CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Further Application of Progressive Verification

Master's thesis in Computer science and engineering

Mia Heljeberg, Arvid Nyberg

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Further Application of Progressive Verification

Mia Heljeberg, Arvid Nyberg

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Further Application of Progressive Verification
Mia Heljeberg, Arvid Nyberg

Supervisor: Elena Pagnin, CSE
Advisor: Pontus Hanssen, Omegapoint
Examiner: Rhouma Rhouma, CSE

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Further Application of Progressive Verification
Mia Heljeberg, Arvid Nyberg
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

A digital signature is a fundamental cryptographic primitive that provides authenticity and integrity by allowing anyone with a public key to verify that a message was produced by a signer with a corresponding secret key. Such verifications typically produce a binary output only when the process is finished. In contrast, progressive verification (PV) performs verification in smaller incremental steps, gradually building confidence in the signature's validity over the course of the process. Progressive verification offers several key advantages for post-quantum cryptographic (PQC) schemes on resource constrained devices as it allows for early rejection of invalid inputs and supports adjustable soundness (allowing for a trade-off between security and efficiency). Furthermore, PV can shrink the public key size which addresses a common challenge of PQC schemes.

This thesis explores the design and applicability of PV on post-quantum secure digital signature schemes currently involved in the NIST PQC standardisation process. The approach utilises a compiler framework developed by Boschini et al [1] which transforms matrix-vector based (**Mv**-style) verifications into progressive ones. We explore whether this approach extends to further multivariate quadratic (MQ) schemes as well as to code based schemes. In addition, we investigate whether the compiler can be applied to zero-knowledge proofs, thereby addressing the broader applicability of progressive verification beyond digital signatures. By identifying the matrix-vector structure in the schemes and analysing how the compiler interacts with the verification steps, we assess correctness preservation, security aspects, and practical feasibility.

Our findings show that the PV compiler applies cleanly to the MQ-based scheme Unbalanced Oil and Vinegar (UOV), enabling gradual verification without modifying the signing or key-generation algorithms. For code-based schemes, we demonstrate that PV is not applicable to the Codes and Restricted Objects Signature Scheme (CROSS), despite it containing a matrix-vector multiplication in the verification. Progressive verification was also shown to be partially applicable to the verification of a zero-knowledge proof.

Overall, this thesis expands the set of post-quantum digital signature schemes known to support progressive verification and highlights design features that make a scheme compatible with PV. These insights can guide both future implementations of PV and the development of new PQC schemes intended for constrained environments.

Keywords: Progressive Verification, Post-quantum Cryptography, Digital Signature, Zero-knowledge Proof, All-or-nothing Verification.

Acknowledgements

We would like thank our supportive and dedicated academic supervisor, Elena Pagnin, for her help and guidance throughout this project. We also extend our sincere gratitude for the opportunity to conduct this thesis in collaboration with Omegapoint, and in particular to our industrial supervisor, Pontus Hanssen, for his support and encouragement.

Mia Heljeberg & Arvid Nyberg, Gothenburg, 2026-02-07

Contents

Abbreviations

PV	Progressive Verification
PQC	Post-quantum Cryptography
NIST	National Institute of Standards and Technology
MQ	Multivariate Quadratic
PPT	Algorithms that are probabilistic and run in polynomial time
H	A parity-check matrix which defines a set of linear consistency checks that ensure that multiplying a codeword by $\mathbf{H}^\mathsf{T}$ produces a zero if the codeword is valid.
Syndrome	Vector computed by multiplying an error vector ($\mathbf{e}$) by $\mathbf{H}^\mathsf{T}$.

List of symbols

pk	Public Key
sk	Secret Key
μ	Message
σ	Signature
α	Confidence Level
$\mathbb{E}$	A cyclic subgroup which is a specific set of elements, generated by the public element g, within a larger multiplicative group.
$\star$	Component-wise multiplication
$\mathbb{Z}$	A ring which is an algebraic structure consisting of a set with two operations (typically addition and multiplication) that satisfy properties similar to those of integers.

1

Introduction

With the rapid evolution of the Internet of Things (IoT) and smart-city infrastructures, resource-constrained devices are becoming central to our digital lives. These systems might have limited computing power, memory or run on battery. They might run under tight time constraints or be subjected to harsh physical conditions. Even so, these systems handle sensitive data and must ensure integrity, authenticity, and confidentiality through robust verification.

Meanwhile, the looming threat of quantum computers jeopardises the classical cryptographic algorithms that currently protect our data. This adds another layer of complexity, as post-quantum cryptographic (PQC) schemes tend to be larger and more computationally intensive. In this context, designing algorithms that can run efficiently on embedded devices becomes increasingly urgent.

The National Institute of Standards and Technology of the U.S. Department of Commerce (NIST) describes quantum computers as devices that exploit principles of quantum mechanics to process information [2]. A central distinction from classical computing lies in the representation of data. Classical bits assume a single value at a given time, however quantum bits can exist in a superposed state, which allows bits to represent both 0 and 1 simultaneously.

This capability enables quantum processors to evaluate an enormous number of possible solutions in parallel, allowing certain computations that are infeasible for classical computers to be performed efficiently. Consequently, cryptographic constructions whose security is based on computational hardness assumptions would no longer offer adequate protection against quantum computers. This motivates the development of post-quantum cryptographic algorithms, which aims to develop schemes that preserve security even when adversaries have access to quantum computers. These algorithms are built upon mathematical problems believed to be resistant even to quantum attacks, ensuring that they can replace classical cryptographic algorithms that are vulnerable to quantum threats.

As part of the counter-effort to the quantum threat, NIST initiated a process to develop standardised post-quantum cryptosystems [3]. Since its start in 2016, many cryptosystems have been proposed, reviewed, and broken. A few have survived scrutiny (thus far) and been selected for standardisation.

One of the cryptographic primitives currently being standardised by NIST are digital signatures. The verification procedures of digital signatures are typically mono-

lithic, meaning that a binary validity decision (accept/reject) is returned only upon completion of the entire procedure. This means that if the verification process is interrupted, nothing can be said about the validity of the given signature.

This thesis explores the design and application of progressive verification (PV), in which verification happens in incremental steps, each increasing the verifier's confidence in the result. Unlike standard binary verification, PV enables early rejection of invalid inputs and supports adjustable soundness, allowing the verifier to accept a valid signature once a desired confidence level has been reached, even if this does not correspond to absolute certainty. Adjustable soundness allows earlier acceptance of valid signatures in scenarios where full certainty is unnecessary, providing increased flexibility in resource-constrained environments. Together, early rejection and adjustable soundness make quantum-proof verification more practical on low-power and resource-constrained devices.

1.1 Aim

The overarching goal is to enhance the flexibility and practical utility of cryptographic primitives in diverse computational environments, with an emphasis on the resource-constrained context of embedded and IoT devices, thereby addressing a key challenge of post-quantum security. The contribution of this thesis will be to explore the applicability of progressive verification for post-quantum secure digital signatures.

Progressive verification, performing verification in incremental steps, offers a way to make certain cryptographic systems more feasible where there is a substantial risk of an interruption. This thesis will explore the applicability of PV on existing post-quantum digital signatures from the NIST PQC standardisation process, focusing on multivariate quadratic (MQ) and code-based schemes.

Moreover, the potential of progressive verification will be attempted to be expanded to ZKPs. Applying progressive verification to ZKPs can demonstrate its versatility and a broader applicability across diverse cryptographic schemes.

1.2 Problem Formulation

This thesis builds on the work by Boschini et al. [1], focusing on signature schemes that include a matrix-vector multiplication in the verification, which they refer to as **Mv**-style verifications. For these schemes, they construct a compiler that transforms monolithic verification algorithms into progressive ones. This way, the remaining algorithms in the schemes, such as key generation and signing, are left unchanged, and instead of PV constituting a primitive in itself, it can be accomplished by simply applying the compiler to existing schemes.

In [1], it is shown that the transformation applies to multiple lattice- and multivariate equation–based post-quantum schemes. Among the schemes addressed are Rainbow, MAYO and LUOV, all of which have been proposed schemes in NISTs

ongoing post-quantum cryptography standardisation process, which is at the forefront of developing quantum-resistant cryptographic schemes. However, Rainbow and LUOV have since been removed from the process. In fact, as of 2025, no digital signature schemes remain among the cryptosystems still in review in the fourth round, and although three digital signature schemes have been selected for standardisation, two of these rely on the same foundation, namely structured lattices. To diversify their portfolio, NIST issued a request for additional digital signature schemes [4], running in parallel with the original process, which is now in its second round of review.

To expand the work by Boschini et al. [1], this thesis shows how to apply their compiler to UOV, one of the multivariate quadratic schemes still in the NIST process (the others are QR-UOV and SNOVA). Furthermore, it investigates possible application on code-based schemes by ways of CROSS, one of two such schemes in the NIST process (the other one is LESS), something that was not touched upon by [1]. Finally, this thesis examines an application of the compiler on other cryptographic primitives than digital signatures, which was the sole focus of [1]. This will entail application of PV on a concrete zero-knowledge proof scheme [5].

1.3 Research Questions

1. To what extent is the progressive verification compiler from [1] applicable to the digital signature schemes we have chosen from the NIST process?
   (a) Can the compiler be applied to multivariate quadratic digital signature schemes, other than the ones treated by [1]?
   (b) Can the compiler be applied to code-based digital signature schemes?
2. Can the progressive verification compiler from [1] be applied to zero-knowledge proof schemes?

2

Background

This chapter outlines the prerequisite knowledge of the main topics discussed in the thesis. The chapter opens by introducing the notation used throughout the report and then proceeds to explain key concepts, such as progressive verification and the cryptographic primitives considered.

2.1 Notation

Throughout the paper vectors are denoted by bold, lower-case letters and matrices by bold, upper-case letters. $\mathbf{v}[i]$ is used to identify the i-th entry of a vector $\mathbf{v}$, and $\mathbf{A}[i, j]$ to identify the entry in the i-th row and j-th column of a matrix $\mathbf{A}$. The transpose of a matrix is denoted by $\mathbf{A}^{\mathsf{T}}$, while $\mathbf{I}$ denotes the identity matrix. The symbol $\star$ denotes component-wise multiplication. Throughout the paper, Σ satisfies the properties of correctness and existential unforgeability.

2.2 Digital Signatures

Digital signatures provide mechanisms for message authentication and integrity protection in the digital setting. By binding a signer's secret key to the message content though a mathematical construction, such schemes allow a verifier to detect any modification of the signed data and prevent adversaries from producing a valid signature.

The idea of a digital signature scheme was first described by Diffie and Hellman in 1976 [6], where they speculated that such schemes existed based trapdoor one-way permutation functions. Shortly afterward, Rivest, Shamir and Adleman introduced the RSA algorithm, which could be used to produce primitive digital signatures [7]. Today, with the anticipated rise of quantum computing, research is increasingly focused on post-quantum digital signature schemes that remain secure against quantum attacks.

A classical digital signature scheme consists of three probabilistic polynomial-time algorithms. These are KeyGen, Sign and Ver. These three algorithms define the basic structure of all digital signature schemes.

KeyGen is an algorithm that generates a pair of keys consisting of a public key pk which is distributed to verifiers, and a secret key sk which is kept exclusively by

the signer. Given the secret key and a message μ, the signer computes a signature $\sigma = \mathsf{Sign}(\mathsf{sk}, \mu)$. The signature encodes evidence that the signer, who knows the secret key, approved the message. The third algorithm is Ver, given the public key, a message and a signature, the verifier check $\mathsf{Ver}(\mathsf{pk}, \mu, \sigma) \in \{0, 1\}$. Verification outputs 1 if the signature is valid for the message under the public key, and 0 otherwise.

A signature scheme must satisfy two properties, correctness and unforgeability. Correctness guarantees that a valid signature will always verify successfully. Unforgeability ensures that no adversary can produce a valid signature for a new message without knowing the secret key.

2.3 Zero-Knowledge Proofs

A zero-knowledge proof (ZKP) is a cryptographic primitive in which one party (the prover) can convince another party (the verifier) that they know a secret without revealing what the said secret is or any additional information.

ZKPs date back to the 1980s and to a paper titled "The knowledge of complexity of interactive proof-systems" by Goldwasser et al. [8]. The authors described a concept called "Interactive protocols" in which a prover and a verifier could communicate back and forth in order to convince the verifier that the prover had knowledge of certain information. While this laid the foundation for zero-knowledge proofs, the interaction proved to be both resource-intensive and time-consuming, which lead to the development of non-interactive zero-knowledge proofs.

In 1986 Fiat and Shamir invented a technique to transform an interactive proof of knowledge into a digital signature [9]. This technique is known as the Fiat-Shamir heuristic. It transforms interactive ZKPs into non-interactive ones, which laid the foundation for more widespread and scalable use of zero-knowledge proofs.

2.4 Progressive Verification

Digital signatures, message authentication codes (MACs), and zero-knowledge proofs are examples of cryptographic primitives that involve verification mechanisms to ensure authenticity of some input. These verifications typically return a binary validity decision (accept or reject) only upon completion of the entire procedure. This means that if the verification process is interrupted, no more can be said about the validity of the input than if nothing were done at all. Fischlin [10] calls this *all-or-nothing verification*, whereas Boschini et al. [1] use the word *monolithic* to describe these procedures.

In contrast to an all-or-nothing verification, *progressive verification* allows for early rejection of invalid inputs, at the same time as it *progressively* (gradually) builds confidence in (the validity of) the input over the course of the verification process. Thus, if at any point the verification process is stopped, it can either reject the input altogether or, if no error has been found yet, output an estimate of the confidence

that the input is correct. Another way to think of it is that, given an invalid input, the likelihood of it being rejected gradually grows, and with it the confidence in an accepting decision.

Motivations for Progressive Verification

Progressive verification offers advantages in settings with constraints on memory, computing power, energy, and time. Firstly, it allows the verifier to dynamically adapt the soundness to available resources or specific requirements. An example given in [1] is a smart device that adjusts the desired soundness level based on the source of the signature or the current battery level. This allows for a trade-off between computational effort and confidence in the result. Secondly, in particularly unpredictable environments where interruptions are common, progressive verification adds resilience by providing meaningful partial results even if the verification process is unexpectedly halted. Unlike traditional all-or-nothing verification, which yields no useful output if interrupted, progressive verification allows the verifier to probabilistically quantify the validity based on the computation done so far. Additionally, progressive verification can speed up the verification process on average by identifying and rejecting invalid inputs early.

2.4.1 Digital Signatures with Progressive Verification

Definition 1 (Digital signatures with progressive verification)**.** A digital signature with progressive verification consists of a 4-tuple of probabilistic polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{ProgVer})$ such that:

- $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ is a correct digital signature scheme.
- the progressive verification algorithm $\mathsf{ProgVer}$ takes as input a public (verification) key pk, a message μ, a signature σ, and some interruption parameter t, and outputs either some real number $\alpha \in [0, 1]$, or the special symbol $\perp$.[1]
- (*Correctness*) if $\mathsf{ProgVer}(\mathsf{pk}, \mu, \sigma, t)$ outputs $\perp$, then $\mathsf{Ver}(\mathsf{pk}, \mu, \sigma) = 0$.
- (*Security*) if $\mathsf{ProgVer}(\mathsf{pk}, \mu, \sigma, t)$ outputs $\alpha \in [0, 1]$, then $\Pr[\mathsf{Ver}(\mathsf{pk}, \mu, \sigma) = 1] \geq \alpha$.

Simply put, if the progressive verification algorithm $\mathsf{ProgVer}$ outputs $\perp$, the signature is invalid (with 100% certainty). The correctness condition above states that if a signature is rejected by $\mathsf{ProgVer}$, then it would also be rejected by Ver, and is therefore an invalid signature. Or equivalently, *progressive verification never rejects valid signatures*. On the other hand, if the output is $\alpha \in [0, 1]$, the algorithm considers the signature valid, and α represents the confidence level. Specifically, *the signature is valid with probability at least* α (and invalid with probability at most $1-\alpha$). If $\alpha = 0$, $\mathsf{ProgVer}$ has no information on the validity of the signature, whereas for $\alpha = 1$, the signature is valid with certainty.

[1] Whereas [1, 11, 12] allow α to take the value $\perp$, we let α denote only an output that is a number in the interval $[0, 1]$, and let $\perp$ be a separate symbol. This makes it more straightforward to reason about α as a quantity.

Extensions by Boschini et al.

In addition to detailing the properties of progressive verification as a black-box procedure in a fashion similar to [11, 12], Boschini et al. [1] extend the definition with a description of how it should work internally.

Specifically, a progressive verification is composed of a number of individual procedures that each perform an independent check on the input, gradually building confidence with each passing check. We denote these subroutines $\mathsf{ProgStep}_i$ for $i = 0$ to T. If any one step fails, the procedure immediately returns $\perp$, indicating rejection of the input. Conversely, assuming that all of the first i steps succeed, the procedure outputs a value determined by a function $\alpha(i) \in [0, 1]$. Viewing verification as a sequence of steps does not restrict the generality, since the conventional verification algorithm Ver can be interpreted as a trivial instance of progressive verification where $T = 0$, consisting of a single verification step Ver.

Boschini et al. model interruptions with a variable t passed as a parameter to $\mathsf{ProgVer}$. One reason for this is to explicitly demonstrate the fact that individual steps work without "knowing" when to stop, which is necessary to model arbitrary interruptions. However, one might argue that this must also be the case for the overarching verification procedure $\mathsf{ProgVer}$. Another reason might be to facilitate formal reasoning around the point of interruption t, which is not as necessary for the purposes of this work as it is for [1]. To facilitate understanding, we both present the more general representation of the framework (Fig. 2.1a), as well as a version that is closer to what an implementation might look like (Fig. 2.1b).

$\mathsf{ProgVer}(\mathsf{pk}, \mu, \sigma, \mathsf{st}, t)$

1: **if** $t < 1$ **return** 0
2: $n \leftarrow \min(t, T)$
3: **for** $i = 0, \ldots, n$
4: $\quad (b, \mathsf{st}) \leftarrow \mathsf{ProgStep}_i(\mathsf{pk}, \mu, \sigma, \mathsf{st})$
5: $\quad$ **if** $b = 0$ **return** $\perp$
6: **return** $\alpha(n)$

(a) $\mathsf{ProgVer}$ with an interruption parameter t, similar to [1].

$\mathsf{ProgVer}(\mathsf{pk}, \mu, \sigma, \mathsf{st})$

1: $r \leftarrow 0$
2: **for** $i = 0, \ldots, T$
3: $\quad (b, \mathsf{st}) \leftarrow \mathsf{ProgStep}_i(\mathsf{pk}, \mu, \sigma, \mathsf{st})$
4: $\quad$ **if** $b = 0$ **return** $r \leftarrow \perp$
5: $\quad$ **else** $\quad r \leftarrow \alpha(i)$
6: **return** r

(b) An implementation-like version of $\mathsf{ProgVer}$.

Figure 2.1: A generic progressive verification algortihm $\mathsf{ProgVer}$ as a sequence of steps $\mathsf{ProgStep}_i$. In contrast to [1], the result here is $\alpha = 0$ if the procedure is interrupted before any computation has been done, and $\perp$ only once an error is found during a $\mathsf{ProgStep}_i$, thus staying true to the correctness condition of Definition 1.

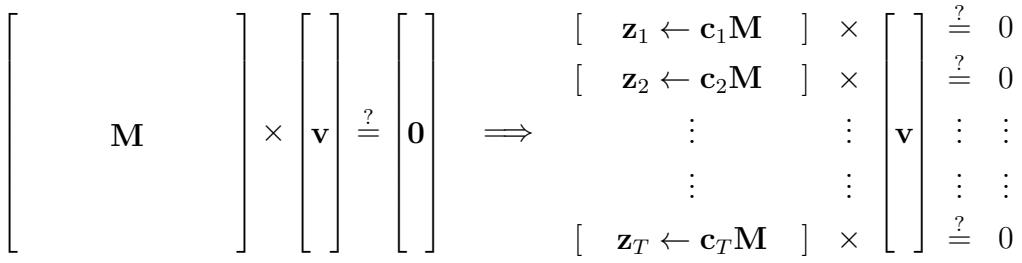

Figure 2.2: An illustration of the main idea of the **Mv**-style progressive verification compiler. A matrix–vector multiplication check is transformed to vector–vector multiplication checks.

2.4.2 Compiler for Mv-style Signature Verifications

Boschini et al. present what they call a *compiler*; a framework for transforming the all-or-nothing verification algorithms of existing signature schemes into progressive ones. The compiler targets schemes featuring what they refer to as **Mv**-style verification, which can be understood as a combination of two types of checks: a matrix–vector multiplication (ensuring $\mathbf{Mv} \stackrel{?}{=} \mathbf{0}$ for a suitable matrix $\mathbf{M}$ and vector $\mathbf{v}$), and a set of auxiliary checks.

The technique reformulates the matrix–vector product in the following manner: Given a matrix–vector multiplication $\mathbf{A} \cdot \boldsymbol{\sigma} = \mathbf{u}$, we collect all terms on the left-hand side to obtain $\mathbf{A} \cdot \boldsymbol{\sigma} - \mathbf{u} = \mathbf{0}$. Now, the expression $\mathbf{A} \cdot \boldsymbol{\sigma} - \mathbf{u} = \mathbf{A} \cdot \boldsymbol{\sigma} - \mathbf{I} \cdot \mathbf{u}$ can be rewritten in block matrix form as

$$\begin{bmatrix}\mathbf{A} & -\mathbf{I}\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{\sigma} \\ \mathbf{u}\end{bmatrix} = \mathbf{0}. \qquad \text{(Eq. 2.1)}$$

This follows from the properties of block matrix multiplication, where the elements themselves are matrices or vectors; the multiplication rules for block matrices are analogous to those for scalar elements. This fact is important to keep in mind when applying this technique, as it often involves concatenating vectors or matrices to form the components described here. Now, by substituting

$$\mathbf{M} = \begin{bmatrix}\mathbf{A} & -\mathbf{I}\end{bmatrix} \quad \text{and} \quad \mathbf{v} = \begin{bmatrix}\boldsymbol{\sigma} \\ \mathbf{u}\end{bmatrix},$$

in Eq. 2.1, from the original equation $\mathbf{A} \cdot \boldsymbol{\sigma} = \mathbf{u}$ we have derived the matrix–vector multiplication $\mathbf{Mv} = \mathbf{0}$ implied by "**Mv**-style" verifications. Multiplying both sides with a vector $\mathbf{c}$ yields $\mathbf{c} \cdot \mathbf{M} \cdot \mathbf{v} = \mathbf{c} \cdot \mathbf{0}$, which simplifies to $(\mathbf{c} \cdot \mathbf{M}) \cdot \mathbf{v} = 0$. By substituting $\mathbf{z} = \mathbf{c} \cdot \mathbf{M}$, we see that the matrix multiplication has been reduced to a dot product $\mathbf{z} \cdot \mathbf{v} = 0$.

A concrete visualisation of this process, using toy matrices populated with random integers, is provided in Appendix A.

With this background, the idea of the compiler is to achieve progressiveness by replacing the monolithic matrix–vector multiplication with T vector–vector multiplications

involving random linear combinations of the columns of $\mathbf{M}$, each check gradually building confidence in the signature. This idea is illustrated in Fig. 2.2. To apply the compiler in practice, the matrix $\mathbf{A}$ (typically derived from pk) and the vectors $\boldsymbol{\sigma}$ (the signature) and $\mathbf{u}$ (derived from the message) are first identified. The procedures GetM and GetV construct $\mathbf{M}$ and $\mathbf{v}$, respectively, as previously described. The procedure GetZ then generates T linearly independent vectors $\mathbf{z}_1, \ldots, \mathbf{z}_T$ by multiplying $\mathbf{M}$ with independently sampled random vectors $\mathbf{c}$ as detailed in Fig. 2.3. This setup is carried out in the initial step $\mathsf{ProgStep}_0$. The subsequent steps $\mathsf{ProgStep}_i$ each perform a single vector–vector multiplication check using the precomputed vectors $\mathbf{z}_i$. Any auxiliary checks beyond these are encapsulated in a subroutine Check and handled during $\mathsf{ProgStep}_0$.

$\mathsf{ProgStep}_0(\mathsf{pk}, \mu, \sigma, \mathsf{st})$

1 : $\mathbf{M} \leftarrow \mathsf{GetM}(\mathsf{pk})$
2 : $\mathbf{Z} \leftarrow \mathsf{GetZ}(\mathbf{M}, T)$
3 : $\mathbf{v} \leftarrow \mathsf{GetV}(\mu, \sigma)$
4 : $b \leftarrow \mathsf{Check}(\mathsf{pk}, \mu, \sigma)$
5 : **return** $(b, (\mathbf{Z}, \mathbf{v}))$

$\mathsf{ProgStep}_i(\mathsf{pk}, \mu, \sigma, \mathsf{st})$

1 : **parse** $\mathsf{st} = (\mathbf{Z}, \mathbf{v})$
2 : **if** $\mathrm{row}_i(\mathbf{Z}) \cdot \mathbf{v} = 0$
3 : **return** $(1, \mathsf{st})$
4 : **else**
5 : **return** $(0, \mathsf{st})$

$\mathsf{GetZ}(\mathbf{M}, T)$

1 : **for** $j = 1, \ldots, T$
2 : $\mathbf{c}_j \leftarrow\!\$\ \mathbb{Z}_q^{1 \times \mathrm{rows}(\mathbf{M})}$
3 : $\mathbf{z}_j \leftarrow \mathbf{c}_j \cdot \mathbf{M}$
4 : // ensure $\{\mathbf{z}_1, \ldots, \mathbf{z}_j\}$ // linearly independent
5 : **return** $\begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_T \end{bmatrix}$

Figure 2.3: The internals of the procedures $\mathsf{ProgStep}_0$ and T subsequent steps $\mathsf{ProgStep}_i$ used in progressive **Mv**-style signature verification. The subroutine GetZ lies at the heart of the PV compiler. It takes as input the matrix $\mathbf{M}$ that would normally be used in an **Mv**-style verification check, and transforms it into T vectors — random and linearly independent combinations of the information contained in $\mathbf{M}$ — to be used in $\mathsf{ProgStep}_i$ for $i = 1 \ldots T$.

Efficiency

The construction just described achieves progressive verification in the sense that if the process is unexpectedly interrupted it still provides meaningful information about the input. Nonetheless, recall from Section 2.4 that beyond providing partial results, progressive verification also enables a computation–soundness trade-off and, on average, accelerates verification compared to the monolithic approach. Now, the trade-off is inherently tied to the number of progressive steps T. However, to realise a meaningful speed-up and, more broadly, to ensure practical feasibility, the computational overhead introduced by progressive verification must not significantly exceed that of a monolithic verification approach. Thus, progressive verification must remain efficient for each message–signature pair.

As a first step towards efficiency, note that the expensive matrix–vector multiplications are contained in the first two lines of $\mathsf{ProgStep}_0$ (Fig. 2.3), and that they are dependent only on the public key, and not any specific message–signature pair. We

can use this fact and reduce overhead by re-using $\mathbf{Z}$ to verify multiple signatures from the same signer.

One way to implement this is in line with the *efficient verification* paradigm of [1], to decouple the computation of $\mathbf{Z}$ as a separate procedure (called OffVer in [1]), an offline setup independent of the arrival of a message–signature pair to verify. However, to keep the implementation in line with Definition 1, we can simply conditionally decide to (re-)compute $\mathbf{Z}$ based on some condition as in Fig. 2.4.

Security relies on the adversary having no information about the randomness sampled by GetZ. For this reason, $\mathbf{Z}$ can alternatively be referred to as a *secret verification key*, abbreviated as svk. In [1], svk includes additional components, but $\mathbf{Z}$ is the primary one.

To illustrate the difference between an efficient implementation and one where svk needs to be recomputed for every new message–signature pair, in Section 3.1 we show two different ways to structure the **Mv**-style verification check of UOV.

$\mathsf{ProgStep}_0(\mathsf{pk}, \mu, \sigma, \mathsf{st})$

1 : **if** some condition
2 : $\quad \mathbf{M} \leftarrow \mathsf{GetM}(\mathsf{pk})$
3 : $\quad \mathbf{Z} \leftarrow \mathsf{GetZ}(\mathbf{M}, T)$
4 : $\mathbf{v} \leftarrow \mathsf{GetV}(\mu, \sigma)$
5 : $b \leftarrow \mathsf{Check}(\mathsf{pk}, \mu, \sigma)$
6 : **return** $(b, (\mathbf{Z}, \mathbf{v}))$

Figure 2.4: Efficient progressive verification. Efficiency comes from reusing $\mathbf{Z}$ (sometimes referred to as the verification key svk) in subsequent verifications and only periodically recomputing it (lines 2 and 3). For security, svk must be kept confidential between verifications.

2.5 UOV

Since it was first introduced in 1999 [13], UOV has proven itself over two decades of scrutiny, attesting to its strong security and sustained reliability. Furthermore, the simplicity in the UOV design has made it a strong foundation for other derivative schemes. UOV is an example of a multivariate quadratic (MQ) cryptosystem, a family of candidate post-quantum cryptographic schemes in which the public/secret key pair is composed of multivariate quadratic polynomials:

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \sum_{i=1}^{n} \sum_{j=i}^{n} p_{i,j} x_i x_j + \sum_{i=1}^{n} p_i x_i + p_0$$

The public key $\mathcal{P}$ consists of a series of m such polynomials in n variables, i.e. a non-linear map from $\mathbb{F}_q^n$ to $\mathbb{F}_q^m$:

$$\mathcal{P}(\mathbf{x}) = \begin{bmatrix} p_1(\mathbf{x}) \\ \vdots \\ p_m(\mathbf{x}) \end{bmatrix} \in \mathbb{F}_q^m, \qquad \mathbf{x} \in \mathbb{F}_q^n$$

In cryptographic applications, the map $\mathcal{P}$ should function as a trapdoored one-way function, equipped with a trapdoor td. Specifically, evaluating $\mathcal{P}$ (i.e. evaluating m polynomials) is computationally efficient. Conversely, inverting $\mathcal{P}$ should be computationally infeasible, without knowledge of some trapdoor information td. With the key pair $(\mathsf{pk} = \mathcal{P}, \mathsf{sk} = \mathsf{td})$ we can construct a digital signature scheme by the hash-and-sign paradigm: To sign, we compute $\mathbf{h} \leftarrow \mathsf{Hash}(\mu) \in \mathbb{F}_q^m$. Then, using the trapdoor td, we find a preimage element $\boldsymbol{\sigma} \in \mathbb{F}_q^n$ of $\mathbf{h}$ under $\mathcal{P}$ to serve as the signature:[2]

$$\boldsymbol{\sigma} = \mathcal{P}^{-1}(\mathsf{Hash}(\mu))$$

Finally, given the public key $\mathcal{P}$ and a message/signature pair $(\mu, \boldsymbol{\sigma})$, the verifier simply checks whether $\mathcal{P}(\boldsymbol{\sigma}) = \mathsf{Hash}(\mu)$:

$$\mathcal{P}(\boldsymbol{\sigma}) \stackrel{?}{=} \mathsf{Hash}(\mu) \tag{Eq. 2.2}$$

For the purposes of this thesis, this short overview of MQ cryptography is sufficient, and we now turn our attention to the verification of UOV, as that is what is relevant for the compiler. For further details on the complete UOV scheme, we refer the reader to [14].

2.5.1 UOV Verification

UOV is an MQ scheme in which the polynomials are homogeneous, meaning they consist solely of quadratic terms:[3]

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \sum_{i=1}^{n} \sum_{j=i}^{n} p_{i,j} x_i x_j. \tag{Eq. 2.3}$$

Each homogeneous quadratic polynomial can be represented by a single upper triangular matrix $\mathbf{P}$, where elements $p_{i,j}$ correspond to the coefficients of the polynomial. Equation 2.3 can then be expressed as $p(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{P}\mathbf{x}$ and the MQ verification check of Equation 2.2 takes the form

$$\mathcal{P}(\boldsymbol{\sigma}) = \begin{bmatrix} \boldsymbol{\sigma}^\mathsf{T}\mathbf{P}_1\boldsymbol{\sigma} \\ \vdots \\ \boldsymbol{\sigma}^\mathsf{T}\mathbf{P}_m\boldsymbol{\sigma} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}^\mathsf{T}\mathbf{P}_1 \\ \vdots \\ \boldsymbol{\sigma}^\mathsf{T}\mathbf{P}_m \end{bmatrix} \cdot \boldsymbol{\sigma} \in \mathbb{F}_q^m. \tag{Eq. 2.4}$$

[2] Although $\mathcal{P}$ is (in this case) not a bijection (i.e. $n \neq m$), and thus by definition not invertible, we can borrow the terminology and notation of inverse functions and write $\boldsymbol{\sigma} = \mathcal{P}^{-1}(\mathbf{h})$.

[3] For reference, homogeneous polynomials are in the literature sometimes referred to as "forms".

Equation 2.3 can alternatively be expressed as the dot product $p(\mathbf{x}) = \mathbf{p} \cdot \mathbf{x}'$ of a coefficient vector $\mathbf{p}$ and a quadratic term vector $\mathbf{x}'$ (all pairs of terms from $\mathbf{x}$), both of length $N = \frac{n(n+1)}{2}$, in which case Equation 2.2 takes the form

$$\mathcal{P}(\boldsymbol{\sigma}) = \begin{bmatrix} \mathbf{p}_1\boldsymbol{\sigma'} \\ \vdots \\ \mathbf{p}_m\boldsymbol{\sigma'} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_m \end{bmatrix} \cdot \boldsymbol{\sigma'} \in \mathbb{F}_q^m, \qquad \text{(Eq. 2.5)}$$

$$\mathbf{p} = \begin{bmatrix} p_{1,1} & \dots & p_{1,n} & p_{2,2} & \dots & p_{2,n} & \dots & p_{n-1,n-1} & p_{n-1,n} & p_{n,n} \end{bmatrix} \in \mathbb{F}_q^N,$$

$$\mathbf{x}' = \begin{bmatrix} x_1^2 & \dots & x_1 \cdot x_n & x_2^2 & \dots & x_2 \cdot x_n & \dots & x_{n-1}^2 & x_{n-1} \cdot x_n & x_n^2 \end{bmatrix} \in \mathbb{F}_q^N.$$

2.6 CROSS

The Codes and Restricted Objects Signature Scheme (CROSS) is a code-based signature scheme from the NIST process, proposed by Baldi et al. [15]. It transforms an interactive zero-knowledge proof, called CROSS-ID, into a signature scheme using the Fiat-Shamir transform [9].

CROSS is based on the hardness of decoding restricted vectors. The signer must prove their knowledge of the restricted vector (referred to as the error vector) $\mathbf{e}$, that solves the public equation made up of a parity-check matrix $\mathbf{H}$ and a syndrome $\mathbf{s}$.

The vector is restricted as all its entries must belong to a small cyclic subgroup ($\mathbb{E}$). This makes the problem computationally difficult to solve for an attacker. There are two variants of CROSS, Restricted Syndrome Decoding Problem (R-SDP) and Restricted Syndrome Decoding Problem with subgroup G (R-SDP(G)), both of which have been proven to be NP-complete. RSD-P states that given a parity-check matrix $\mathbf{H}$ and a syndrome $\mathbf{s}$, find an error vector $\mathbf{e}$ where all entries belong to a fixed, restricted subset of the finite field, such that $\mathbf{s} = \mathbf{e}\mathbf{H}^\mathsf{T}$. For the R-SDP(G) variant, instead of any restricted vector, the solution must come from a subgroup G of these restricted vectors.

2.6.1 CROSS Verification

The Verify algorithm of CROSS, as introduced by Baldi et al. [15], is reproduced in Fig. B.1. This section aims to provide a focused explanation of the algorithm, emphasising the components most pertinent to the implementation of progressive verification.

To achieve strong security, CROSS executes the verification protocol in parallel over multiple rounds. In each round, the verifier will issue two challenges, referred to as $chall_1$ and $chall_2$. The first challenge $chall_1$ is a vector challenge that mixes the secret error vector $\mathbf{e}'$ with a random mask vector $\mathbf{u}'$ in order to form the masked vector $\mathbf{y}$, where each component is computed as $\mathbf{y}[i] = \mathbf{u}'[i] + chall_1[i]\mathbf{e}'[i]$.

The second challenge $chall_2$ is a bit 0 or 1 that selects which property the signer must reveal. In the case of $chall_2 = 1$, the signer is forced to prove that the vector $\mathbf{e}$

really lies within the cyclic subgroup through pseudorandom regeneration and hash checks. This is done by the signer revealing the random seed used to generate $(\mathbf{u}', \mathbf{e}')$ and the verifier recomputing these vectors to check that the commitments match.

If $chall_2 = 0$ the signer is forced to prove consistency with the syndrome equation. In this case, the signer discloses the masked vector $\mathbf{y}$ and a transformation vector $\mathbf{w}$. The transformation vector $\mathbf{w}$ is referred to as $\mathbf{v}$ in Fig. B.1. The verifier will check if the provided $\mathbf{y}$ matches its previous hash digest which guarantees consistency. After which the verifier will compute the $\mathbf{s}'[i] = \mathbf{y}'\mathbf{H}^\mathsf{T} - chall_1[i] \cdot \mathbf{s}$ where $\mathbf{y}' = \mathbf{w}[i] \star \mathbf{y}[i]$. This step is the computationally heavy part as it requires a full matrix vector multiplication. Finally, the verifier will check that hashing $(\mathbf{s}'[i], \mathbf{w}[i])$ reproduces the original commitment, and that $\mathbf{w}$ lies within $\mathbb{E}$. If and only if all tests succeed, the round is accepted.

Both cases of $chall_2$ ensure that a signer cannot satisfy both checks (i.e. the syndrome equation and the subgroup restrictions) without knowing a valid secret key. By repeating this process across multiple rounds, with a fixed number of $chall_2 = 0$ and $chall_2 = 1$ challenges the chances of forgery become negligible.

In order to apply PV there must be an $\mathbf{Mv}$-style multiplication within the signature's verification. This multiplication can be found in CROSS when $chall_2 = 0$. Therefore, this is the part of the verification that will be focused on in this thesis. In Fig. B.1 is the original Verify algorithm from CROSS [15].

2.7 ZKP by Lyubashevsky et al.

The paper "Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General" by Lyubashevsky et al. [5] proposes a scheme for lattice-based relations. The main goal is for the prover to prove knowledge of a short vector $\mathbf{s}$ that satisfies the linear relation $\mathbf{As} = \mathbf{t} \bmod q$, where $\mathbf{A}$ is a public matrix, $\mathbf{t}$ is a public vector and q is a modulus defining the ring $\mathbb{Z}_q$ over which the linear relations are computed. This is a fundamental problem for lattice-based cryptographic primitives.

This scheme utilises a novel "ABDLOP" commitment which combines the Ajtai [16] and BDLOP [17] commitment schemes. This allows high dimension secret messages to be committed without increasing the commitment size. This method maps vector inner-products to polynomial coefficients, achieving shorter proofs compared to earlier lattice-based ZKPs, while remaining flexible enough to support a wide range of cryptographic applications.

The ADLOP scheme is based on the hardness of the Module Short Integer Solution (MSIS) problem, which is a central challenge in lattice-based ZKPs and involves proving that the secret vector $\mathbf{s}$ is short (meaning that it is norm bounded), without revealing any information on $\mathbf{s}$ itself. For further details on the ABDLOP commitment, readers can refer to the original work by Lyubashevsky et al. [5], though this is not necessary for understanding of the thesis.

2.7.1 Proof of Knowledge for Committed Messages

Lyubashevsky et al. [5] specifically details the Proof of Knowledge for Committed Messages $(\mathbf{s_1}, \mathbf{m})$ and the verification that they satisfy a public linear relation $\mathbf{R}_1\mathbf{s}_1 + \mathbf{R}_m\mathbf{m} = \mathbf{u} \bmod q$. The original figure from the paper by Lyubashevsky et al. [5] can be found in Fig. C.1. To do this, the verifier performs three checks.

In the first check, the verifier ensures that the masked openings $(\mathbf{o}_1, \mathbf{o}_2)$ sent by the prover fall within the norm bounds. Specifically, it checks:

$$\|\mathbf{o}_1\| \leq \mathfrak{s}_1\sqrt{2m_1d}, \qquad \|\mathbf{o}_2\| \leq \mathfrak{s}_2\sqrt{2m_2d}$$

where d is the power of two that represents the degree of the underlying ring $\mathbb{Z}_q$ and $\mathfrak{s}_1$ and $\mathfrak{s}_2$ are standard deviations.

The second check involves verifying the internal consistency of the commitment using the public matrices $\mathbf{A}_1, \mathbf{A}_2$ and the challenge c. This check is presented in the paper as follows:

$$\mathbf{A}_1\mathbf{o}_1 + \mathbf{A}_2\mathbf{o}_2 - c\mathbf{t}_A = w$$

This equation verifies that the prover knows a valid opening for the commitment t_A. It links $\mathbf{o}_1, \mathbf{o}_2$ to the initial commitment and the initial masking value w sent by the prover.

The final check confirms that the committed value satisfies the required linear relation $\mathbf{R}_1\mathbf{s}_1 + \mathbf{R}_m\mathbf{m} = \mathbf{u} \bmod q$, where $\mathbf{R}_1$ and $\mathbf{R}_m$ are public matrices and $\mathbf{u}$ is the target vector. The verifier computes:

$$\mathbf{R}_1\mathbf{o}_1 + \mathbf{R}_m(c\mathbf{t}_B - \mathbf{B}\mathbf{o}_2) - c\mathbf{u} = \mathbf{x}$$

$\mathbf{B}$ is a public matrix that links secret randomness to the unrestricted message m. Thus, in this check, the term $c\mathbf{t}_b - \mathbf{B}\mathbf{o}_2$ represents the masked version of the message $\mathbf{m}$, which is defined as $\mathbf{t}_B - \mathbf{B}\mathbf{s}_2$. By validating the equation against the prover's initial value $\mathbf{x}$, the verifier confirms the linear relationship without the prover having to reveal the secret message.

The first check falls outside of the scope of the PV compiler as it has no **Mv**-style check. Therefore this thesis will focus on the second and third verification checks.

3

Results

This chapter is divided into three main sections, each depicting an application of progressive verification on a cryptographic primitive. Each section details how the four main subroutines (**parse** pk, Check, GetM and GetV) interact with the verification steps, and a security analysis for PV on the primitive.

3.1 UOV

As discussed in Section 2.5.1, UOV admits two distinct representations of its homogeneous quadratic polynomials: a matrix-based representation (Eq. 2.4) and a vector-based representation (Eq. 2.5). These representations each give rise to a different matrix–vector multiplication in the verification, leading to two distinct ways to apply the PV compiler.

The common feature of the two solutions is that both **parse** pk and Check are trivial, as pk consists only of the multivariate quadratic polynomials and no auxiliary checks are performed, i.e. Check always returns 1.

With the matrix-based representation (Eq. 2.4), the GetM algorithm extracts the set of m matrices $\{\mathbf{P}_i \in \mathbb{F}_q^{n\times n}\}_{i\in m}$ from pk. For each $\mathbf{P}_i$ it computes the row vector $\boldsymbol{\sigma}^\mathsf{T}\mathbf{P}_i$, stacks these to form $\mathbf{A}$, and extends it to $\mathbf{M}$ by appending the negative identity matrix:

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{\sigma}^\mathsf{T}\mathbf{P}_1 \\ \vdots \\ \boldsymbol{\sigma}^\mathsf{T}\mathbf{P}_m \end{bmatrix} \in \mathbb{F}_q^{m\times n}, \qquad \mathbf{M} = \begin{bmatrix} \mathbf{A} & -\mathbf{I}_m \end{bmatrix} \in \mathbb{F}_q^{m\times(n+m)}.$$

The GetV algorithm simply computes $\mathbf{h} = \mathsf{Hash}(\mu)$ and returns the vector

$$\mathbf{v} = \begin{bmatrix} \boldsymbol{\sigma} \\ \mathbf{h} \end{bmatrix} \in \mathbb{F}_q^{n+m}.$$

Now, while this approach achieves progressiveness, it does not promise efficiency for repeated verifications. The signature $\boldsymbol{\sigma}$ is used directly in GetM to construct $\mathbf{A}$, which means the secret verification key (svk), derived from $\mathbf{M}$, cannot be reused

across multiple verifications. As discussed in Section 2.4.2, reusing svk is critical for reducing the computational overhead of progressive verification.

To address the efficiency limitations of the first approach, we instead use the vector-based representation (Eq. 2.5). Here, the GetM algorithm extracts the set of m coefficient vectors $\{\mathbf{p}_i \in \mathbb{F}_q^N\}_{i \in m}$ from pk and constructs the matrices:

$$\mathbf{A} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_m \end{bmatrix} \in \mathbb{F}_q^{m \times N}, \qquad \mathbf{M} = \begin{bmatrix} \mathbf{A} & -\mathbf{I}_m \end{bmatrix} \in \mathbb{F}_q^{m \times (N+m)}.$$

The GetV algorithm, again, computes the hash of the message $\mathbf{h} = \mathsf{Hash}(\mu)$, but now also transforms the signature vector $\boldsymbol{\sigma}$ of length n into a vector $\boldsymbol{\sigma'}$ of length $N = \frac{n(n+1)}{2}$ by computing all products of pairs of elements in $\boldsymbol{\sigma}$. It then returns the vector

$$\mathbf{v} = \begin{bmatrix} \boldsymbol{\sigma'} \\ \mathbf{h} \end{bmatrix} \in \mathbb{F}_q^{N+m}.$$

Security Analysis

UOV inherits the generic security of the PV compiler. This analysis details the correctness and security properties of PV with UOV, ensuring that the properties of the original algorithm are preserved.

Correctness

For any valid signature in UOV, $\mathbf{Mv} = 0$ will hold exactly. Each PV step is an inner-product check $\mathbf{Z}'[i, *] \cdot \mathbf{v} = 0$, which will succeed for a valid signature with the probability of 1 regardless of the chosen random test vector.

Security

For an invalid signature where $\mathbf{Mv} \neq 0$, each PV step performs the same $\mathsf{ProgStep}$ check as above. However, each additional step reduces the probability that the invalid signature is accepted. If the signature is invalid, the probability that the inner-product check will be satisfied is at most $\frac{1}{q}$, where q is the field size of the finite field $\mathbb{F}_q$. Consequently, the probability that an invalid signature passes i independent PV steps is $\frac{1}{q^i}$. Thus the confidence value output α after i successful steps is $\alpha_{prog}(i) = 1 - \frac{1}{q^i}$.

This shows that the likelihood of an invalid signature passing multiple progressive steps decreases exponentially, providing adjustable soundness while preserving security guarantees from the original verification.

3.2 CROSS

In the verification of CROSS (Section 2.6), (in rounds where $chall_2 = 0$) the core computational step is the reconstruction of a syndrome $\mathbf{s}'$, provided by the signer. For this, the verifier checks $\mathbf{y}'\mathbf{H}^\mathsf{T} - chall_1\mathbf{s} \stackrel{?}{=} \mathbf{s}'$, where $\mathbf{H}$ and $\mathbf{s}$ are the public parity-check matrix and the public syndrome that constitute the public key. This can be expressed as the matrix–vector multiplication

$$\begin{bmatrix}\mathbf{H} & -\mathbf{s}^\mathsf{T}\end{bmatrix}\begin{bmatrix}\mathbf{y}' \\ chall_1\end{bmatrix} \stackrel{?}{=} \mathbf{s}',$$

theoretically fitting the $\mathbf{Mv}$-structure required by the compiler, as described in Section 2.4.2.

However, for the PV compiler to be applicable, the all-or-nothing verification step must be reducible to a linear check of the form $\mathbf{M}\cdot\mathbf{v} = 0$. In CROSS, this is not the case: the verifier does not directly compare the reconstructed value to $\mathbf{s}'$. Instead, the validity condition is checked against the commitment cmt_0, which is the digest of $\mathbf{s}'$. The verifier computes the hash of the reconstructed vector and checks if it matches the commitment provided in the signature:

$$\mathsf{Hash}(\mathbf{y}'\mathbf{H}^\mathsf{T} - chall_1\mathbf{s}) \stackrel{?}{=} cmt_0.$$

This approach leverages the collision resistance of the cryptographic hash function, ensuring that if the digests match, the input values are equal with high probability. However, the use of hashing obscures the underlying linearity of the inputs, preventing the restructuring required for $\mathbf{Mv}$-style progressive verification. This introduces a fundamental barrier to applying the PV compiler.

Hashes are often employed to enhance efficiency (reducing the size of the values to be verified or shortening signatures), but sometimes they provide essential properties required for security proofs. Consequently, removing or replacing the hashing mechanism, for instance with homomorphic hashing, cannot be undertaken lightly, as it may have significant security implications. A thorough analysis of such changes falls outside the scope of this thesis.

In summary, while CROSS incorporates a matrix-vector multiplication structure that aligns with the requirements of Mv-style progressive verification, its reliance on cryptographic hash functions for equality checks fundamentally prevents direct application of the PV compiler. The hash-based verification obscures the linearity required for progressive verification, rendering the scheme incompatible with the PV framework in its current form.

3.3 Proof of Knowledge for Committed Messages

As discussed in Section 2.7.1, the verification in zero-knowledge proof scheme proposed by Lyubashevsky et al. [5] consists of three checks, two of which rely heavily on matrix-vector multiplications.

Check One: The first check involves verifying that the response vectors are within the norm bounds. This does not include an **Mv**-style check and is therefore not relevant to the application of PV.

Check Two:

$$\mathbf{A}_1\mathbf{o}_1 + \mathbf{A}_2\mathbf{o}_2 - c\mathbf{t}_A = \mathbf{w}$$

Which to match $\mathbf{Mv} = 0$ form can be rewritten as:

$$\mathbf{A}_1\mathbf{o}_1 + \mathbf{A}_2\mathbf{o}_2 - (\mathbf{w} + c\mathbf{t}_A) = 0$$

Check Three:

$$\mathbf{R}_1\mathbf{o}_1 + \mathbf{R}_m(c\mathbf{t}_b - \mathbf{B}\mathbf{o}_2) - c\mathbf{u} = \mathbf{x}$$

Which to match $\mathbf{Mv} = 0$ form can be rewritten as:

$$\mathbf{R}_1\mathbf{o}_1 - \mathbf{R}_m\mathbf{B}\mathbf{o}_2 - (\mathbf{x} + c\mathbf{u} - \mathbf{R}_m c\mathbf{t}_B) = 0$$

parse pk is trivial as pk consists solely of the public matrices and vectors. The norm bound check is handled by the Check subroutine, accepting only if the response vectors lie within the norm bounds.

The algorithm GetM constructs the matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{A_1} & \mathbf{A_2} & -(\mathbf{w} + c\mathbf{t_A}) \\ \mathbf{R_1} & -\mathbf{R_m}\mathbf{B} & -(\mathbf{x} + c\mathbf{u} - \mathbf{R_m}c\mathbf{t_B} \end{bmatrix}$$

The GetV algorithm extracts the "prover-controlled" values and returns the vector:

$$\mathbf{v} = \begin{bmatrix} \mathbf{o_1} \\ \mathbf{o_2} \\ 1 \end{bmatrix}$$

Security Analysis

Applying progressive verification to the Proof of Knowledge for Committed Messages by Lyubashevsky et al. [18] will preserve the correctness and security properties of the original verification algorithm as described below. These security analyses apply only in the second and third verification checks. In first verification check, progressive verification is not invoked and the verifier instead follows the original verification procedure. Consequently, the security in the first check remains identical to that of the original paper.

Correctness:

For any valid committed message, $\mathbf{A}_1\mathbf{o}_1 + \mathbf{A}_2\mathbf{o}_2 - (\mathbf{w} + c\mathbf{t}_A) = 0$ and $\mathbf{R}_1\mathbf{o}_1 - \mathbf{R}_m\mathbf{B}\mathbf{o}_2 - (\mathbf{x} + c\mathbf{u} - \mathbf{R}_m c\mathbf{t}_B) = 0$ will hold exactly. Both of these checks can be rewritten as $\mathbf{Mv} = 0$ as described above. Each PV step is an inner-product check ($\mathbf{Z}'[i, *] \cdot \mathbf{v} = 0$), which will succeed for a valid committed message with the probability of 1 regardless of the chosen random test vector.

Security:

For an invalid proof, at least one of the linear equations will not hold. The probability that an inner-product check will succeed is at most $\frac{1}{q}$ where q is the modulus defining the underlying ring $\mathbb{Z}_q$ over which the scheme is instantiated.

The probability that an invalid proof passes i independent steps is bounded by $\frac{1}{q^i}$. Equivalently, after i successful steps the confidence level in the output is $\alpha_{prog}(i) = 1-\frac{1}{q^i}$. This entails that the likelihood of an invalid proof passing multiple progressive steps decreases exponentially.

In summary, applying progressive verification to the Proof of Knowledge for Committed Messages will preserve the correctness and security properties of the original verification algorithm.

4

Discussion

The purpose of this thesis was to explore whether the PV compiler by Boschini et al. [1] could be applied to post-quantum digital signature schemes beyond the ones that had previously been investigated, as well as to zero-knowledge proofs. The focus was on MQ- and code-based schemes from the NIST process, as well as a zero-knowledge proof. The results of this research shows that progressive verification has wider applicability than previously known, and can be applied across fundamentally different cryptographic families, i.e multivariate quadratic and zero-knowledge proof schemes. This significantly strengthens the relevance of progressive verification as a general technique rather than a scheme specific optimisation. Furthermore, findings showed that a direct application of the PV compiler was not possible for any code-based schemes from the NIST process.

The findings show that the main obstacle of deploying post-quantum signature schemes to resource constrained devices is mainly the size of the public key or μ, which impacts the verifications running time. This can be mitigated using PV, as long as the scheme's verification structure allows for it. A general observation is that the applicability of PV depends entirely on whether or not verification procedure of a scheme contains an $\mathbf{Mv}$-style check of the form $\mathbf{Mv} = 0$. The progressive verification compiler relies on decomposing this monolithic check into randomised inner-product checks, as illustrated in Fig. 2.2 in Section 2.4.2. As $\mathbf{Mv}$-style verification appears naturally in many PQC schemes, PV is able to exploit this structure to transform an "all or nothing" computation into a gradual process without modifying the underlying signing or key-generation algorithms.

4.1 Evaluation of Research Questions

This section revisits the research questions introduced in Section 1.3 and addresses them in light of the results presented in Chapter 3. Each research question is considered individually, drawing on the finding of the thesis to evaluate the extent to which it has been answered, as well as to discuss the implications and limitations revealed by the results.

RQ1a: Can the compiler be applied to multivariate quadratic digital signature schemes, other than the ones treated by [1]?

For MQ schemes such as UOV, the verification equation can be expressed as a

matrix-vector product, where verifying the signature requires checking that a set of public polynomials evaluated at the signature match the hashed message. This structure corresponds directly to the **Mv**-style form required by the compiler. In both initial and improved constructions, the UOV verification step was successfully decomposed into a progressive sequence of inner-product checks.

However, the first straightforward construction (presented in Section 3.1) revealed the practical challenge of efficiency. In UOV, computing the matrix $\mathbf{A}$ requires incorporating the message/ signature pair into the matrix construction, which prevents the reuse of the precomputed randomness $\mathbf{Z}$. As the goal was not only to offer progressiveness but also reduce average computation time, this limitation was significant. The revised solution resolved this by representing multivariate quadratic polynomials in a way that decouples the public-key dependent preprocessing with message-specific operations. This allowed $\mathbf{Z}$ to be reused across multiple verifications, recovering the efficiency benefits from Boschini et al. [1] and in the efficiency guidelines of Section 2.4.2.

These findings illustrate that while MQ schemes are structurally compatible with PV, careful consideration is required if efficiency is to be preserved.

RQ1b: Can the compiler be applied to code-based digital signature schemes?
This research highlighted the inapplicability of the PV compiler on the code-based scheme CROSS. While initial analysis identified a matrix-vector multiplication in CROSS's verification (suggesting PV may be applicable), closer inspection revealed that the verification structure prevented direct transformation. CROSS uses a final cryptographic hash comparison which hides the underlying structure of the inputs. If the hash were to be before the **Mv**-style check (as is the case in UOV) it would provide constant inputs for a linear equation, which would preserve the algebraic structure required for PV. However, hashing after the **Mv**-style check encapsulates the linear operations inside a non-linear black box, forcing the verifier to compute the full result monolithically to validate the hash.

Because the PV compiler relies on restructuring algebraic equations into randomised inner-product checks, the nature of the hash function breaks the linearity required for the transformation. Furthermore, any modification or partial check of the vector $\mathbf{s}'$ would produce a cascade of differences in the resulting hash, making it impossible to verify the signature incrementally without completing the full matrix-vector multiplication.

This result highlights that the feasibility of progressive verification does not solely rely on the presence of a matrix-vector multiplication in the scheme's verification, but also the requirement that the verification steps are purely algebraic and free from structural alterations. Consequently, the all-or-nothing nature of the hash-digest comparison in CROSS renders the PV compiler inapplicable without a fundamental redesign of the scheme, which would carry significant and unverified security implications.

RQ2: Can the progressive verification compiler from [1] be applied to zero-knowledge proof schemes?

A notable result of this thesis is the demonstration that zero-knowledge proofs, specifically the scheme proposed by Lyubyashevsky et al. [5], also support progressive verification. This is novel and non-trivial as previous work on the PV compiler had primarily focused on digital signatures. Showing that the compiler can be extended to ZKPs supports the hypothesis that PV can work as a generic cryptographic technique rather than a scheme specific optimisation.

The results show that the second and third verification checks contain an **Mv**-style formulation making them compatible with the PV compiler. However, the first verification check (which ensures that the prover's response vectors lie within the norm bounds) can not be expressed as a linear equality check. Therefore it falls outside of the scope for the PV compiler and must remain monolithic. This limits the extent to which the verification process for the Proof of Knowledge for Committed Messages can be made progressive.

This partial applicability suggests that progressive verification in zero-knowledge proofs would mainly benefit in scenarios where the **Mv**-style checks dominate the computational costs. In lattice-based constructions, matrix-vector multiplications are often the most expensive operations, whereas norm checks are comparatively cheap as they are quick, lightweight and do not require a large public key. Therefore, even though full progressiveness is not achievable, applying the PV compiler to the **Mv**-style checks would still yield efficiency gains and early rejection of invalid inputs.

4.2 PV for IOT Devices

On IOT devices, verification may be interrupted due to resource constraints or power loss. With progressive verification a confidence level in the validity of the signature is still obtained at such an interruption, which aids PQC verification on constrained platforms. Furthermore, since each progressive step independently detects inconsistencies; incorrect or adversarial inputs are likely to be rejected after only a few steps. This reduces wasted computation and energy, which is particularly valuable for resource- or battery constrained devices. Lastly PV allows for a trade-off between soundness and efficiency, which is especially relevant in devices that have varying energy budgets and latency requirements.

All of this shows that there is realistic hope for PQC schemes on resource constrained devices, but some aspects must be accounted for. The restriction is no longer due to heavy computations, but signature sizes and public keys remain large in comparison to classical schemes. However, progressive verification offers a way to make these costs manageable and adaptable rather than rigid and worst-case driven.

Multiple MQ schemes can fully benefit from PV, making them especially attractive candidates for constrained environments where early rejection and reuse of precomputation are feasible. Furthermore, the ZKP studied, also contains a verification structure that can be exploited, demonstrating that progressive verification is not restricted to a single cryptographic primitive. As a whole, progressive verification can be integrated with existing schemes without modifying the underlying security assumptions or protocol logic, preserving existing scheme designs.

These results indicate that progressive verification can act as an enabling layer between post-quantum cryptography and constrained hardware. PV allows well-studied, existing schemes to be adapted to such constrained settings, thus IoT devices can perform "best effort" verification, stopping early when resources are scarce while still obtaining meaningful information.

4.3 Ethical Considerations

A significant aspect of progressive verification is the inherent trade-off between security and computational efficiency. While this could be beneficial for resource-constrained devices, it can also lead to some risks as it allows for lower security levels which can be exploited if not properly managed. This must be transparently communicated to potential users in order for them to make informed decisions. The user must not be led to believe that progressive verification is a magical solution that offers both perfect security and efficiency.

Moreover, introducing validity as a non-binary output opens up an ethical dilemma which requires careful consideration. Misinterpreting or overestimating the safety of the given output value could lead to incorrect decisions and potentially severe consequences. Ethically, developers must hold the responsibility and justify the security threshold, ensuring that the chosen compromise between efficiency and security is appropriate for the systems intended use.

4.4 Future Work

This chapter outlines several potential directions for future work within progressive verification. It discusses how the results of this thesis could be extended or applied in new contexts and highlights promising avenues for further research.

4.4.1 Practical Implementations

Building on the theoretical results of this research, a natural progression for future works involves a transition from mathematical proofs to practical implementations and benchmarking. While this thesis has established the compatibility of PV on UOV and the Proof of Knowledge for Committed Messages scheme, the actual performance gain on physical hardware remain to be quantified. Thus, future work could focus on implementing these on IOT devices which would allow for precise measurements.

For the most interesting results, benchmarking should target specific scenarios where early rejection and adjustable soundness provide most benefit. By measuring the verification speed-up when processing invalid inputs, future work could validate the efficiency claims of the PV compiler in real-world scenarios.

4.4.2 Remaining NIST candidates

Fourteen additional post-quantum digital signature schemes remain under consideration for NIST standardisation beyond those examined in this work and in the work done by Boschini et al. [1]. These additional schemes are presented in Table 4.1. Further investigation of these schemes could be an interesting angle for future work, displaying the wider versatility of progressive verification for further PQC digital signatures. We have begun a preliminary analysis of the remaining signature schemes and presented the findings in this chapter.

Multivariate Schemes

Two of the remaining candidates in the NIST process are multivariate quadratic schemes, specifically Simple Noncommutative unbalanced Oil and Vinegar scheme with randomness Alignment (SNOVA) and Quotient Ring - Unbalanced Oil and Vinegar (QR-UOV).

There is good reason to expect that progressive verification could also be applied to the remaining multivariate schemes. The compiler introduced in [1] is explicitly designed to apply to multivariate polynomial based schemes, a category that includes both SNOVA and QR-UOV. Moreover, verification in both schemes requires the evaluation of MQ polynomials, which can be structured as or reduced to matrix-vector multiplication. Finally, SNOVA and QR-UOV are themselves variants of UOV, and since progressive verification has been already been demonstrated for LUOV (a UOV variant), Rainbow (a multilayer UOV variant) and UOV itself, it is reasonable to infer that the same approach should extend to other UOV-derived constructions.

Code-based Schemes

One additional code-based scheme also remains in contention for the NIST standardisation process, this is the Linear Equivalence Signature Scheme (LESS). Preliminary research shows that the progressive verification compiler is not directly applicable to LESS for many of the same reasons as to why it was inapplicable to CROSS. The inapplicability of the PV compiler to LESS stems from structural limitations and its reliance on a final hash check, which mirrors the issue discussed in Section 4.1.

The compiler requires that a signature scheme's verification procedure be **Mv**-style, i.e dominated by a matrix-vector check of the form $\mathbf{Mv} = 0$, which the compiler then compresses into probabilistic sub-checks. LESS, however, does not possess such an **Mv**-style verification routine. Its verification arises from a Fiat-Shamir Sigma protocol and involves reduced row-echelon computations, monomial map operations and canonical form tests. These steps are non-linear and are not able to be collapsed into an algebraic relation which would fit the required form for PV. Therefore the heterogeneous verification of LESS does not seem to fall within the scope for the compiler introduced by [1]. Consequently, applying progressive verification to LESS would require a fundamental redesign of the scheme, which would defeat the purpose of using PV solely as an extension.

Scheme Type	**Scheme**	**PV Friendly**?
Code	CROSS	×
	LESS	×
Isogony	SQIsign	?
Lattice	Hawk	?
	CRYSTALS-DILITHIUM	?
	Falcon	?
MPC	Mirath	?
	MQOM	?
	PERK	?
	RYDE	?
	SDitH	?
Multivariate	MAYO	✓
	QR-UOV	✓
	SNOVA	✓
	UOV	✓
Symmetric	FAEST	?
Hash	**SPHINCS**$^+$	?

Table 4.1: The applicability of progressive verification on schemes in the NIST standardisation process, where ✓ denotes proven applicability, ✓denotes presumed applicability, × denotes proven inapplicability and × denotes presumed inapplicability. ? are schemes that have not yet been examined.

5

Conclusion

The aim of this thesis was enhance the flexibility and practical utility of post-quantum cryptographic primitives by investigating the applicability of progressive verification on said primitives. The research focused on digital signatures from the NIST standardisation process [4] and on a zero-knowledge proof proposed by Lyubashevsky et al. [5].

Building on the PV compiler by Boschini et al. [1], this work examined whether existing verification procedures could be transformed into progressive ones without modifying the key generation or signing algorithms. The central requirement for such a transformation is that the verification can be expressed as a matrix-vector equation of the form $\mathbf{Mv} = 0$.

The findings showed that the multivariate quadratic scheme UOV fully supports progressive verification. Two alternative formulations were presented, of which the latter enables reuse of verifier pre-computation thus preserving the efficiency benefits of PV. This confirms that UOV is structurally well suited for progressive verification while maintaining the scheme's original security guarantees.

In contrast, the code-based signature scheme CROSS was shown to be incompatible with the PV compiler. Although a matrix-vector multiplication appears in parts of the verification, the correctness of the computation is ultimately validated through hash-based equality checks. This obscures the linear structure required for progressive verification and prevents incremental validation as discussed in Section 4.4.2.

Beyond digital signature, this thesis demonstrated that the progressive verification compiler can also be applied to zero-knowledge proofs. For the Proof of Knowledge for Committed Messages, two of the three verification checks use an $\mathbf{Mv}$-style formulation and can be made progressive. While full progressiveness is not achievable due to the norm bound check, the most computationally expensive components can still benefit from PV. This result extends the applicability of progressive verification beyond digital signatures and shows that it can function as a general cryptographic technique.

To summarise, this thesis expands the set of schemes known to support the progressive verification technique. The findings indicate that PV can act as an enabling layer between post-quantum cryptography and resource constrained devices. It presents a possibility for existing schemes to be deployed in environments where monolithic PQC verification would be infeasible. While not universally applicable, progressive

verification offers an approach to managing the cost of post-quantum verification in real-world systems.

Bibliography

[1] C. Boschini, D. Fiore, E. Pagnin, L. Torresetti, and A. Visconti, "Progressive and efficient verification for digital signatures: Extensions and experimental results," *J Cryptogr Eng*, vol. 14, no. 3, pp. 551–575, Sep. 2024, ISSN: 2190-8508, 2190-8516. DOI: 10.1007/s13389-024-00358-0. Accessed: Sep. 30, 2025. [Online]. Available: https://link.springer.com/10.1007/s13389-024-00358-0.

[2] "What Is Post-Quantum Cryptography?" en, *NIST*, Aug. 2024, Last Modified: 2025-06-11T10:31-04:00. Accessed: Nov. 25, 2025. [Online]. Available: https://www.nist.gov/cybersecurity/what-post-quantum-cryptography.

[3] NIST CSRC. "Post-quantum cryptography," NIST CSRC, Accessed: Sep. 30, 2025. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/.

[4] NIST CSRC. "Additional digital signature schemes," NIST CSRC, Accessed: Sep. 30, 2025. [Online]. Available: https://csrc.nist.gov/projects/pqc-dig-sig.

[5] V. Lyubashevsky, N. K. Nguyen, and M. Plancon, *Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general*, Published: Cryptology ePrint Archive, Paper 2022/284, 2022. [Online]. Available: https://eprint.iacr.org/2022/284.

[6] W. Diffie and M. E. Hellman, "New Directions in Cryptography," en, in *Democratizing Cryptography*, R. Slayton, Ed., 1st ed., New York, NY, USA: ACM, Aug. 2022, pp. 365–390, ISBN: 978-1-4503-9827-5. DOI: 10.1145/3549993.3550007. Accessed: Nov. 11, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3549993.3550007.

[7] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," en, *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/359340.359342. Accessed: Nov. 11, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/359340.359342.

[8] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, 2019, pp. 203–225.

[9] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," en, in *Advances in Cryptology CRYPTO 86*, A. M. Odlyzko, Ed., Berlin, Heidelberg: Springer, 1987, pp. 186–194, ISBN: 978-3-540-47721-1. DOI: 10.1007/3-540-47721-7_12.

[10] M. Fischlin, "Progressive verification: The case of message authentication," in *Progress in Cryptology - INDOCRYPT 2003*, T. Johansson and S. Maitra, Eds., Berlin, Heidelberg: Springer, 2003, pp. 416–429, ISBN: 978-3-540-24582-7. DOI: 10.1007/978-3-540-24582-7_31.

[11] D. V. Le, M. Kelkar, and A. Kate, "Flexible signatures: Making authentication suitable for real-time environments," in *Computer Security ESORICS 2019*, K. Sako, S. Schneider, and P. Y. A. Ryan, Eds., Cham: Springer International Publishing, 2019, pp. 173–193, ISBN: 978-3-030-29959-0. DOI: 10.1007/978-3-030-29959-0_9.

[12] A. R. Taleb and D. Vergnaud, "Speeding-up verification of digital signatures," *Journal of Computer and System Sciences*, vol. 116, pp. 22–39, Mar. 1, 2021, ISSN: 0022-0000. DOI: 10.1016/j.jcss.2020.08.005. Accessed: Sep. 29, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000020300854.

[13] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced oil and vinegar signature schemes," in *Advances in Cryptology EUROCRYPT 99*, J. Stern, Ed., Berlin, Heidelberg: Springer, 1999, pp. 206–222, ISBN: 978-3-540-48910-8. DOI: 10.1007/3-540-48910-X_15.

[14] W. Beullens et al., *UOV: Unbalanced oil and vinegar*, Feb. 5, 2025.

[15] M. Baldi et al., *CROSS: Codes and restricted objects signature scheme*, Jul. 31, 2025.

[16] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," en, in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, Philadelphia, Pennsylvania, United States: ACM Press, 1996, pp. 99–108, ISBN: 978-0-89791-785-8. DOI: 10.1145/237814.237838. Accessed: Jan. 3, 2026. [Online]. Available: http://portal.acm.org/citation.cfm?doid=237814.237838.

[17] C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, and C. Peikert, "More Efficient Commitments from Structured Lattice Assumptions," en, in *Security and Cryptography for Networks*, D. Catalano and R. De Prisco, Eds., Cham: Springer International Publishing, 2018, pp. 368–385, ISBN: 978-3-319-98113-0. DOI: 10.1007/978-3-319-98113-0_20.

[18] V. Lyubashevsky, N. K. Nguyen, and G. Seiler, *Shorter lattice-based zero-knowledge proofs via one-time commitments*, Published: Cryptology ePrint Archive, Paper 2020/1448, 2020. [Online]. Available: https://eprint.iacr.org/2020/1448.

A

PV Visualisation Example

In order to illustrate how PV would be applied to digital signature, a small example has been constructed using a 4×4 matrix. In real digital signature schemes these matrices are much larger (e.g in UOV where a matrix may be of size 244×244), however the structure is the same. The goal of the example is to show how the PV compiler transforms the digital signatures verification equations into a sequence of inner-product checks.

Recall Section 2.4.2, given a signature whose verification includes a matrix-vector multiplication $\mathbf{A} \cdot \boldsymbol{\sigma} = \mathbf{u}$, the equation can be restructured as $\mathbf{A} \cdot \boldsymbol{\sigma} - \mathbf{u} = \mathbf{0}$, which in block matrix form is

$$\begin{bmatrix}\mathbf{A} & -\mathbf{I}\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{\sigma} \\ \mathbf{u}\end{bmatrix} = \mathbf{0}.$$

The first step is to construct the matrix $\mathbf{A}$. For this example, $\mathbf{A}$ has been populated with small integers for clarity.

$$\mathbf{A} = \begin{bmatrix}1 & 0 & 2 & 1 \\ 2 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 2\end{bmatrix}$$

A 4×4 matrix has a negative identity matrix **-I** which is given by

$$-\mathbf{I} = \begin{bmatrix}-1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1\end{bmatrix}$$

As explained in Section 2.4.2, these matrices are concatenated horizontally to form the matrix $\mathbf{M}$, i.e $\mathbf{M} = \begin{bmatrix}\mathbf{A} & -\mathbf{I}\end{bmatrix}$. In this example, this yields an $\mathbf{M}$ of:

$$\mathbf{M} = \begin{bmatrix}1 & 0 & 2 & 1 & -1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 1 & 1 & 1 & 2 & 0 & 0 & 0 & -1\end{bmatrix}$$

At this point, the original verification condition has been reduced to checking whether $\mathbf{M} \cdot \mathbf{v} = 0$ holds for an appropriate vector $\mathbf{v}$. Therefore the next step is to construct $\mathbf{v}$, which is defined as $\mathbf{u} = \begin{bmatrix} \sigma \\ \mathbf{u} \end{bmatrix}$.

Assume the signature vector σ is

$$\sigma = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Let the vector $\mathbf{u}$ be

$$\mathbf{u} = \begin{bmatrix} 2 \\ 5 \\ 0 \\ 0 \end{bmatrix}$$

The full verification vector $\mathbf{v}$ is obtained by stacking σ and $\mathbf{u}$, which gives

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 2 \\ 5 \\ 0 \\ 0 \end{bmatrix}$$

Rather than evaluating the full matrix-vector multiplication, PV compresses this check into a sequence of inner-products. To do so, the verifier samples a random matrix $\mathbf{c}$ whose rows define independent PV checks. In this example, a $\mathbf{c}$ consisting of two rows has been chosen for simplicity of the visualisation, meaning two inner-product checks will be performed.

$$\mathbf{c} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

The compressed matrix $\mathbf{z}$ is then computed as

$$\mathbf{z} = \mathbf{c} \cdot \mathbf{M}$$

Each row of $\mathbf{z}$ is a combination of rows of $\mathbf{M}$, as determined by the corresponding row of $\mathbf{c}$.

Row 1 of $\mathbf{z}$ is obtained by adding rows 1 and 3 of $\mathbf{M}$:
$\mathbf{z}[1, *] = [1, 0, 1, 0] \cdot \mathbf{M}$
$\mathbf{z}[1, *] = [1, 1, 3, 1, -1, 0, -1, 0]$

Row 2 of $\mathbf{z}$ is obtained by adding rows 2 and 4 of $\mathbf{M}$:
$\mathbf{z}[2,*] = [0,1,0,1] \cdot \mathbf{M}$
$\mathbf{z}[2,*] = [3,2,1,3,0,-1,0,-1]$

Thus, $\mathbf{z}$ is the 2×8 matrix

$$\mathbf{z} = \begin{bmatrix} 1 & 1 & 3 & 1 & -1 & 0 & -1 & 0 \\ 3 & 2 & 1 & 3 & 0 & -1 & 0 & -1 \end{bmatrix}$$

These are the rows for two PV inner-product checks. Each row of $\mathbf{z}$ defines a PV inner-product check of the form $\mathbf{z}[i,*] \cdot \mathbf{v} = 0$. If the input signature is valid, all such checks should evaluate to zero. Conversely, if the verification equation does not hold, at least one of these checks will be non-zero with high probability, causing PV to reject early.

Using the chosen vector $\mathbf{v}$ and computing the inner product yields

$$\mathbf{z}[1,*] \cdot \mathbf{v} \stackrel{?}{=} 0 \text{ and } \mathbf{z}[2,*] \cdot \mathbf{v} \stackrel{?}{=} 0$$

For demonstration purposes, the vector $\mathbf{v}$ has been selected in a way that it satisfies $\mathbf{M}\mathbf{v} = 0$. Following is a visualisation that shows how both inner-products evaluate to zero.

Compute the products:
$\mathbf{z}[1,*] \cdot \mathbf{v} = 1 \cdot 1 + 1 \cdot 1 + 3 \cdot 0 + 1 \cdot 0 + (-1) \cdot 2 + 0 \cdot 5 + (-1) \cdot 0 + 0 \cdot 0$
$\mathbf{z}[1,*] \cdot \mathbf{v} = 1 + 1 + 0 + 0 - 2 + 0 + 0 + 0$
$\mathbf{z}[1,*] \cdot \mathbf{v} = 0$

$\mathbf{z}[2,*] \cdot \mathbf{v} = 3 \cdot 1 + 2 \cdot 1 + 1 \cdot 0 + 3 \cdot 0 + 0 \cdot 2 + (-1) \cdot 5 + 0 \cdot 0 + (-1) \cdot 0$
$\mathbf{z}[2,*] \cdot \mathbf{v} = 3 + 2 + 0 + 0 + 0 - 5 + 0 + 0$
$\mathbf{z}[2,*] \cdot \mathbf{v} = 0$

Both checks give zero which means that the PV steps pass for this $\mathbf{v}$.

B

CROSS verify algorithm

This appendix shows the CROSS verify algorithm exactly as it is presented in the original paper by Baldi et al. [15]. The R-SDP and R-SDP(G) variants are distinguished by colour, where the R-SDP(G) specific steps are in orange and the R-SDP specific steps are in teal.

It should be noted that the vector referred to as $\mathbf{v}$ in this verification algorithm is denoted as $\mathbf{w}$ in the rest of the thesis. This renaming is done to ensure uniform notation throughout the report.

Algorithm 3: Verify(pk, Msg, Sgn)

Input: pk: $(\mathtt{Seed}_{\mathtt{pk}}, \mathbf{s})$ public key;
$\mathtt{Msg} \in \{0,1\}^*$: message;
Sgn: $(\mathtt{Salt}, \mathtt{digest}_{\mathtt{cmt}}, \mathtt{digest}_{\mathtt{chall}_2}, \mathtt{Path}, \mathtt{Proof}, \mathtt{resp})$ signature;
Output: {True, False};
Data: λ: security parameter;
g: generator of $\mathbb{E}$;
t: number of rounds; w: weight of second challenge;
c: constant defined as $2t-1$;

// Recovering public key

$(\overline{\mathbf{W}}, \mathbf{V}) \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^{m\times(n-m)}\times\mathbb{F}_p^{(n-k)\times k}}(\mathtt{Seed}_{\mathtt{pk}} \mid 3t+2)$ | $\mathbf{V} \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_p^{(n-k)\times k}}(\mathtt{Seed}_{\mathtt{pk}} \mid 3t+2)$

$\mathbf{H} \leftarrow [\mathbf{V} \mid \mathrm{Id}_{n-k}]$

$\overline{\mathbf{M}} \leftarrow [\overline{\mathbf{W}} \mid \mathrm{Id}_m]$

// Computing challenges

$\mathtt{digest}_{\mathtt{Msg}} \leftarrow \mathsf{Hash}(\mathtt{Msg})$

$\mathtt{digest}_{\mathtt{chall}_1} \leftarrow \mathsf{Hash}(\mathtt{digest}_{\mathtt{Msg}} \mid \mathtt{digest}_{\mathtt{cmt}} \mid \mathtt{Salt})$

$\mathtt{chall}_1 \leftarrow \mathsf{CSPRNG}_{-(\mathbb{F}_p^*)^t}(\mathtt{digest}_{\mathtt{chall}_1} \mid t+c)$

$\mathtt{chall}_2 \leftarrow \mathsf{CSPRNG}_{-\mathcal{B}_{(t,w)}}(\mathtt{digest}_{\mathtt{chall}_2} \mid t+c+1)$

// Computing commitments

$(\mathsf{Seed}[i])_{i:\mathtt{chall}_2[i]=1} \leftarrow \mathsf{RebuildLeaves}(\mathtt{Path} \mid \mathtt{chall}_2 \mid \mathtt{Salt})$

for i **from** 1 **to** t **do**

  if $\mathtt{chall}_2[i] = 1$ **then**

    $\mathtt{cmt}_1[i] \leftarrow \mathsf{Hash}(\mathsf{Seed}[i] \mid \mathtt{Salt} \mid i+c)$

    $\overline{\mathbf{e}}'_G[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^m\times\mathbb{F}_p^n}(\mathsf{Seed}[i] \mid \mathtt{Salt} \mid i+c)$ | $\overline{\mathbf{e}}'[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^n\times\mathbb{F}_p^n}(\mathsf{Seed}[i] \mid \mathtt{Salt} \mid i+c)$

    $\overline{\mathbf{e}}'[i] \leftarrow \overline{\mathbf{e}}'_G[i]\overline{\mathbf{M}}$

    for j **from** 1 **to** n **do**

      $\mathbf{e}'[i]_j \leftarrow g^{\overline{\mathbf{e}}[i]_j}$

    $\mathbf{y}[i] \leftarrow \mathbf{u}'[i] + \mathtt{chall}_1[i]\mathbf{e}'[i]$

  if $\mathtt{chall}_2[i] = 0$ **then**

    $\mathtt{cmt}_1[i] \leftarrow \mathtt{resp}[i]_1$

    $(\mathbf{y}[i], \overline{\mathbf{v}}_G[i]) \leftarrow \mathtt{resp}[i]_0$ | $(\mathbf{y}[i], \overline{\mathbf{v}}[i]) \leftarrow \mathtt{resp}[i]_0$

    Check if $\overline{\mathbf{v}}_G[i] \in \mathbb{F}_z^m$ | Check if $\overline{\mathbf{v}}[i] \in \mathbb{F}_z^n$

    $\overline{\mathbf{v}}[i] \leftarrow \overline{\mathbf{v}}_G[i]\overline{\mathbf{M}}$

    for j **from** 1 **to** n **do**

      $\mathbf{v}[i]_j \leftarrow g^{\overline{\mathbf{v}}[i]_j}$

    $\mathbf{y}'[i] \leftarrow \mathbf{v}[\mathbf{i}] \star \mathbf{y}[i]$

    $\mathbf{s}'[i] \leftarrow \mathbf{y}'[i]\mathbf{H}^\top - \mathtt{chall}_1[i]\mathbf{s}$

    $\mathtt{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \overline{\mathbf{v}}_G[i] \mid \mathtt{Salt} \mid i+c)$ | $\mathtt{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \overline{\mathbf{v}}[i] \mid \mathtt{Salt} \mid i+c)$

// Checking digests

$\mathtt{digest}_{\mathtt{cmt}_0} \leftarrow \mathsf{RecomputeRoot}(\mathtt{cmt}_0 \mid \mathsf{Proof} \mid \mathtt{chall}_2)$

$\mathtt{digest}_{\mathtt{cmt}_1} \leftarrow \mathsf{Hash}(\mathtt{cmt}_1[1] \mid \cdots \mid \mathtt{cmt}_1[t])$

$\mathtt{digest}'_{\mathtt{cmt}} \leftarrow \mathsf{Hash}(\mathtt{digest}_{\mathtt{cmt}_0} \mid \mathtt{digest}_{\mathtt{cmt}_1})$

$\mathtt{digest}'_{\mathtt{chall}_2} \leftarrow \mathsf{Hash}(\mathbf{y}[1] \mid \cdots \mid \mathbf{y}[t] \mid \mathtt{digest}_{\mathtt{chall}_1})$

if $\mathtt{digest}_{\mathtt{cmt}} = \mathtt{digest}'_{\mathtt{cmt}}$ **and** $\mathtt{digest}_{\mathtt{chall}_2} = \mathtt{digest}'_{\mathtt{chall}_2}$ **then**

  return True

return False

Figure B.1: CROSS Verify algorithm, presented in the exact same way as Figure 3 in the original paper by Baldi et al. [15]

C

Proof of Knowledge for Committed Messages

This appendix shows the zero-knowledge proof of knowledge for committed messages by Lyubashevsky et al[5]. The figure depicts both the local computations performed by each party and the information exchanged between them.

Note that the vector referred to as $\mathbf{v}$ in this figure is denoted as $\mathbf{x}$, and $\mathbf{z}_1, \mathbf{z}_2$ are denoted as $\mathbf{o}_1, \mathbf{o}_2$ in the rest of the thesis. This renaming is done to ensure uniform notation throughout the report.

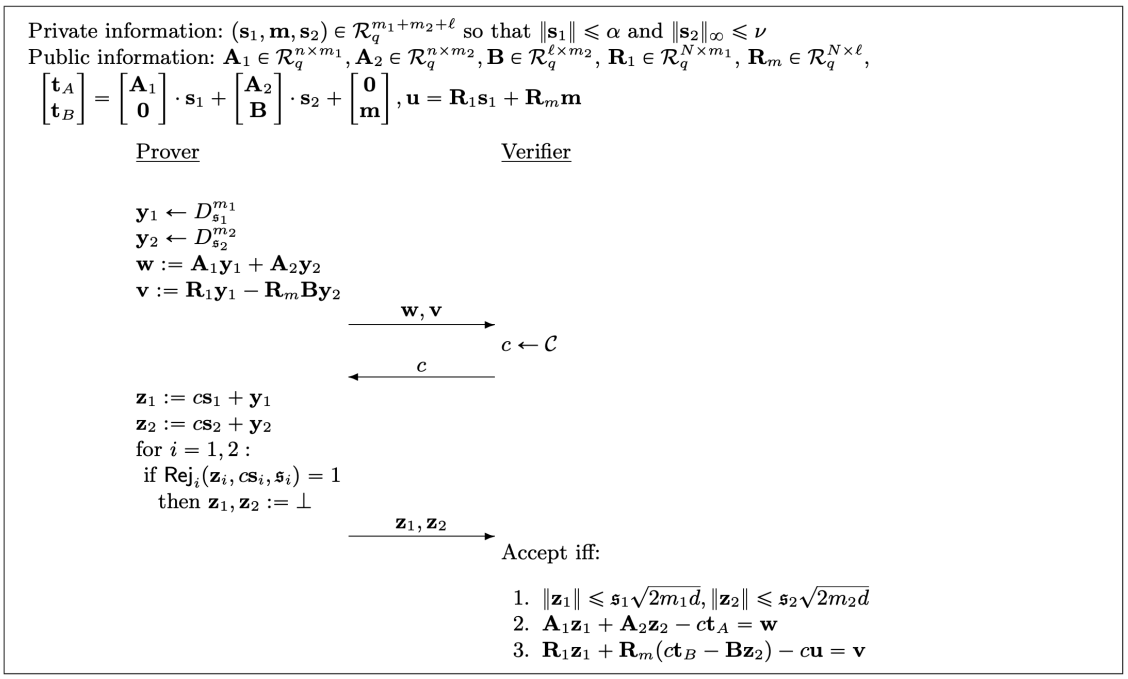

Figure C.1: Proof of Knowledge for Committed Messages, presented in the exact same way as Figure 4 in the original paper by Lyubashevsky et al. [5]