signature. The correctness experiment is machine-checked in EasyCrypt relative to the abstract Falcon signing interface. The formalisation does not verify the concrete low-level Falcon implementation or its arithmetic, including the NTT implementation; instead, the properties of Falcon signing required by the PKR transformation are represented explicitly at the interface.

5.4 Proof Architecture

The EUF-CMA proof of Theorem 1 follows the four-step structure in Figure 2. We first instrument the Falcon-PKR game with badCR (Step 1). On the $\neg \text{badCR}$ branch, $h_{\text{rec}} = h$, yielding the reference-verifier game $G_{\text{mid-ref}}$ (Step 2), which reduces to standard Falcon EUF-CMA (Step 4). On the badCR branch, $h_{\text{rec}} \neq h$ and $H_{pk}(h_{\text{rec}}) = H_{pk}(h)$, yielding a collision for H_{pk} and a reduction to its collision-resistance game (Step 3). The two branches give the additive bound of Theorem 1.

The figure uses high-level aliases for the corresponding EasyCrypt games: EUF_CMA_PKR , $G_{\text{PKR_CR}}$, and $G_{\text{mid-ref}}$. The right branch reduces to CR_PKHash via Red_{CR} , while the left branch reduces to standard Falcon EUF-CMA via Red_{σ} .

Step 1, Isolating the collision event. We instrument the PKR game to track a *bad event*: the forger produces a signature from which a public key h_{rec} is recovered that differs from the real key h yet hashes to the same digest under H_{pk} . The challenger records this event in an internal flag never exposed to the adversary, preserving the adversary's view and winning probability. In Figure 2, this is the transition $\text{Game}_0 \rightarrow \text{Game}_1$.

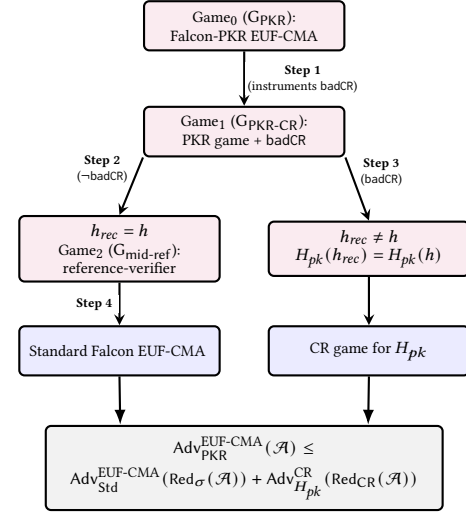


Figure 2: Proof roadmap for the Falcon-PKR EUF-CMA reduction. The PKR game is partitioned by badCR : when $\neg \text{badCR}$ holds, $h_{\text{rec}} = h$ and PKR verification reduces to standard Falcon EUF-CMA; when badCR holds, (h_{rec}, h) yields a collision for H_{pk} via Red_{CR} . The two cases give the final additive security bound. The numbered arrows correspond directly to Steps 1–4 in Section 5.4.

Step 2, Separating the good and bad cases. The PKR verifier first reconstructs the Falcon public key from the signature and checks that its hash matches the stored public-key digest before applying the norm test; standard Falcon verification applies the norm test directly against the known public key. We therefore introduce an intermediate reference-verifier game $G_{\text{mid-ref}}$, whose verifier applies the Falcon norm test directly to the recovered key, without the hash comparison.

The key observation is that, outside badCR , the recovered key must equal the original key, i.e., $h_{\text{rec}} = h$. Hence, on the $\neg \text{badCR}$ branch, the PKR verifier and the reference verifier are equivalent. The reference verifier is then related to standard Falcon verification through the recovery relation $s_2^{-1}(c - s_1) = h \Leftrightarrow s_1 + s_2 \cdot h = c$. The probability bound follows from a case split on badCR within the single instrumented game $G_{\text{PKR-CR}}$:

$$\begin{aligned} \Pr[G_{\text{PKR-CR}} : \text{res}] &= \Pr[G_{\text{PKR-CR}} : \text{res} \wedge \text{badCR}] \\ &\quad + \Pr[G_{\text{PKR-CR}} : \text{res} \wedge \neg \text{badCR}]. \end{aligned}$$

The first term is bounded by $\Pr[G_{\text{PKR-CR}} : \text{badCR}]$ by probability monotonicity. On the $\neg \text{badCR}$ branch, the second term is related to $\Pr[G_{\text{mid-ref}} : \text{res}]$ by a two-sided equivalence argument with postcondition $\text{res}_1 \Rightarrow \text{res}_2 \vee \text{badCR}_1$. Here, res_1 and res_2 denote the Boolean outputs of the left and right games in the corresponding pRHL judgment, and badCR_1 denotes the value of badCR in the left-hand run in EasyCrypt notation).

Here, res_1 and res_2 denote the Boolean outputs of the left and right games in the corresponding pRHL judgment.

Step 3, Reducing the bad event to collision resistance. Any adversary that triggers the bad event—finding $h_{\text{rec}} \neq h$ with matching hashes—yields a collision for H_{pk} directly. A black-box reduction

formalises this: it simulates the PKR game for the forger and outputs the colliding pair. In Figure 2, this is the right branch from Game_1 to the CR game for H_{pk} .

Step 4, Reducing the standard game to Falcon EUF-CMA. A second reduction embeds the standard Falcon challenger inside the PKR game: it receives a standard Falcon public key h , computes $pk_h = H_{pk}(h)$, and uses the standard signing oracle to simulate PKR signatures perfectly (since $s_1 = c - s_2 \cdot h$ requires only h and the standard signature (r, s_2)). The random oracle $H(\cdot)$ for HashToPoint is shared globally between the standard Falcon challenger and the PKR adversary through the same lazy-sampled RO module, so random-oracle queries made by $\mathcal{A}$ during signing are answered consistently with the same map that the standard Falcon challenger uses. The forger’s output (r, s_2) is then a valid standard Falcon forgery. In Figure 2, this is $\text{Game}_2 \rightarrow \text{Standard Falcon EUF-CMA}$. Appendix B provides the corresponding game code, highlighting the statements introduced or modified at each hop and the two reductions underlying the final security bound.

The formalisation implications for implementation and deployment are presented in Section 6.1, followed by the empirical evaluation and integrated discussion in Section 6.

6 Practical Implications and Evaluation

This section links the formal results to deployment and performance evidence. We first extract implementation requirements implied by the machine-checked proof (Section 6.1), then analyse the SECUER deployment architecture (Section 6.2), evaluate Falcon-PKR performance and size trade-offs (Section 6.3), and finally provide an integrated discussion across these findings (Section 6.4).

6.1 Formalisation-Driven Implementation Requirements

The machine-checked formalisation exposed three requirements that are important for both the security argument and the implementation of Falcon-PKR.

Collision resistance and domain separation of H_{pk} . The reduction shows that collision resistance of the public-key hash is essential for relating a recovered key h_{rec} to the honestly generated key h . In particular, if verification accepts with $H_{pk}(h_{rec}) = H_{pk}(h)$ and $h_{rec} \neq h$, the reduction obtains a collision in H_{pk} . The formalisation therefore makes the collision-resistance assumption explicit rather than treating it as an informal property of the construction. The hash must also be domain-separated from the HashToPoint oracle used by Falcon signing. This allows the collision-resistance assumption to be used independently of the random-oracle reasoning for HashToPoint. Our implementation consequently uses a dedicated, domain-separated SHAKE-256 [14] invocation for H_{pk} .

Invertibility of s_2 . PKR requires s_2 to be invertible in $\mathbb{Z}_q[x]/\phi$. If s_2 is not invertible, the recovered key is not used as a valid public key, and the verifier rejects through its explicit invertibility condition. The mechanised correctness proof therefore makes this condition explicit and confirms the role of the rejection-sampling loop in `sign_recoverable` (Section 4.1): successful signing produces an invertible s_2 , rather than this check being merely an implementation optimisation.

An intermediate reference verifier. The mechanisation introduces an intermediate reference-verifier game $G_{\text{mid-ref}}$ (Section 5.4, Step 2) that connects PKR verification with the standard Falcon verification relation. PKR verification operates on the recovered public key, whereas standard verification operates directly on the original public-key polynomial. Introducing $G_{\text{mid-ref}}$ separates these two steps and makes the security-preserving transformation explicit: the collision-resistance argument relates h_{rec} to h , after which the reference verification relation can be connected to standard Falcon EUF-CMA security.

Together, the two theorems and the three insights above provide a formal cryptographic justification for Falcon-PKR as a compact PQ replacement for ECDSA in the Zcash [23] transparent protocol. The formalisation is parameter-independent at the abstract ring level; the security and correctness theorems hold for any concrete instantiation of the ring, and in particular cover both Falcon-512 and Falcon-1024 without modification. The two parameter sets are evaluated separately in the experimental implementation (Section 6.3), so the formal guarantees apply to both sets of benchmarks presented there.

6.2 SECUER Deployment and Attack Surface

This subsection discusses and connects the results of this work, preceding a remark on the libraries of Prest [37] and PQCclean [25] for Falcon-PKR, the provided tools of Basso et al. [8] for SECUER PoK, and an Elliptic Curve Cryptography (ECC) library [50] in Python for a subsequent performance comparison. Generally, we put a special emphasis on the comparison in a blockchain context. We identify formatting errors in the SECUER PoK tools, which lead to compilation issues on the Apple Silicon architecture; missing support for the hash functions of PyCryptodome [16] in the ECC library, which are required for the following performance analysis; and missing PKR mode for Falcon in the libraries of Prest [37] and PQCclean [25]. PRs containing the resolved issues will be sent to the respective public open-source GitHub repositories. All performance benchmarks are executed on an Apple MacBook Air with Apple Silicon architecture, the M3 Apple chip, and 24 GB of memory. The work of Basso et al. [8] extends over several SIKE parameters; however, their provided tools showed severe execution errors such that we were only able to build our evaluation on the SIKE parameter 434.

In Section 4.2, we analyze our implementation of the real-world deployment regarding the distributed SECUER setup protocol. Therefore, we investigate each property and compare it with our realised architecture (Fig. 1). Moreover, we discuss which properties may lead to attacks, identify two shortcomings in the original proposition, and outline a solution. First, a third repository extends the architecture to store sensitive resources, e.g., scripts for PR verification and hash values for verified directories, and reduces the mirroring GitLab repository to act solely as an execution environment. Independent entities should manage all three repositories to spread the risk of a single point of failure or compromise. Second, we point out that it is not sufficient to ensure that a PR does not change the directories for the verified sequences of proofs and curves. Rather, no other file or directory than the one responsible

Table 1: Proof-of-Knowledge (PoK) performance (ms).

Algorithm	Proof	Verify
SECUER-PoK	2763	316.2
Schnorr signature (libsecp256k1)	0.0106	0.0199
SQISign	30.746	2.114

for new proofs and curves should be modified. A simple counter-measure is to create an additional hash value of all directories and files, except the directory for new additions, and verify against it. Hence, we argue this implementation reduces the attack surface primarily to that of the underlying infrastructure of GitHub and GitLab. Overall, the implementation adheres to the requirements in [8] and therefore presumably correctly implements the protocol which was proven to be secure. However, the reliance on the Git infrastructure leads to limited usability and inconsistent behaviour; e.g., if a participant slightly deviates from the protocol, verification against the valid hash value across all directories and files will reject the PR.

We conclude the examination of the SECUER implementation with a performance-based comparison. Table 1 shows the benchmark results of the SECUER PoK [8] tools and two signature schemes fundamentally built on a PoK. The Schnorr Signature scheme [31, 54] relies on the hardness of the Discrete Logarithm Problem (DLP). Similarly, SQISign [1] relies on the hardness of computing isogenies between supersingular elliptic curves, an assumption that also underlies the SECUER PoK.

Our measurements show that the PoK [8], particularly in the provided implementation, is orders of magnitude slower than both the Schnorr and SQISign signatures.

Although the SECUER PoK [8] does not yield the best results for further implementation study, we select it for its quantum-secure properties and, in particular, its distributed setup protocol. Large-scale MPC ceremonies such as the Zcash setup [52] have been conducted over several days or weeks, with individual contributions requiring substantial computation [10, 11, 21]. Our implementation of the distributed SECUER setup protocol achieves an average runtime of 1.12 minutes per successful contribution. This suggests that the protocol can support practical ceremony workflows, including settings with hundreds of participants. Combined with the quantum-secure properties of the SECUER PoK, these results motivate its use as a candidate for quantum-secure blockchain setup.

6.3 Falcon-PKR Performance and Size Evaluation

Building on Section 4.1, this subsection quantifies the runtime and size impact of enabling PKR mode in Falcon. We compare Falcon-PKR against standard Falcon and representative classical/PQ signatures, with emphasis on ECDSA [24] in Zcash-like transparent settings [23], to make the deployment trade-off explicit. Measurements are obtained from the Python [37] and C [25] implementations introduced earlier.

Table 2 and Table 3 hold the performance values of the ECDSA library benchmark [50] and those of Prest's Falcon library [37] in Python. These tables indicate Falcon's significantly higher execution time for key pair generation; however, this overhead can

Table 2: Transparent ECDSA (μ s).

Function	SHAKE256	TurboSHAKE256	cSHAKE256	KangarooTwelve
generateKeyPair	404.89	409.15	410.28	404.47
sign	410.86	411.53	408.95	409.24
recover	5,972.06	5,927.64	5,932.20	5,845.63
verify	1,584.45	1,559.91	1,557.17	1,548.78

Table 3: Transparent Falcon-512 (μ s).

Function	SHAKE256	TurboSHAKE256	cSHAKE256	KangarooTwelve
generateKeyPair	1,711,027.28	1,729,447.30	1,718,898.43	1,675,201.66
sign	11,034.33	11,089.65	10,992.24	11,066.97
signRecoverable	14,255.76	14,350.07	14,235.87	14,224.97
recover	2,366.34	2,390.74	2,373.85	2,430.66
verify	2,226.31	2,231.70	2,220.97	2,290.61
verifyRecoverable	479.95	520.00	475.33	484.43

be considered negligible, as key pairs in general and especially in blockchain networks are typically generated only infrequently, e.g., a participant creates a new personal wallet. Additionally, Falcon's signing functions are at least around 27 times slower than ECDSA's. While this is a negative aspect, it can be relativized as digital signatures are mostly created once per blockchain transaction per participant. However, blockchain network nodes verify these signatures to check the authenticity of the transactions and the integrity of the blockchain. Consequently, the verification operation has a greater impact on blockchain performance than the signing operation, as it is executed more frequently. In this regard, the Falcon implementation in recoverable form outperforms the ECDSA implementation by a factor of approximately 3.3.

In general, the total time-based cost for k number of validations of transactions of potential blockchains using PKR can be expressed as:

$$T_{\text{total}} = T_{\text{sign}}^{\text{rec}} + k \cdot \left(T_{\text{pk}}^{\text{rec}} + T_{\text{verify}}^{\text{rec}} \right) \quad (2)$$

where $T_{\text{sign}}^{\text{rec}}$ is the time for generating a recoverable signature, $T_{\text{pk}}^{\text{rec}}$ is the time for recovering the public key from the signature, $T_{\text{verify}}^{\text{rec}}$ is the time for verifying the recoverable signature with the recovered key.

Using the values from Table 2 and Table 3 for SHAKE256SHAKE-256 [14], one can observe the following.

$$\begin{aligned}
 T_{\text{total}}^1 &> T_{\text{total}}^2 \\
 410.86 + n * (5,972.06 + 1,584.45) \\
 &> 14,255.76 + n * (2,366.34 + 479.95) \\
 \Leftrightarrow 410.86 + n * 7,556.51 &> 14,255.76 + n * 2,846.29 \\
 \Leftrightarrow 410.86 &> 14,255.76 - n * 4,710.22 \\
 \Leftrightarrow -13,844.90 &> -n * 4,710.22 \\
 \Leftrightarrow n * 4,710.22 &> 13,844.90 \\
 \Leftrightarrow n &> 2.93933 \\
 \Leftrightarrow n &\geq 3
 \end{aligned} \quad (3)$$

For $n \geq 3$, the ECDSA Python implementation [50] is overall slower than the Falcon Python implementation [37], while Falcon additionally provides quantum-secure properties. Besides, we create the benchmarks for different eXtensible Output Function

Table 4: Lengths of the digital signature components (in bytes).

Algorithm	Public Key	Private Key	Signature
ECDSA (Python [50])	64	32	64
libsecp256k1	64	32	64
Falcon-512 (Python [37])	1014	1938	666 (recoverable: 1291)
Falcon-512	897	1,281	752
Falcon-1024	1,793	2,305	1,462
ML-DSA-44	1,312	2,560	2,420
ML-DSA-65	1,952	4,032	3,309
ML-DSA-87	2,592	4,896	4,627
SPHINCS-SHAKE256-128f-simple	32	64	17,088
SPHINCS-SHAKE256-128s-simple	32	64	7,856
SPHINCS-SHAKE256-192f-simple	48	96	35,664
SPHINCS-SHAKE256-192s-simple	48	96	16,224
SPHINCS-SHAKE256-256f-simple	64	128	49,856
SPHINCS-SHAKE256-256s-simple	64	128	29,792
SPHINCS+-SHAKE256-128f	32	64	17,088
SPHINCS+-SHAKE256-128s	32	64	7,856
SPHINCS+-SHAKE256-192f	48	96	35,664
SPHINCS+-SHAKE256-192s	48	96	16,224
SPHINCS+-SHAKE256-256f	64	128	49,856
SPHINCS+-SHAKE256-256s	64	128	29,792

(XOF) hash functions to evaluate significant performance differences. Here our contributions to accommodate the hash functions of PyCryptodome [16] come into play, before these, the XOF hash functions have not been featured. A dataset of 2000 instances is utilized, where each instance records the runtime of a specific operation type (generate key pair, sign, recover, verify) with a specific XOF hash function (SHAKE256, TurboSHAKE256, cSHAKE256, KangarooTwelve). This dataset allows us to assess the statistically significant influence of the operation type and the XOF hash function on the runtime. We create a multiple linear regression model, where the dependent variable is the runtime and the predictors (independent variables) are the XOF hash function type and the operation type. The results demonstrate that the type of hash function has a statistically significant impact on the runtime for ECDSA, but not for Falcon, at the strict significance level of $\alpha = 0.01\%$ (i.e., there is a 0.01% probability of incorrectly rejecting the null hypothesis, which states that the predictors have no effect on runtime). This is verified by the Overall F-test [39], which checks whether the predictors improve the model performance compared to assuming no effect at all. On average, KangarooTwelve yields the best performance for the Python ECDSA implementation.

It is worth noting that runtime is not the only critical factor when evaluating the performance and suitability of a digital signature algorithm in the blockchain context. The lengths of the keys and the signatures themselves also play an important role, as shown in Table 4.

Comparing the Python implementations reveals that Falcon’s public key is nearly 16 times larger than ECDSA’s, and Falcon’s signature is more than 10 times larger than those of ECDSA’s signature, as well as 20 times larger in its recoverable form. Both the public key and the signature are cryptographic components that are publicly available on a blockchain network and utilized for verification and integrity. These larger sizes likely affect network performance negatively, although the exact impact depends on the concrete circumstances.

The private key of the Falcon algorithm is more than 60 times larger than ECDSA’s. In a blockchain context, private keys are typically stored in personal wallets; thus, their size has a greater

impact on individual storage than on the blockchain network itself. Overall, the cryptographic components of Falcon are at least an order of magnitude larger than those of ECDSA. Thus, it is evident that the storage is increased significantly, which also increases the necessary resources.

In concluding whether Falcon is a suitable candidate to replace ECDSA in terms of the utilized Python implementations, the security levels of the algorithms must also be taken into account. NIST categorizes five security levels [34] for cryptographic algorithms, which are listed (algorithm and level) to other standardized cryptographic algorithms: Falcon-512/-1024 [20] with 1/5, ECDSA (secp256k1) with one, ML-DSA-44/-65/-87 [44] with 2/3/5 and SPHINCS, as well as SPHINCS+, in the variants 128/192/256 with 1/3/5. The results indicate that ECDSA on the curve secp256k1 and Falcon-512 fall within the same NIST security level. Consequently, if the PKR mode is employed, ECDSA is inferior to Falcon-512 in terms of performance in the blockchain context, as shown in Eq. 3, but has the edge on storage. This implies that omitting PKR leads to a significant performance reduction for Falcon-512, while utilizing PKR results in a performance improvement, in addition to providing quantum-secure properties. Since ECDSA and Falcon-PKR have PKR features, we do not need to compare that.

The programming language Python is known for its simplicity and understandability, but not for its efficiency and performance. Even though it is an unconventional choice for cryptographic implementations, it can deploy cryptographic algorithms, as in the PyCryptodome library [16], and should be considered in security and cryptographic research. Nevertheless, we also perform benchmarks on C implementations of relevant cryptographic digital signature algorithms. We begin by comparing ECDSA on the secp256k1 curve [53] with the Falcon implementation from the PQClean project [25]. Table 5 shows that the libsecp256k1 functions sign, recover, and verify each require approximately 20 μ s. In contrast, Table 5, which we generate by adapting the PQClean implementation of Falcon-512 [25] to support the PKR mode, indicates that the functions keypair, sign, verify, sign recoverable, and verify recoverable (verify + recover) are slower by at least a factor of 3 and Falcon-1024, shown in Table 5, by at least a factor of 6. This demonstrates that Falcon causes a performance loss compared to ECDSA at a high level of optimization in the C programming language and limits its advantages to its quantum-secure properties. In general, this does not conclusively determine whether Falcon is a suitable replacement for ECDSA, as it is still assumed that quantum-secure properties are required to prepare for the potential emergence of a QC in the near future.

On this account, we compare the performance of Falcon with different ML-DSA and SPHINCS variants from the PQClean project [25], which are also selected winners of the NIST PQC competition [33]. NIST selected SPHINCS+ for standardization; however, since PQClean provides only SPHINCS, we take SPHINCS+ as its reference implementation [49], tailored for the AARCH64 architecture like the PQClean implementations, for the following benchmarks. The PKR functionality is excluded from this comparison. Table 6 depicts that both Falcon variants are among the highest ones in terms of key generation. This should be considered as a clear disadvantage, but as described above, it would not impact the performance of a blockchain itself. Considering NIST security

Table 5: (Recoverable) Functions in libsecp256k1 and PQClean Falcon.

Algorithm	Function	Avg (μ s)
Falcon-512	keypair	18,595.0
Falcon-1024	keypair	56,266.0
ECDSA (libsecp256k1)	sign	16.40
Falcon-512	sign	1,154.0
Falcon-1024	sign	2,378.0
ECDSA (libsecp256k1)	sign recoverable	16.40
Falcon-512	sign recoverable	1,231.0
Falcon-1024	sign recoverable	2,604.0
ECDSA (libsecp256k1)	verify	21.30
Falcon-512	verify	70.7
Falcon-1024	verify	141.0
ECDSA (libsecp256k1)	verify + recover	44.0 (21.30 + 22.70)
Falcon-512	verify recover	89.5
Falcon-1024	verify recover	180.0

levels 1 and 5, as described in 6.3 (Falcon-512 and Falcon-1024, respectively), Falcon performs roughly an order of magnitude slower than all ML-DSA variants, but at least an order of magnitude faster than SPHINCS and SPHINCS+ for signing. Regarding verification, Falcon is only slightly slower than the ML-DSA variants, while all SPHINCS and SPHINCS+ variants are slower than the other algorithms, making Falcon and ML-DSA the clear performers in this metric.

6.4 Integrated Discussion

The combined evidence of performance and component sizes indicates that no single signature family is uniformly optimal across all criteria. In our measurements, ML-DSA variants are faster than Falcon in key generation and signing (Table 6), while Falcon remains substantially faster than SPHINCS/SPHINCS+ for signing. On verification, Falcon-512 (70.7 μ s) is slightly slower than ML-DSA-44 (54.9 μ s) at the same NIST security level, but faster than all SPHINCS/SPHINCS+ variants.

For blockchain settings in which PKR is a hard requirement, Falcon remains a practical option because it supports a PKR-compatible construction and admits a formal security reduction for the transformation under explicit assumptions. This advantage is structural: it is not only a benchmark outcome but also a compatibility property of the signature workflow.

Storage and throughput trade-offs remain explicit. Table 4 shows that Falcon signatures and keys are larger than ECDSA, while PKR reduces combined public-key-plus-signature representation for Falcon in our implementation setting by about 15%. Compared with SPHINCS/SPHINCS+, Falcon offers markedly smaller signatures; compared with ML-DSA, the trade-off is parameter-dependent between runtime and object sizes.

Overall, the results support a constrained claim: Falcon-PKR is a credible candidate for systems that simultaneously require PQ security, practical verification performance, and PKR-style transaction formats. For systems without PKR constraints, algorithm selection should remain workload-driven and threat-model-driven rather than fixed to a single primitive.

Table 6: PQClean digital signature algorithms (μ s).

Algorithm	Keypair	Sign	Verify
Falcon-512	18,595	1,154	70.7
Falcon-1024	56,266	2,378	141
ML-DSA-44	65.4	136	54.9
ML-DSA-65	122	230	81.7
ML-DSA-87	152	267	126
SPHINCS-SHAKE256-128f-simple	501	11,859	832
SPHINCS-SHAKE256-128s-simple	31,843	24,9617	309
SPHINCS-SHAKE256-192f-simple	869	22,925	1,414
SPHINCS-SHAKE256-192s-simple	56,999	531,835	528
SPHINCS-SHAKE256-256f-simple	2,711	56,959	1,731
SPHINCS-SHAKE256-256s-simple	44,708	545,146	913
SPHINCS+-SHAKE256-128f	865.6	13,382.1	862
SPHINCS+-SHAKE256-128s	35,796.8	263,952	308.8
SPHINCS+-SHAKE256-192f	1,105.8	20,624.2	1,319.2
SPHINCS+-SHAKE256-192s	53,301.5	450,565.2	485.8
SPHINCS+-SHAKE256-256f	3,533.6	41,932.8	1,292.6
SPHINCS+-SHAKE256-256s	35,701.2	393,889.6	654

7 Conclusion and Future Work

This paper studies PQ migration in blockchain-relevant cryptographic workflows through two complementary case studies: PKR mode for Falcon and SECUER-based ceremony infrastructure. For Falcon-PKR, we provide a machine-checked EasyCrypt analysis of security and correctness at an abstract Falcon interface. In the ROM, we prove that the EUF-CMA advantage of Falcon-PKR is bounded by the EUF-CMA advantage of standard Falcon under the standard reduction and the collision-resistance advantage against a domain-separated public-key hash. The proof therefore isolates the additional assumption introduced by PKR. It also makes explicit the requirements for domain separation and invertibility of s_2 . Separately, the correctness proof establishes that, under the specified signing-interface properties, honest PKR signatures recover the original public key and are accepted by the PKR verifier.

For implementation and evaluation, we realise Falcon-PKR for Falcon-512 and Falcon-1024 and benchmark it against classical and PQ alternatives. The results show a clear trade-off profile: PKR improves representation efficiency for Falcon in our setting and can improve verification-side costs in workflows that include recovery, while Falcon still incurs larger keys/signatures than ECDSA and does not uniformly dominate all alternatives across all metrics. For ceremony infrastructure, we implement and analyse a practical SECUER-based deployment architecture and identify concrete operational hardening requirements for Git-based workflows. This complements the signature-side analysis by addressing setup-layer security in the same blockchain migration context.

Overall, the evidence supports a bounded conclusion: Falcon-PKR is a credible candidate for systems that simultaneously require PQ security and PKR-style transaction formats, provided its explicit assumptions are acceptable, and its storage/runtime trade-offs match the target workload.

Future work. Our results suggest three directions for future research. First, extending evaluation from library-level benchmarks to end-to-end blockchain test deployments to quantify mempool,

bandwidth, and validation effects under realistic load. Second, refining the SECUER deployment architecture toward stronger usability, automation, and decentralised operational controls while preserving its security invariants. Third, extending the formal analysis from ROM to QROM. This is an explicit scope boundary rather than an omission. PQ-Squirrel [3] targets computationally sound symbolic verification of post-quantum protocols, whereas qRHL toolchains [51] support relational reasoning about quantum programs. Adapting the Falcon-PKR reduction and its algebraic interface to any of these settings remains future work. A renewed EasyPQC [5] framework is under development, making a fully machine-checked QROM treatment of Falcon-PKR a natural next step.

References

- [1] Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Isaac A. Canales Martínez, Jorge Chávez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan K. Eriksen, Tako B. Fouotsa, Décio L. Gazzoni Filho, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez-Henríquez, Sina Schaeffler, and Benjamin Wesolowski. 2025. *SQIsign*. Technical Report. <https://sqisign.org>
- [2] José B. Almeida, Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Vincent Laporte, Jean-Christophe L  chenet, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, and Peter Schwabe. 2023. Formally verifying Kyber: Episode IV: Implementation correctness. *LACR Transactions on Cryptographic Hardware and Embedded Systems* 2023, 3 (2023), 164–193. doi:10.46586/tches.v2023.i3.164-193
- [3] David Baelde, Antoine Dallon, St  phanie Delaune, Charlie Jacomme, and Adrien Koutsos. 2026. Robust logical foundations for mechanizing post-quantum cryptography in Squirrel. In *CCS 2026-ACM Conference on Computer and Communications Security*.
- [4] Manuel Barbosa, Gilles Barthe, Christian Doczkal, Jelle Don, Serge Fehr, Benjamin Gr  goire, Yu-Hsuan Huang, Andreas H  lsing, Yi Lee, and Xiaodi Wu. 2023. Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium. In *Annual International Cryptology Conference*. Springer, 358–389.
- [5] Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Gr  goire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. 2021. EasyPQC: Verifying Post-Quantum Cryptography. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*. 2564–2586.
- [6] Manuel Barbosa, Fran  ois Dupressoir, Andreas H  lsing, Matthias Meijers, and Pierre-Yves Strub. 2024. A tight security proof for SPHINCS+, formally verified. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 35–67.
- [7] Gilles Barthe, Fran  ois Dupressoir, Benjamin Gr  goire, C  sar Kunz, Benedikt Schmidt, and Pierre-Yves Strub. 2012. EasyCrypt: A tutorial. *International School on Foundations of Security Analysis and Design* 8604 (2012), 146–166. doi:10.1007/978-3-319-10082-1_6
- [8] Andrea Basso, Giulio Codogni, Deirdre Connolly, Luca De Feo, Tako Boris Fouotsa, Guido Maria Lido, Travis Morrison, Lorenz Panny, Sikhar Patranabis, and Benjamin Wesolowski. 2023. Supersingular curves you can trust. In *Advances in Cryptology - EUROCRYPT 2023*, Vol. 14005. Springer Nature Switzerland, 405–437.
- [9] Muhammed A. Bingol. 2024. Multi-Party setup ceremony for generating multi-variate ZK-SNARK parameters. <https://eprint.iacr.org/2024/1671.pdf>
- [10] Sean Bowe. 2018. *Announcing the Sapling MPC*. Technical Report.
- [11] Sean Bowe. 2018. *Completion of the Sapling MPC*. Technical Report.
- [12] Daniel R.L. Brown and Don B. Johnson. 2001. Formal security proofs for a signature scheme with partial message recovery. In *Cryptographers' Track at the RSA Conference*. Springer, 126–142.
- [13] Ran Canetti, Asaf Cohen, and Yehuda Lindell. 2015. A simpler variant of universally composable security for standard multi-party computation. In *Advances in Cryptology - CRYPTO 2015*, Rosario Gennaro and Matthew Robshaw (Eds.). Springer Berlin Heidelberg, 3–22.
- [14] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. 2020. Recommendation for stateful hash-based signature schemes. Specification: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf>.
- [15] Shi Dong, Khushnood Abbas, Meixi Li, and Joarder Kamruzzaman. 2023. Blockchain technology and application: An overview. *PeerJ Computer Science* 9 (2023), e1705. doi:10.7717/peerj-cs.1705
- [16] Helder Eijls. 2026. *Python library "PyCryptodome"*. <https://www.pycryptodome.org>
- [17] Luca De Feo, Samuel Dobson, Steven D. Galbraith, and Lukas Zobernig. 2022. SIDH Proof of Knowledge. In *Advances in Cryptology - ASIACRYPT 2022*, Shweta Agrawal and Dongdai Lin (Eds.). Springer Nature Switzerland, 310–339.
- [18] Luca De Feo, David Jao, and J  r  me Pl  t. 2014. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology* 8, 3 (2014), 209–247. doi:10.1515/jmc-2012-0015
- [19] Pierre-Alain Fouque, Phillip Gajland, Hubert De Groote, Jonas Janneck, and Eike Kiltz. 2026. A closer look at Falcon. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 3–31.
- [20] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2018. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's Post-Quantum Cryptography standardization process* 36, 5 (2018), 1–75. <https://falcon-sign.info/falcon.pdf>
- [21] Billy Garrison. 2018. *Participating in the Zcash Sapling MPC*. Technical Report.
- [22] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing* 17, 2 (1988), 281–308. doi:10.1137/0217017
- [23] Daira-Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2024. Zcash protocol specification. Specification: <https://zips.z.cash/protocol/protocol.pdf>.
- [24] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The Elliptic Curve Digital Signature Algorithm (ECDSA). *Int. J. Inf. Secur.* 1, 1 (Aug. 2001), 36–63. doi:10.1007/s102070100002
- [25] Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, , and Thom Wiggers. 2022. Improving software quality in cryptography standardization projects. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 19–30.
- [26] Kwangjo Kim. 2025. Comparison of post-quantum signatures - Falcon vs SOLMAE with Python. In *Proceedings of the 20th Asia Joint Conference on Information Security (EPiC Series in Computing, Vol. 106)*, Akira Yamada, Huy Kang Kim, Yujue Wang, and Tung-Tso Tsai (Eds.). EasyChair, 187–198.
- [27] Sofia Kireieva and Roman Olynykov. 2025. Integrating Falcon-512 signatures into Ethereum 2.0: Implementation and experimental analysis. *Collection of Scientific Papers Scientia* (2025), 161–168.
- [28] Miguel A. Le  n-Ch  vez, Lucas P. Perin, and Francisco Rodr  guez-Henr  quez. 2022. Post-Quantum digital signatures for Bitcoin. In *Principles and Practice of Blockchains*. Springer, 251–270.
- [29] Dominik Marchsreiter. 2025. Towards quantum-safe blockchain: Exploration of PQC and public-key recovery on embedded systems. *IET Blockchain* 5, 1 (2025). doi:10.1049/blc2.12094
- [30] National Institute of Standards and Technology (NIST). 2026. Post-quantum cryptography standardization. <https://csrc.nist.gov/projects/post-quantum-cryptography>
- [31] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. 2020. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. Association for Computing Machinery, 1717–1731.
- [32] National Institute of Standards and Technology (NIST). 2015. FIPS PUB 180-4: Secure Hash Standard (SHS). Specification: <https://csrc.nist.gov/pubs/fips/180-4/upd1/final>.
- [33] National Institute of Standards and Technology (NIST). 2025. Post-Quantum Cryptography: Selected algorithms. Report: <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/selected-algorithms>.
- [34] National Institute of Standards and Technology (NIST). 2026. *Post-Quantum Cryptography security (evaluation criteria)*. Technical Report. [https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-\(evaluation-criteria\)](https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria))
- [35] Aurel Page and Benjamin Wesolowski. 2023. The supersingular endomorphism ring and one endomorphism problems are equivalent. *Cryptology ePrint Archive*, Paper 2023/1399. <https://eprint.iacr.org/2023/1399>
- [36] Chanik Park, Moonhyeon Chung, and Donghyeon Ryu. 2023. A blockchain-based protocol of trusted setup ceremony for zero-knowledge proof. In *Proceedings of the 5th Blockchain and IoT Conference*. Association for Computing Machinery, 35–40.
- [37] Thomas Prest. 2026. Falcon.py: A Python implementation of the signature scheme Falcon. Software repository. <https://github.com/tprest/falcon.py>
- [38] Muhammad T. Ramzan and Stelvio Cimato. 2025. Blockchain in the quantum era: Surveying security challenges and post-quantum cryptography. In *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2218–2223.
- [39] Surya Rao Rayarao. 2025. F-tests: A comprehensive review of theory, applications, and statistical inference. *Authorea* 2025, 0711 (2025). doi:10.22541/au.175225954.47914284/v1

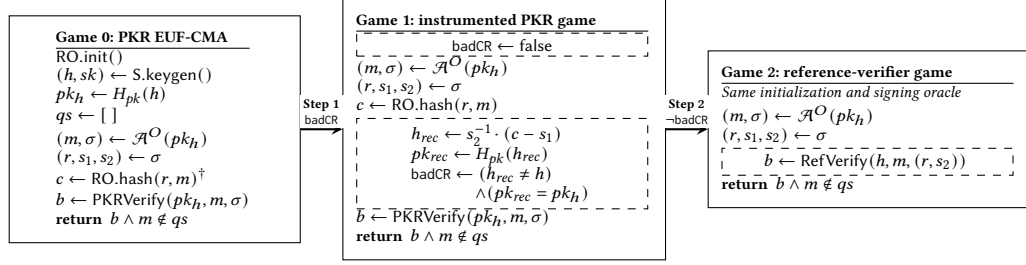
- [40] Nalavala R. Reddy, Supriya Suryadevara, K. Guru Raghavendra Reddy, Ramisetty Umamaheswari, Ramakrishna Guttula, and Rajitha Kotoju. 2025. Quantum-secured blockchain framework for enhancing post-quantum data security. *Scientific Reports* 15, 1 (2025), 31048. doi:10.1038/s41598-025-16315-8
- [41] K. Revathi and K. Suganthi. 2025. Enhancing blockchain security against quantum threats through integration of post-quantum cryptographic algorithms. *Computers and Electrical Engineering* 127 (2025), 110610. doi:10.1016/j.compeleceng.2025.110610
- [42] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (Feb. 1978), 120–126. doi:10.1145/359340.359342
- [43] Bjorn Oude Roelink, Mohammed El-Hajj, and Dipti Sarmah. 2024. Systematic review: Comparing ZK-SNARK, ZK-STARK, and bulletproof protocols for privacy-preserving authentication. *Security and Privacy* 7, 5 (2024), e401. doi:10.1002/spy2.401
- [44] Ben Salter, Adam Raine, and Daniel Van Geest. 2024. Use of the ML-DSA signature algorithm in the Cryptographic Message Syntax (CMS). Report: <https://www.ietf.org/archive/id/draft-salter-lamps-cms-ml-dsa-00.html>.
- [45] Bora B. Sezer, Sedat Akleylek, and Urfat Nuriyev. 2025. PP-PQB: Privacy-preserving in post-quantum blockchain-based systems: A systematization of knowledge. *IEEE Access* 13 (2025), 41382–41405. doi:10.1109/ACCESS.2025.3545943
- [46] Rasha Shajahan, Kurunandan Jain, and Prabhakar Krishnan. 2024. A survey on NIST 3rd round post-quantum digital signature algorithms. In *2024 5th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI)*. 132–140.
- [47] Peter W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (1997), 1484–1509. doi:10.1137/S0097539795293172
- [48] Victor Shoup. 2004. Sequences of games: A tool for taming complexity in security proofs. Cryptology ePrint Archive, Paper 2004/332. <https://eprint.iacr.org/2004/332>
- [49] SPHINCS+ Team. 2026. SPHINCS+. Software repository. <https://github.com/sphincs/sphincsplus>
- [50] TlsfuzzerGithub. 2025. *Pure-Python ECDSA signature/verification and ECDH key agreement*. <https://github.com/tlsfuzzer/python-ecdsa>
- [51] Dominique Unruh. 2019. Quantum relational Hoare logic. *Proc. ACM Program. Lang.* 3, POPL, Article 33 (Jan. 2019), 31 pages. doi:10.1145/3290346
- [52] Zooko Wilcox. 2020. *Zcash: The Design of the Ceremony*. Technical Report. <https://www.utwente.nl/en/ces/sal/exams/Blockchain-and-Distributed-Ledger-Technology-test/3-Privacy/zcash-the-design-of-the-ceremony.pdf>
- [53] Pieter Wuille. 2026. *Libsecp256k1*. <https://github.com/zcash/zcash/tree/master/src/secp256k1>
- [54] Pieter Wuille, Jonas Nick, and Tim Ruffing. 2020. *BIP 340: Schnorr Signatures for secp256k1*. Technical Report. <https://bips.dev/340/>

A Open Science

To support reproducibility and double-blind evaluation, we provide an anonymous artifact repository containing the EasyCrypt formalisation, the Falcon-PKR implementations and benchmarks, and the SECUER-based evaluation workflow. It includes dependencies, build and reproduction instructions, experiment scripts, and a mapping from paper claims to artifact components. Anonymous link: <https://anonymous.4open.science/r/PQMZ-FECD>.

B Game Hops and Reduction to Falcon EUF-CMA

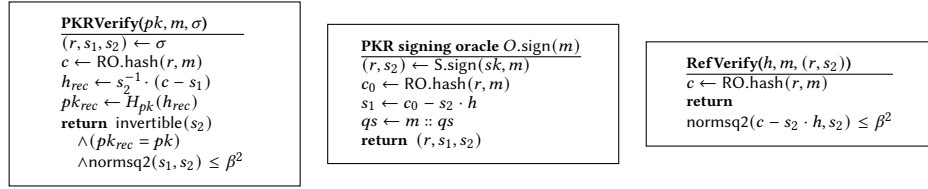
This appendix presents the games and reductions underlying Theorem 1. They correspond to the EasyCrypt modules EUF_CMA_PKR, G_PKR_CR, G_mid_ref, Red_CR, and Red2, which are mechanically checked in the formalisation.



Games 0–2 share the PKR signing oracle $O.\text{sign}$.
Games 0–1 use PKRVerify; Game 2 uses RefVerify.

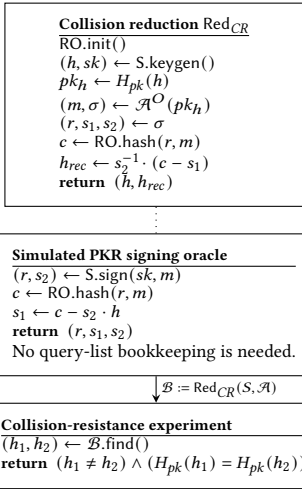
Probability split in Game 1:

$$\Pr[\text{res}] \leq \Pr[\text{res} \wedge \neg \text{badCR}] + \Pr[\text{badCR}]$$



$$\begin{aligned}
\Pr[\text{G}_{\text{PKR}} : \text{res}] &= \Pr[\text{G}_{\text{PKR-CR}} : \text{res}] \\
&\leq \Pr[\text{G}_{\text{mid-ref}} : \text{res}] + \Pr[\text{G}_{\text{PKR-CR}} : \text{badCR}] \\
&\leq \text{Adv}_{\text{Std}}^{\text{EUF-CMA}}(\text{Red}_{\sigma}(\mathcal{A})) + \text{Adv}_{H_{pk}}^{\text{CR}}(\text{Red}_{CR}(\mathcal{A})).
\end{aligned}$$

Step 3—badCR branch: collision resistance



Step 4—reference game: standard Falcon EUF-CMA

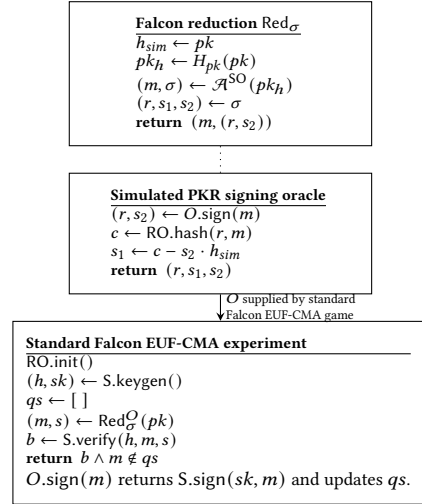


Figure 3: Full game code underlying the four-step proof of Fig. 2, derived from our development. *Top panel*: the games of Steps 1–2, with dashed boxes marking statements introduced or changed relative to the preceding game, and the shared oracle/verifier sub-procedures below each. [†]This oracle call is textually present in EUF_CMA_PKR but its result is unused there; it becomes load-bearing once instrumented as badCR in G_PKR_CR. *Bottom-left panel (Step 3)*: the reduction Red_CR to collision resistance of H_{pk}, plugged in as the adversary B of CR_PKHash. [‡]Unlike the oracle in the top panel, Red_CR.O.sign performs no query-list bookkeeping, since the collision-resistance game has no freshness condition to check. *Bottom-right panel (Step 4)*: the reduction Red₂, which simulates G_mid_ref's oracle for A via SimOracle while forwarding signing queries to the standard oracle O of EUF_CMA_Std.