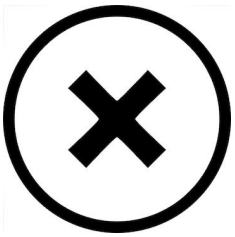

CROSS

Codes and Restricted Objects Signature Scheme

Submission to the NIST Post-Quantum Cryptography Standardization Process

Algorithm Specifications and Supporting Documentation

Version 1.2 - February 3, 2024

⊛ Marco Baldi, Polytechnic University of Marche Department for Communications Engineering

⊛ Alessandro Barenghi, Politecnico di Milano, Department of Electronics, Information and Bioengineering

⊛ Sebastian Bitzer, Technical University of Munich, Institute for Communications Engineering

⊛ Patrick Karl, Technical University of Munich, Chair of Security in Information Technology

⊛ Felice Manganiello, Clemson University, School of Mathematical and Statistical Sciences

⊛ Alessio Pavoni, Polytechnic University of Marche Department for Communications Engineering

⊛ Gerardo Pelosi, Politecnico di Milano, Department of Electronics, Information and Bioengineering

⊛ Paolo Santini, Polytechnic University of Marche Department for Communications Engineering

⊛ Jonas Schupp, Technical University of Munich, Chair of Security in Information Technology

⊛ Freeman Slaughter, Clemson University, School of Mathematical and Statistical Sciences

⊛ Antonia Wachter-Zeh, Technical University of Munich, Institute for Communications Engineering

⊛ Violetta Weger, Technical University of Munich, Institute for Communications Engineering

Submitters: The team above, with names listed alphabetically, is the principal submitter. There are no auxiliary submitters.

Inventors/Developers: Same as the principal submitter.

Implementation Owners: The submitters.

Email Address (preferred): info@cross-crypto.com

Postal Address and Telephone:
Paolo Santini
Polytechnic University of Marche
Department for Communications Engineering
Brecce Bianche 12
60131 Ancona
Italy
Tel: +39 071 2204128

Backup Contact Telephone and Address:
Sebastian Bitzer, Violetta Weger
Technical University of Munich
Institute for Communications Engineering
Theresienstraße 90
80333 Munich
Germany
Tel: +498928929051

Signature: (See "Statement by Each Submitter" or "Cover Sheet")

Contents

Change Log

This section summarizes the changes corresponding to different CROSS specification documents.

Version 1.2 Version 1.2 includes a set of minor updates with respect to version 1.1:

1. The lenghts of the required amount of randomness to be drawn from the CSPRNGs has been corrected in the codebase.
2. Two additional domain separation constants (c and dsc) are employed in computing cmt_0, cmt_1 and in the CSPRNG for transformation sampling.
3. The size of the signatures has been updated in Table 3 (λ bits were missing).
4. A new NP-hardness proof of R-SDP is included in Section 3.2.2.

Version 1.1 With regards to Version 1.0, the following changes have been made to this second version.

1. Improved security analysis for R-SDP(G): In Section 7.1.2, we consider an improved solver for the R-SDP(G), and we updated the parameters for CROSS R-SDP(G) accordingly. The parameters for the CROSS R-SDP instances are unchanged.
2. To prevent collision attacks on CSPRNG seeds, we include salting and a unique index per CSPRNG instance in each round of the signature. We detail these tweaks in the procedural description of CROSS, in Section 5.
3. To add hedging against multikey attacks we raise the length of the seeds for keypair generation to 2λ: this allows to prevent collision attacks relying on the collection of $2^{\frac{\lambda}{2}}$ keypairs. We updated Algorithm 1, Algorithm 2, and Algorithm 3 accordingly.
4. We propose parameters for an additional optimization corner that aims for even lower latency than the previous *fast* optimization corner (at the cost of larger signatures). While previous parameter sets featured a *small* and *fast* optimization targets, the new security categories provide a *small*, *balanced* (formerly *fast*), and *fast* version.
5. We report updated versions of Algorithm 1, Algorithm 2, and Algorithm 3, where we revised the generation of the $\mathbf{V}, \mathbf{W}$, values. Doing so saves a CSPRNG call during key generation, $2t$ CSPRNG during signature, and $2w$ CSPRNG calls during verification, at no security margin loss.
6. We revised the CSPRNG implementation strategy, extracting always a constant amount of pseudo-random bits from each CSPRNG call. We make this possible, in the rejection sampling scenarios, considering the amount of required bits so that the CSPRNG extraction fails with probability $\frac{1}{2^\lambda}$. We detail this CSPRNG strategy in Section 9. This approach makes constant time implementations easier.
7. We now consider the objects in their bit-packed representation when they are employed as the inputs of cryptographic hashed, reducing the amount of required computation. This is reported in Section 9.
8. We switched from SHA-3 as a cryptographic hash to SHAKE with a 2λ bit extracted string. This improves on the overall speed, while keeping the same security margin (as the bottleneck for attacks was SHA-3 collision resistance, which matches the one of the appropriate SHAKE with a 2λ bit output). We provide details in Section 9.
9. We report the performance figures from an optimized implementation for the Intel AVX2 instruction set in Section 10.
10. We report the memory footprints of a stack-size optimized portable implementation, fitting all our parameter sets on a Cortex-M4-based microcontroller, namely the STM32F407VG present on the STM32F4 Discovery board by STMicroelectronics employed by the pqm4 benchmarking project in Section 10.

1 Introduction

CROSS is a signature scheme obtained by applying the Fiat-Shamir transform to an interactive Zero-Knowledge (ZK) Identification protocol. CROSS is based on the so-called Restricted Syndrome Decoding Problem (R-SDP), an NP-complete problem that can be seen as a variant of the classical Syndrome Decoding Problem (SDP). Let $\mathbb{F}_p$ be the finite field with p elements (with p being a prime), and let $g \in \mathbb{F}_p^*$: in R-SDP, the solution to the decoding problem is required to take values in the cyclic subgroup $\mathbb{E} = \langle g \rangle \subseteq \mathbb{F}_p^*$.

Contrary to the SDP, we can even guarantee the uniqueness of the solution while having maximum Hamming weight: this corresponds to demanding that the solution to R-SDP is in $\mathbb{E}^n$ (here, n denotes the code length). We also consider a specialized version of R-SDP, called R-SDP(G), in which solutions are required to live in a subgroup $G \subset \mathbb{E}^n$, with $|G| < |\mathbb{E}^n| = z^n$. This allows us to work with more compact objects, taking $0 < m < n$ with $|G| = z^m$.

The hardness of solving R-SDP relates directly to that of SDP since the most efficient solvers are the same, namely, Information Set Decoding (ISD) algorithms. The same considerations apply to R-SDP(G), with the addition that the structure of G may be somehow exploited to speed up decoding attacks. Naturally, we considered this possibility and chose the subgroup G to rule out structural weaknesses. CROSS signatures are grouped into two main families, depending on the employed version of R-SDP. Instances based on R-SDP have slightly larger signatures but are more conservative (since the problem is analogous to SDP). Conversely, R-SDP(G) is a somewhat less standard problem but offers more compact signatures.

The main features of CROSS are listed below.

Based on restricted errors. CROSS is based on decoding a restricted error vector. Both the R-SDP and the R-SDP(G) are proven to be NP-hard. Using restricted errors has several benefits: generic decoders have a larger computational cost than the associated problem in the Hamming metric, and restricted errors can be more compactly represented using only $n \log_2(z)$ bits, respectively $m \log_2(z)$ bits for R-SDP(G). Restricted errors also allow us to avoid the need to compute permutations, a task which is challenging when constant-time implementation constraints are taken into account. Since this is quite a novel approach, we carefully study its security. In particular, we provide an analysis specifically tailored to the recommended choices for p and z.

Based on well-known and secure constructions. Using ZK protocols and the Fiat-Shamir transform to create a signature scheme comes with a long history and strong security aspects. In particular, the Existential Unforgeability under Chosen Message Attacks (EUF-CMA) security proof can be directly obtained from the soundness of the ZK protocol.

Based on an implementation friendly problem. The reduced size of the objects (parity-check matrices, error vectors, and syndromes) obtained when constructing CROSS from R-SDP/R-SDP(G) reduces the amount of computational effort compared to the traditional SDP in the Hamming metric, where the objects are larger. Furthermore, about half of the arithmetic operations in CROSS are performed over a smaller field $\mathbb{F}_z$, with z a prime number, where we can substitute modular multiplications with less expensive additions. Finally, the R-SDP/R-SDP(G) structure allows us to choose implementation-friendly values for p and z. Namely, we pick them both as Mersenne primes for R-SDP and as a small prime plus a Mersenne prime for R-SDP(G). We recall that modular arithmetic modulo a Mersenne prime does not require a divisor functional unit to be computed. Finally, the R-SDP/R-SDP(G) structure allows us to tune the security level smoothly by adjusting the chosen code length while keeping p and z fixed. This, in turn, allows for the implementation of only two sets of arithmetic primitives, which reduces the code size (in software implementations, where it is critical in Flash-memory-constrained microcontrollers) or the required silicon area (in hardware implementations).

Compact signatures. When using R-SDP instead of the classical SDP, the cost of ISD algorithms, the state-of-the-art solvers for decoding problems, increases. This allows us to select smaller parameters to attain the desired security levels, positively impacting both signature sizes and computational complexity. In addition, the linear transformations on the considered error vectors are much smaller than the linear isometries in the Hamming metric. Consider that isometries have been instrumental in designing several ZK protocols based on SDP; when utilized in R-SDP, it is enough to use a vector in $\mathbb{F}_z^n$ to represent restricted vectors and transformations on them. This greatly reduces the binary size of isometries

compared to other settings, such as the Hamming metric. Consequently, this efficient representation significantly reduces the communication cost of the ZK protocol we use in CROSS.

Simplicity. CROSS has been designed to strive for simplicity and algorithmic efficiency. Indeed, key generation, signature, and verification primitives in CROSS require only consolidated symmetric primitives (such as CSPRNGs and cryptographic hashes) and vector/matrix operations among small elements. This structural simplicity allows us to reduce the amount of implementation footguns [1], i.e., potential points for implementation errors that lead to vulnerabilities, either directly or through the exploitation of side channel information leakage. The structural simplicity also allows for a straightforward, constant-time implementation of the scheme. Indeed, all the linear algebra operations are natively performed in a memory-access-pattern oblivious way, while CSPRNGs and hashes are available as consolidated and tested constant-time implementations.

Achieving different trade-offs. For R-SDP and R-SDP(G), we propose different instances of CROSS to achieve heterogeneous trade-offs. Indeed, the signature size can be traded with the computational complexity: executing more rounds of the ZK protocol increases latency but makes signatures more compact. For each security category and each version of the problem (i.e., either R-SDP or R-SDP(G)), we consider three variants of CROSS.

- CROSS-fast: small values of t (the number of repetitions for the ZK protocol). This variant aims at fast signature generation and verification.
- CROSS-small: large values of t. This variant aims at achieving short signatures.
- CROSS-balanced: moderate values of t. This variant comes as a trade-off between the other two variants. Signatures are larger than those of the short variant but shorter than those of the fast variant. Analogously, timings are worse than those of the fast variant but are better than those of the short variant.

Small Public Key Sizes. We can compress both the private and the public keys of CROSS. In particular, the private key is reduced to its optimal size: a single random seed. All the elements in the public key, apart from a short vector over $\mathbb{F}_p$, can also be regenerated from a seed with acceptable computational overhead. This allows us to compress the public key size - in practice, less than 121 B for R-SDP and less than 74 B for R-SDP(G) - for all NIST security categories. These reduced key sizes allow CROSS keypairs to fit even on constrained embedded devices where persistent (flash) memory may be scarce, such as in low-end microcontrollers.

1.1 Historical Background

The first code-based Zero-Knowledge (ZK) identification protocol was proposed by Stern in 1993 [49]. The scheme is based on a couple of fundamental ideas that are simple but rather powerful: a secret, a low-weight vector can be masked using isometries and sums with random vectors. These very same ideas have been used in subsequent schemes such as [51, 2] and [26]. The latter, which we will refer to as CVE, is the first ZK protocol based on SDP for codes over non-binary finite fields.

As it is well known, interactive ZK protocols can be turned into signature schemes using the celebrated Fiat-Shamir transform [34]. As a very positive aspect, the resulting schemes generically have quite compact public keys and benefit from high-security guarantees since the proof of knowledge is constructed from a truly random instance of some hard problem (i.e., neither a trapdoor nor an ad-hoc code construction is required). The other required security assumptions come from fundamental cryptographic tools, e.g., the hardness of finding collisions for a hash function. In this sense, signature schemes obtained from the Fiat-Shamir paradigm achieve security in an exceptionally pure way.

However, for all the aforementioned protocols, the resulting signatures are relatively large, owing to the need for several parallel executions to reduce the overall soundness error[1]. For this reason, signatures obtained from the Fiat-Shamir paradigm have been deemed impractical for several years. Now, the scenario has drastically changed. Indeed, in recent years quite some effort has been dedicated to the development of new ZK protocols, with signatures becoming shorter and shorter [17, 21, 37, 32, 33, 3,

[1] If the soundness error is not negligible in the security parameter, the cost of forgery attacks may be below the claimed security level.

20, 30, 19, 41]. Techniques to reduce signature sizes consist, for instance, of using a binary tree structure to generate randomness, Merkle trees to reduce commitment sizes, and an unbalanced distribution for the verifier's messages (the challenges that admit a more compact response are asked more frequently). Other popular approaches are the so-called protocol-with-helper [18] and the Multi-Party Computation (MPC) in-the-head (MPCitH) paradigm, proposed in [42] and first adapted to the code-based setting in [33]. Notice that several modern schemes can be thought of as clever and highly optimized re-writings of historical proposals such as [49] and [26], since they are still based on the very same fundamental ideas (e.g., masking through isometries). The reductions in the signature size are sometimes motivated by reductions in the soundness error; however, this comes at the cost of a non-trivial computational overhead for both the prover and the verifier (this is the case for protocols-with-helper or using the MPCitH approach).

In [8], the authors study ZK identification protocols based on the R-SDP setting (they also introduce the R-SDP(G) variant). The paper shows that R-SDP appears to be better fitted for ZK protocols than the classical SDP. Indeed, messages become significantly shorter since codes become smaller, and isometries can be represented more compactly. In [8], two main protocols are investigated: GPS [37] and BG [19], originally proposed for the Permuted Kernel Problem (PKP). The authors show that these schemes would enjoy significantly shorter signatures when adapted to use R-SDP and identify BG as a strong candidate, with signatures in the order of 7 kB for 128 bits of security. Benchmarks for a proof-of-concept implementation are also provided: timings are very promising and confirm that the R-SDP setting can lead to efficient and practical schemes.

Notice that both the GPS and BG protocols use the same ideas of the CVE scheme [26]. GPS is a protocol-with-helper and, like all such schemes, can achieve competitive signatures only if quite a few rounds are performed - this leads to slower signature generation and verification procedures. The BG protocol amplifies the soundness of each round by repeating, dozens of times, the operations that would have been performed in a single round of CVE. Even though these operations are inherently fast (calling hashes and PRNG, applying isometries, and summing vectors), this intense repetition would prevent the subsequent signature scheme from being fast.

CROSS is designed for simplicity and very high algorithmic efficiency. For this reason, we chose to pursue an alternative approach to the schemes presented in [8]. Indeed, the underlying ZK protocol employed in CROSS corresponds to the basic CVE protocol plus standard optimizations (PRNG calls, Merkle trees, and unbalanced challenges). Even though the original CVE protocol has a rather large soundness error (approximately 1/2), this is not an issue for R-SDP, as messages become very short. To achieve a negligible soundness error, CROSS requires a moderate number of rounds; however, each round corresponds to a single round of the CVE protocol and, as such, is intrinsically simple and computationally light. In the end, if compared with the protocols in [8], CROSS performs fewer operations: we pay a small price in signature size but gain an immense boost for what concerns computational complexity.

1.2 Overview of Basic Idea

The underlying problem. CROSS is built on a Zero-Knowledge (ZK) protocol for the Restricted Syndrome Decoding Problem (R-SDP). R-SDP has been first introduced in [7] and subsequently generalized in [8]. The R-SDP depends on a restricted set $\mathbb{E}$ which is given by an element $g \in \mathbb{F}_p$ of multiplicative order z, namely we define $\mathbb{E} = \{g^i \mid i \in \{1, \dots, z\}\}$. Given a parity-check matrix $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, a syndrome $\mathbf{s} \in \mathbb{F}_p^{n-k}$ and a restricted set $\mathbb{E}$, R-SDP asks to find a vector $\mathbf{e} \in \mathbb{E}^n$ such that $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$. In the following sections, we will study the R-SDP in detail.

The crucial difference to the classical SDP is that we consider error vectors of full weight. The hardness of decoding now stems from the restriction that the entries of the error vector can only live in $\mathbb{E}$, a subgroup of the multiplicative group of the finite field. In the R-SDP, this restriction of the ambient space replaces the traditional weight constraint of the standard SDP while still enjoying an NP-hardness proof. Having error vectors of full weight allows us to avoid the need to send permutations within the ZK protocol. This yields another significant reduction in the communication cost and simplifies implementations (as constant-time implementation of permutations is non-trivial).

The Zero-Knowledge protocol. The ZK identification protocol CROSS-ID is an adaption of the classical CVE protocol [26]. The signer chooses a random parity-check matrix $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$ (the matrix is generated from a seed) and an element $g \in \mathbb{F}_p^*$ of prime order z. This is the generator of the subgroup

$\mathbb{E} = \{g^i \mid i \in \{1, \ldots, z\}\}$. The signer samples at random a vector ℓ in $\mathbb{F}_z^n$ (we always choose z so that it is prime), then computes $\mathbf{e} = g^\ell = (g^{\ell_1}, \ldots, g^{\ell_n}) \in \mathbb{E}^n$ and its syndrome $\mathbf{s} = \mathbf{e}\mathbf{H}^\top \in \mathbb{F}_p^{n-k}$. The signer publishes both the seed for $\mathbf{H}$ and $\mathbf{s}$, while the values of p, g, and z are fixed. Within one round of the CROSS-ID protocol, the signer will either prove that the secret error vector $\mathbf{e}$ satisfies the syndrome equations $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$ or is such that $\mathbf{e} \in \mathbb{E}^n$. This is done with a 5-pass protocol, which can be classified as a $q2$-Identification scheme, i.e., the first challenge can take q different values (in our case, $q = p-1$), while the second challenge can take two different values (either 0 or 1). As in the original CVE scheme, this protocol achieves $(2, 2)$-out-of-$(q, 2)$ special soundness (in the following, we will sometimes say $(2, 2)$-special soundness for simplicity) and consequently has soundness error $\frac{q+1}{2q} = \frac{p}{2(p-1)} \approx \frac{1}{2}$.

The signature scheme. Using the Fiat-Shamir transform, we can render the CROSS-ID protocol into a signature scheme; due to the protocol being a $q2$-Identification scheme, we immediately obtain the EUF-CMA security proof. We will reduce the resulting signature sizes further by employing well-known techniques: we send the hash of the commitments for all t rounds beforehand and use a Merkle tree to recover the commitments for the challenges $b = 0$. This reduces the signature size further as we need fewer hashes than using the Merkle tree for both challenges.

Since sending the response to the challenge $b = 1$ is only a seed and much smaller than the response to $b = 0$, we use a weighted challenge vector $\mathbf{b} = (b^{(1)}, \ldots, b^{(t)}) \in \{0, 1\}^t$ to optimize the signature size. Note that this unbalanced distribution in the second challenge value affects the cost of forgeries: we consider attacks such as [43], which we adapt and optimize to our considered setting.

2 Notation

This section establishes the notation and conventions we will adopt throughout this document.

2.1 Mathematical Notation

We use $[a; b]$ to denote the set of all reals $x \in \mathbb{R}$ such that $a \leq x \leq b$. For a finite set $A \subset \{1, \ldots, n\}$, the expression $a \xleftarrow{\$} A$ means that a is chosen uniformly at random from A. In addition, we denote by $|A|$ the cardinality of A, by $A^C = \{1, \ldots, n\} \setminus A$ its complement, and by $A_0 = A \cup \{0\}$ the set union with the 0 element.

For m a positive integer we denote by $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$ the ring of integers modulo m. Let p be a prime: we denote by $\mathbb{F}_p$ the finite field of order p and by $\mathbb{F}_p^*$ its multiplicative group. We denote by $\mathrm{ord}(g)$ the multiplicative order of a $g \in \mathbb{F}_p^*$.

We use uppercase (resp. lowercase) letters to indicate matrices (resp. vectors). If J is a set, we use $\mathbf{A}_J$ to denote the matrix formed by the columns of $\mathbf{A}$ indexed by J; a similar notation will be used for vectors. The identity matrix of size m is denoted as $\mathbf{I}_m$. We use $\mathbf{0}$ to denote both the null matrix and the null vector without specifying dimensions (which will always be apparent from the context). Finally, we denote by h_p the p-ary entropy function.

2.2 Cryptographic Notation

This document uses conventional cryptographic notations, e.g., λ denotes a security parameter. We will consider $\lambda \in \{128, 192, 256\}$, as this greatly helps in establishing how symmetric primitives are employed. We list, in the following, the cryptographic functions which are used in this document:

- a cryptographically secure hash function is indicated as $\mathsf{Hash} : \{0, 1\}^* \to \{0, 1\}^{2\lambda}$, with input of any size and digest with binary length 2λ;
- a Merkle tree constructed from t leaves $(a^{(t)}, \ldots, a^{(t)})$ is indicated as $\mathcal{T} = \mathsf{MerkleTree}(a^{(1)}, \cdots, a^{(t)})$. The root of the tree is extracted as $\mathcal{T}.\mathsf{Root}(\,)$, while the function that computes the Merkle proof for the leaves in the positions indexed by a set J is $\mathcal{T}.\mathsf{MerkleProof}(J)$. Recomputation of the root, starting from some leaves $\{a^{(i)}\}_{i \in J}$ and the associated Merkle proofs $\mathtt{MerkleProof}$, is indicated as $\mathsf{VerifyMerkleRoot}\big(\{c_1^{(i)}\}_{i \notin J}, \mathtt{MerkleProof}\big)$. The hash function employed to construct the tree has digests of length 2λ, that is, any leaf in the tree is a binary string with length 2λ;
- for a PRNG tree constructed from a root $\mathtt{Root}$, we indicate the function that computes the t leaves $\big(\mathsf{Seed}^{(1)}, \ldots, \mathsf{Seed}^{(t)}\big)$ as $\mathsf{SeedTree}(\mathtt{Root})$. All generated seeds have binary length λ. To extract the

path SeedPath, which can be used to generate the seeds indexed by a set J, we use the function SeedPath($\texttt{Root}, J$). To regenerate such seeds, we use the function GetSeeds($\texttt{SeedPath}, J$). We will use salted trees, i.e., will generate the PRNG tree using as a root a length-λ master seed MSeed, concatenated with a length-2λ string Salt. This will be made explicit by using $\texttt{MSeed}, \texttt{Salt}$.

The notation $a \xleftarrow{\texttt{Seed}} A$ means that a is sampled through a deterministic cryptographically secure random generator, outputting elements as a uniform pick from A, and being initialized with Seed as the seed value.

3 Preliminaries

3.1 Signatures from Zero-Knowledge Protocols

To be concise, we do not give general definitions and properties that can be easily found in the literature (see, e.g., [31, 33]), but only recall what is strictly necessary for CROSS.

Zero-Knowledge Identification Schemes A ZK protocol, also called ZK Identification scheme or ZK Proof of Knowledge, is an interactive protocol in which a *prover* $\mathcal{P}$ aims to convince a *verifier* $\mathcal{V}$ that they know a secret that verifies some public statement, without revealing said secret. The two parties interact by exchanging messages. We consider only protocols in which five messages are sent, with the prover always sending the first and last. This type of protocol is usually called 5-pass, and messages are given specific names; to this end, see Figure 1. The first message sent by the prover is called *commitment*. The two other messages sent by the prover are called *responses*, while the two messages sent by the verifier are called *challenges*. The challenges are drawn uniformly at random from two sets C_1 and C_2, respectively.

Thus, one execution of the protocol corresponds to an (ordered) set of messages of the form

$$T = \Big(\texttt{Com}, \texttt{Ch}_1, \texttt{Rsp}_1, \texttt{Ch}_2, \texttt{Rsp}_2\Big).$$

We call T a *transcript* of the protocol. At the end of the interaction, the verifier returns a value $\texttt{Out} \in \{0;1\}$ (which is computed as a function of all the messages that have been exchanged in the round). We say T is an *accepting* transcript if it corresponds to the verifier returning 1.

The following properties must be satisfied for a protocol as in Figure 1 to be a ZK protocol.

- *Completeness*: an honest prover gets always accepted: a round initiated by a prover who knows sk and follows the protocol always ends in the verifier outputting 1.
- *Zero Knowledge*: the interaction between $\mathcal{P}$ and $\mathcal{V}$ does not reveal any useful information about the secret sk. This implies that knowing the challenge values in advance, a cheating prover can produce accepting transcripts that are indistinguishable from those produced by an honest prover.

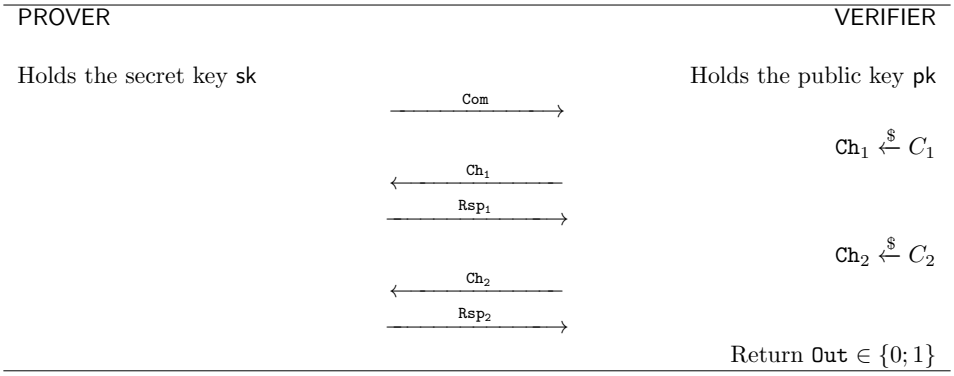

Figure 1: 5-pass interactive protocol

- *Soundness*: when $\mathcal{V}$ is honest (i.e., challenges are sampled from the uniform distributions over C_1 and C_2), a malicious prover $\mathcal{P}^*$ (i.e., a prover that does not know $\mathtt{sk}$) can only convince $\mathcal{V}$ with some probability $\varepsilon < 1$. The quantity ε is typically called *soundness error* and, in practice, corresponds to the probability that a cheating prover can correctly guess which subset of challenges will be sampled by the verifier (plus a negligible quantity, corresponding to the likelihood that the prover can solve some hard problem).

When t parallel repetitions of a N-pass protocol with soundness error ε are considered, we obtain a new N-pass protocol with soundness error ε^t. To distinguish between different rounds, we will use the superscript $^{(i)}$, e.g., $\mathtt{Com}^{(i)}$ will denote the commitment in the i-th round.

Fiat-Shamir transformation The Fiat-Shamir transformation [34] is a standard technique to turn an interactive ZK protocol into a signature scheme. The transform makes the scheme non-interactive, with the prover simulating the verifier by generating the challenges as the output of some one-way function (e.g., a hash), using all the former messages as input. The message to be signed $\mathtt{Msg}$ is provided as another input to the hash function; this way, the transcript becomes associated with $\mathtt{Msg}$. To sum up, the Fiat-Shamir transform operates as follows:

1) generate the commitment
$$\mathtt{Com} = \left(\mathtt{Com}^{(1)} \ , \ \mathtt{Com}^{(2)} \ , \ \ldots \ , \ \mathtt{Com}^{(t)}\right);$$

2) generate the first challenge as
$$\mathtt{Ch}_1 = \left(\mathtt{Ch}_1^{(1)} \ , \ \mathtt{Ch}_1^{(2)} \ , \ \ldots \ , \ \mathtt{Ch}_1^{(t)}\right) = \mathsf{Hash}\Big(\mathtt{Msg} \ , \ \mathtt{Com}\Big);$$

3) compute the first response
$$\mathtt{Rsp}_1 = \left(\mathtt{Rsp}_1^{(1)} \ , \ \mathtt{Rsp}_1^{(2)} \ , \ \ldots \ , \ \mathtt{Rsp}_1^{(t)}\right);$$

4) generate the second challenge as
$$\mathtt{Ch}_2 = \left(\mathtt{Ch}_2^{(1)} \ , \ \mathtt{Ch}_2^{(2)} \ , \ \ldots \ , \ \mathtt{Ch}_2^{(t)}\right) = \mathsf{Hash}\Big(\mathtt{Msg} \ , \ \mathtt{Com} \ , \ \mathtt{Ch}_1 \ , \ \mathtt{Rsp}_1\Big);$$

5) compute the second response
$$\mathtt{Rsp}_2 = \left(\mathtt{Rsp}_2^{(1)} \ , \ \mathtt{Rsp}_2^{(2)} \ , \ \ldots \ , \ \mathtt{Rsp}_2^{(t)}\right).$$

The transcript of the protocol gives the resulting signature. In other words, the signing algorithm consists of doing what the prover $\mathcal{P}$ would do in an honest execution of the protocol, except that the prover locally generates the challenges. The verification algorithm, instead, emulates what the verifier $\mathcal{V}$ would do after all messages have been received. Depending on the transcript, the output will be 1 (the signature is accepted) or 0 (the signature is rejected).

Intuitively, the obtained signature scheme is secure since each challenge is generated as a one-way, pseudo-random function of all the previously exchanged messages. As an example, changing the commitment $\mathtt{Com}$ after the first challenge $\mathtt{Ch}_1$ has been generated is not possible (unless hash collisions are found) since even a slight modification in $\mathtt{Com}$ would lead to unpredictable changes in the challenge $\mathtt{Ch}_1$.

To reduce the signature size, the challenges are normally omitted since they can be re-generated during verification. In other words, the final signature will be of the form
$$\mathtt{Sign} = \left(\mathtt{Salt} \ , \ \mathtt{Com} \ , \ \mathtt{Rsp}_1 \ , \ \mathtt{Rsp}_2\right).$$

Commitments, seeds, and salt Commitments are typically implemented with hash functions: to commit to a value x, the prover computes and sends $\mathtt{Com} = \mathsf{Hash}(x)$. Formally, commitment functions should be hiding and binding.[2] Informally, the meaning of these properties is as follows:

[2]Notice that these properties come in different flavors, e.g., a commitment function may be statistically or computationally binding.

- *hiding*: given Com, no information about x can be determined;
- *binding*: it is unfeasible to find two distinct messages $x \neq x'$ which correspond to the same commitment Com. Thus, the prover cannot modify x after committing to it.

Also, sometimes the value to which the prover commits is a length-λ seed. Without proper precautions, attacks may find collisions in the commitments in time $O(2^{\lambda/2})$. This way, an attacker may learn a value even though the prover does not disclose it. Examples of attacks of this type and efficient countermeasures are provided in [27]. In practice, it is enough to sample, for each newly generated signature, a fresh salt of size 2λ. The salt is used as another input to the hash function: the commitment of a message x is computed as

$$\texttt{Com} = \mathsf{Hash}(x, \texttt{Salt}).$$

The salt is communicated with the signature, so the verifier recomputes and verifies the commitment.

Notice that, however, an attacker may still exploit the fact that the same salt is used for all t rounds of each produced signature: finding a collision for one of these t rounds would cost (slightly) less than $O(2^{\lambda})$. Even though, in practice, the values of t are so small that the practical advantage would be irrelevant, one can also protect from this attack with a simple countermeasure (again, recommended in [27]). Namely, in round i, the prover commits to a value x by sending

$$\texttt{Com} = \mathsf{Hash}(x, \texttt{Salt}, i).$$

Fiat-Shamir transformation of 5-pass schemes The Fiat-Shamir transformation of a ZK interactive protocol with soundness error ε leads to a signature scheme admitting forgery attacks running in time $O(\varepsilon^{-1})$. Indeed, a cheating prover can repeatedly simulate the protocol by guessing the challenge values and preparing coherent commitments and responses with such guesses. With probability ε, the generated challenge values will be coherent with the initial guess: so this requires ε^{-1} attempts on average. Formally, one can prove that signature schemes obtained via the Fiat-Shamir paradigm achieve EUF-CMA security. However, to protect from forgeries, some caution is needed.

When considering the Fiat-Shamir transformation of a ZK protocol obtained by t parallel executions of a scheme with soundness error ε (as we do for CROSS), the above reasoning results in a forgery attack with cost $O(\varepsilon^{-t})$. So, one may choose t such that $\varepsilon^{-t} > 2^{\lambda}$: this is sometimes called the ε^t-heuristic. However, for the case of 5-pass schemes, there exist instances in which the ε^t-heuristic fails, in the sense that it does not rule out forgery attacks with time complexity lower than 2^{λ}. An example is the attack in [43], which is primarily described for MQDSS but applies to several 5-pass schemes. The idea is that a malicious prover can take advantage of non-interactivity and, ultimately, produce transcripts that would get accepted with a probability greater than ε^t. Note that the existence of this attack does not invalidate the security proof for the scheme: the attack takes advantage of the proof being non-tight. Indeed, the security proof for MQDSS [41] uses the forking lemma and is consequently non-tight: this implies that the scheme asymptotically achieves EUF-CMA security but does not tell us how t should be chosen.

The authors of [43] provide accurate formulas for the cost of such forgery attacks. According to their analysis, the cost depends only on t and the sizes of the challenges spaces, $|C_1|$ and $|C_2|$. Since C_1 and C_2 are essentially fixed (depending on the underlying ZK protocol), one must choose t large enough so the cost is above 2^{λ}. In practice, when the attack in [43] applies, the actual value of t is (moderately) larger than that given by the ε^t-heuristic.

$q2$-Identification schemes CROSS is based on a ZK protocol which follows the structure of $q2$-Identification schemes employed in MQDSS [41]. A ZK protocol is classified as a $q2$-Identification scheme when it possesses the following properties:

i) $|C_1| = q$;

ii) $|C_2| = 2$;

iii) The probability that Com takes a given value is negligible in the security parameter.

This classification is helpful since following [41], it holds that the Fiat-Shamir transformation of parallel executions of a $q2$-Identification protocol results in a signature scheme that is EUF-CMA secure. The proof is based on the existence of a $q2$-Extractor [41], that is, a Probabilistic Polynomial-Time algorithm

$\mathcal{E}$ that returns (with non-negligible success probability $1-\varepsilon$) the secret $\mathtt{sk}$, given four distinct accepting transcripts of the form

$$T = (\mathtt{Com}\ ,\ \mathtt{Ch}_1\ ,\ \mathtt{Rsp}_1\ ,\ \mathtt{Ch}_2\ ,\ \mathtt{Rsp}_2), \qquad T' = (\mathtt{Com}\ ,\ \mathtt{Ch}'_1\ ,\ \mathtt{Rsp}'_1\ ,\ \mathtt{Ch}'_2\ ,\ \mathtt{Rsp}'_2),$$
$$T'' = (\mathtt{Com}\ ,\ \mathtt{Ch}''_1\ ,\ \mathtt{Rsp}''_1\ ,\ \mathtt{Ch}''_2\ ,\ \mathtt{Rsp}''_2), \qquad T''' = (\mathtt{Com}\ ,\ \mathtt{Ch}'''_1\ ,\ \mathtt{Rsp}'''_1\ ,\ \mathtt{Ch}'''_2\ ,\ \mathtt{Rsp}'''_2),$$

with

$$\mathtt{Ch}_1 = \mathtt{Ch}''_1 \neq \mathtt{Ch}'_1 = \mathtt{Ch}'''_1,$$
$$\mathtt{Ch}_2 = \mathtt{Ch}''_2 \neq \mathtt{Ch}'_2 = \mathtt{Ch}'''_2.$$

For the case of $q2$-Identification schemes, proving existence of the $q2$-Extractor is the same as showing that the protocol is $(2,2)$-out-of-$(q,2)$ special sound (see for instance [4, 5]), which immediately implies soundness error

$$\varepsilon = 1 - \left(1 - \frac{1}{q}\right)\left(1 - \frac{1}{2}\right) = \frac{q+1}{2q}.$$

To sum up, once a ZK protocol is classified as a $q2$-Identification protocol, this guarantees that the resulting signature scheme, obtained by Fiat-Shamir transforming t parallel executions, is EUF-CMA secure. To choose the number t of parallel repetitions, we remark that one should consider the cost of forgery attacks as the one in [43].

3.2 Coding Theory

3.2.1 Classical Coding Theory

A *linear code* $\mathcal{C}$ over the finite field $\mathbb{F}_p$ with length n and dimension $k \leq n$ is a k-dimensional linear subspace of $\mathbb{F}_p^n$. A compact representation for a code is a *generator matrix*, that is, a full-rank $\mathbf{G} \in \mathbb{F}_p^{k\times n}$ such that $\mathcal{C} = \{\mathbf{uG} \mid \mathbf{u} \in \mathbb{F}_p^k\}$. We say that a code of length n and dimension k has *rate* $R = \frac{k}{n}$ and *redundancy* $r = n - k$. Equivalently, one can represent a code through a full-rank $\mathbf{H} \in \mathbb{F}_p^{r\times n}$, called *parity-check matrix*, such that $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_p^n \mid \mathbf{cH}^\top = \mathbf{0}\}$. The *syndrome* of some $\mathbf{x} \in \mathbb{F}_p^n$ is the length-r vector $\mathbf{s} = \mathbf{xH}^\top$. A set $J \subseteq \{1,\ldots,n\}$ of size k is called *information set* for $\mathcal{C}$ if $|\mathcal{C}_J| = p^k$, where $\mathcal{C}_J = \{\mathbf{c}_J \mid \mathbf{c} \in \mathcal{C}\}$. It follows that $\mathbf{G}_J$ and $\mathbf{H}_{J^C}$ are invertible matrices. We say that a generator matrix, respectively, a parity-check matrix is in *systematic form* (with respect to the information set J), if $\mathbf{G}_J = \mathbf{I}_k$, respectively $\mathbf{H}_{J^C} = \mathbf{I}_{n-k}$. We endow the vector space $\mathbb{F}_p^n$ with the *Hamming metric*: given $\mathbf{x} \in \mathbb{F}_p^n$, its Hamming weight $\mathrm{wt}(\mathbf{x})$ is the number of non-zero entries. The *minimum distance* of a linear code is given by $d = \min\{\mathrm{wt}(\mathbf{c}) \mid \mathbf{c} \in \mathcal{C}, \mathbf{c} \neq \mathbf{0}\}$. Recall that the Gilbert-Varshamov (GV) bound states that $R \geq 1 - h_p(d/n)$. It is well known that a random code attains the Gilbert-Varshamov bound for large enough length n, meaning, for a random code, we may assume $\delta = d/n = h_p^{-1}(1-R)$. Code-based cryptography usually relies on the following NP-complete problem [14, 10].

Problem 1. ***Syndrome Decoding Problem (SDP)***
Given $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $t \in \mathbb{N}$, and $\mathbf{s} \in \mathbb{F}_p^{n-k}$, decide if there exists an $\mathbf{e} \in \mathbb{F}_p^n$ such that $\mathrm{wt}(\mathbf{e}) \leq t$ and $\mathbf{eH}^\top = \mathbf{s}$.

We usually assume that the instance of the SDP is chosen uniformly at random, and therefore, the code with the parity-check matrix $\mathbf{H}$ attains the GV bound. If the target weight t is less than the minimum distance δn of the Gilbert-Varshamov bound, we expect to have a unique solution, if we have any, since on average the number of solutions is given by $p^{n(h_p(\delta)-1+R)} \leq 1$.

3.2.2 Restricted Decoding Problem

This problem was first introduced for $z = 2$ in [7], then later for any z in [8].

Let us consider $g \in \mathbb{F}_p^*$ of prime order z and the subgroup $\mathbb{E} = \{g^i \mid i \in \{1,\ldots,z\}\} \subset \mathbb{F}_p^*$. Let us denote by $\star$ the component-wise multiplication of vectors.

The Restricted Syndrome Decoding Problem (R-SDP), first introduced in [7], reads as follows.

Problem 2. ***Restricted Syndrome Decoding Problem (R-SDP)***
Given $g \in \mathbb{F}_p^*$ of order z, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_p^{n-k}$, and $\mathbb{E} = \{g^i \mid i \in \{1,\ldots,z\}\} \subset \mathbb{F}_p^*$, decide if there exists $\mathbf{e} \in \mathbb{E}^n$ such that $\mathbf{eH}^\top = \mathbf{s}$.

The R-SDP is strongly related to other well-known hard problems. For example, when $z = p-1$, the R-SDP is close to the classical SDP; if $z = 1$, the R-SDP is similar to the Subset Sum Problem (SSP) over finite fields. Consequently, it is unsurprising that the R-SDP is NP-complete for any choice of $\mathbb{E}$.

Theorem 3. The R-SDP (Problem 2) is NP-complete.

The proof for the NP-completeness can be found in [53], where the authors show that the additivity of the weight and having a unit in the ambient space of the error vector is enough to obtain NP-completeness. For the sake of completeness, we add the proof here.

Proof. Recall the NP-hard 3-Dimensional Matching (3DM) problem, where one is given the instance $T = \{b_1, \ldots, b_t\}$, with $|T| = t, U \subset T \times T \times T$ and $|U| = u$ and asks whether there exists a $W \subset U$ with $|W| = t$ and no two words in W coincide in any position.

Recall that the original SDP has a reduction from 3DM, through the following construction: let $\mathbf{H} \in \mathbb{F}_p^{(3t)\times u}$ be the incidence matrix, i.e., each column of $\mathbf{H}$ corresponds to a word in U and the rows correspond to $T \times T \times T$, thus the rows $\{1, \ldots, t\}$ correspond to the first position of the word $\mathbf{u}$, the rows $\{t+1, \ldots, 2t\}$ correspond to the second position of $\mathbf{u}$ and the rows $\{2t+1, \ldots, 3t\}$ correspond to the third position of $\mathbf{u}$. More formally, let $T = \{b_1, \ldots, b_t\}$, $U = \{\mathbf{a}_1, \ldots, \mathbf{a}_u\}$ and

- for $i \in \{1, \ldots, t\}$, we set $h_{i,j} = 1$ if $\mathbf{a}_j[1] = b_i$ and $h_{i,j} = 0$ else,
- for $i \in \{t+1, \ldots, 2t\}$, we set $h_{i,j} = 1$ if $\mathbf{a}_j[2] = b_i$ and $h_{i,j} = 0$ else,
- for $i \in \{2t+1, \ldots, 3t\}$, we set $h_{i,j} = 1$ if $\mathbf{a}_j[3] = b_i$ and $h_{i,j} = 0$ else.

We also set $\mathbf{s} \in \mathbb{F}_p^{3t}$ be the all one vector. From the original reduction, we know that any solution $\mathbf{e} \in \mathbb{F}_p^u$ with $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$ has weight t (it has weight at least t as we need to reach the all one vector in $\mathbb{F}_p^{3t}$ and each column gives weight 3, and it has weight at most t as p is larger than u and else we would get syndrome entries larger than 1) and its support corresponds to the solution W. That is the columns of $\mathbf{H}$ indexed by the support of $\mathbf{e}$ are the t words in W.

The polynomial reduction from 3DM to R-SDP uses this construction as well. Let T of size t and $U \subset T \times T \times T$ of size u be an instance of 3DM. Let $\mathbf{H} \in \mathbb{F}_p^{(3t)\times u}$ be the incidence matrix and let

$$\widetilde{\mathbf{H}} = \begin{pmatrix} \mathbf{H} & -g \star \mathbf{H} \\ \mathrm{Id}_u & \mathrm{Id}_u \end{pmatrix} \in \mathbb{F}_p^{(3t+u)\times 2u}$$

be a parity-check matrix. Let us consider the syndrome $(\mathbf{s}, \mathbf{s}') \in \mathbb{F}_p^{3t+u}$ with $\mathbf{s} = (1-g^2, \ldots, 1-g^2) \in \mathbb{F}_p^{3t}$ and $\mathbf{s}' = (1+g, \ldots, 1+g) \in \mathbb{F}_p^u$. Thus, the instance of R-SDP given by $\widetilde{\mathbf{H}}$ and $(\mathbf{s}, \mathbf{s}')$ is asking for $(\mathbf{e}, \mathbf{e}') \in \mathbb{E}^{2u}$ such that

$$(\mathbf{e}, \mathbf{e}')\widetilde{\mathbf{H}}^\top = (\mathbf{s}, \mathbf{s}'),$$

where $\mathbb{E} = \{g^i \mid i \in \{0, \ldots, z-1\}\}$. By assumption of R-SDP, we use a g of order $2 < z < q-1$.

We consider two cases.

1. Assume that the R-SDP solver returns "yes", i.e., there exists $\mathbf{e}, \mathbf{e}' \in \mathbb{E}^u$ such that $(\mathbf{e}, \mathbf{e}')\widetilde{\mathbf{H}}^\top = (\mathbf{s}, \mathbf{s}')$. Hence,

$$\mathbf{H}\mathbf{e}^\top - g \star \mathbf{H}\mathbf{e}'^\top = (1-g^2, \ldots, 1-g^2)^\top,$$
$$\mathbf{e} + \mathbf{e}' = (1+g, \ldots, 1+g).$$

   Hence, for each $i \in \{1, \ldots, u\}$ we have $e_i + e'_i = 1 + g$. Let us assume (we later show that this hypothesis is not needed, but it facilitates the proof) that the only elements in $\mathbb{E}$ that add to $1+g$ is 1 and g.

   Hence, whenever $e_i = 1$, we must have $e'_i = g$ and whenever $e_i = g$, we must have $e'_i = 1$. Thus, we split $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_g$ and $\mathbf{e}' = \mathbf{e}'_1 + \mathbf{e}'_g$ where $\mathbf{e}_1, \mathbf{e}'_1 \in \{0,1\}^u, \mathbf{e}_g, \mathbf{e}'_g \in \{0, g\}^u$ and $\mathrm{supp}(\mathbf{e}_1) = S = \mathrm{supp}(\mathbf{e}'_g)$ and $\mathrm{supp}(\mathbf{e}'_1) = S^C = \mathrm{supp}(\mathbf{e}_g)$. From this also follows that $\mathbf{e}_g = g \star \mathbf{e}'_1$ and $\mathbf{e}'_g = g \star \mathbf{e}_1$.

The first parity-check equation can now be reformulated as

$$\begin{aligned}&\mathbf{H}\mathbf{e}^\top - g \star \mathbf{H}\mathbf{e}'^\top\\ =&\mathbf{H}\mathbf{e}_1^\top - g \star \mathbf{H}\mathbf{e}_g'^\top + \mathbf{H}\mathbf{e}_g^\top - g \star \mathbf{H}\mathbf{e}_1'^\top\\ =&\mathbf{H}\mathbf{e}_1^\top - g^2 \star \mathbf{H}\mathbf{e}_1^\top + g \star \mathbf{H}\mathbf{e}_1'^\top - g \star \mathbf{H}\mathbf{e}_1'^\top\\ =&(1-g^2) \star \mathbf{H}\mathbf{e}_1^\top\\ =&(1-g^2,\ldots,1-g^2) = \mathbf{s}',\end{aligned}$$

thus, $\mathbf{H}\mathbf{e}_1^\top = (1,\ldots,1)$ is such that $\mathrm{supp}(\mathbf{e}_1)$ corresponds to a solution W of 3DM, as in the classical reduction.

2. Assume that the R-SDP solver returns "no", i.e., there exists no $\mathbf{e},\mathbf{e}' \in \mathbb{E}^u$ such that $(\mathbf{e},\mathbf{e}')\widetilde{\mathbf{H}}^\top = (\mathbf{s},\mathbf{s}')$. Let us assume by contradiction, that the 3DM has a solution W. We can then define S to be the indices of words in U belonging to the solution W. Let us define $\mathbf{e}_1,\mathbf{e}_1' \in \{0,1\}^u, \mathbf{e}_g,\mathbf{e}_g' \in \{0,g\}^u$ with $\mathrm{supp}(\mathbf{e}_1) = S = \mathrm{supp}(\mathbf{e}_g')$ and $\mathrm{supp}(\mathbf{e}_1') = S^C = \mathrm{supp}(\mathbf{e}_g)$. From this also follows that $\mathbf{e}_g = g \star \mathbf{e}_1'$ and $\mathbf{e}_g' = g \star \mathbf{e}_1$. Then the vector $(\mathbf{e}_1 + \mathbf{e}_g, \mathbf{e}_1' + \mathbf{e}_g') \in \mathbb{E}^{2u}$ is a solution to the R-SDP, as in case 1, which gives the desired contradiction, to the R-SDP solver returning "no".

Note that the hypothesis, that only 1 and g in $\mathbb{E}$ add up to $1+g$ is not necessary. For this assume that there exists $g^i, g^j \in \mathbb{E}$, with $0 \neq i < j < z$ such that $g^i + g^j = 1 + g$. Thus, the splitting of $\mathbf{e}$ and $\mathbf{e}'$ is a bit more complicated:

$$\begin{aligned}\mathbf{e} &= \mathbf{e}_1 + \mathbf{e}_g + \mathbf{e}_i + \mathbf{e}_j,\\ \mathbf{e}' &= \mathbf{e}_1' + \mathbf{e}_g' + \mathbf{e}_i' + \mathbf{e}_j',\end{aligned}$$

where $\mathbf{e}_1,\mathbf{e}_1' \in \{0,1\}^u$,$\mathbf{e}_g,\mathbf{e}_g' \in \{0,g\}^u$,$\mathbf{e}_i,\mathbf{e}_i' \in \{0,g^i\}^u$,$\mathbf{e}_j,\mathbf{e}_j' \in \{0,g^j\}^u$ with

$$\begin{aligned}\mathrm{supp}(\mathbf{e}_1) &= S_1 = \mathrm{supp}(\mathbf{e}_g'),\\ \mathrm{supp}(\mathbf{e}_g) &= S_1' = \mathrm{supp}(\mathbf{e}_1'),\\ \mathrm{supp}(\mathbf{e}_i) &= S_i = \mathrm{supp}(\mathbf{e}_j'),\\ \mathrm{supp}(\mathbf{e}_j) &= S_i' = \mathrm{supp}(\mathbf{e}_i'),\end{aligned}$$

and the supports S_1, S_1', S_i, S_i' are distinct and partition $\{1,\ldots,u\}$. Again it follows that

$$\begin{aligned}\mathbf{e}_g &= g \star \mathbf{e}_1',\\ \mathbf{e}_g' &= g \star \mathbf{e}_1,\\ \mathbf{e}_j &= g^{j-i} \star \mathbf{e}_i',\\ \mathbf{e}_j' &= g^{j-i} \star \mathbf{e}_i.\end{aligned}$$

Thus, rewriting the first parity-check equation, we get

$$\begin{aligned}&\mathbf{H}\mathbf{e}^\top - g \star \mathbf{H}\mathbf{e}'^\top\\ =&\mathbf{H}\mathbf{e}_1^\top + \mathbf{H}\mathbf{e}_g^\top + \mathbf{H}\mathbf{e}_i^\top + \mathbf{H}\mathbf{e}_j^\top\\ &- g \star \mathbf{H}\mathbf{e}_1'^\top - g \star \mathbf{H}\mathbf{e}_g'^\top - g \star \mathbf{H}\mathbf{e}_i'^\top - g \star \mathbf{H}\mathbf{e}_j'^\top\\ =&\mathbf{H}\mathbf{e}_1^\top + g \star \mathbf{H}\mathbf{e}_1'^\top + \mathbf{H}\mathbf{e}_i^\top + g^{j-i} \star \mathbf{H}\mathbf{e}_i'^\top\\ &- g \star \mathbf{H}\mathbf{e}_1'^\top - g^2 \star \mathbf{H}\mathbf{e}_1^\top - g \star \mathbf{H}\mathbf{e}_i'^\top - g^{j-i+1} \star \mathbf{H}\mathbf{e}_i^\top\\ =&(1-g^2) \star \mathbf{H}\mathbf{e}_1^\top + (1-g^{j-i+1}) \star \mathbf{H}\mathbf{e}_i^\top + (g^{j-i}-g) \star \mathbf{H}\mathbf{e}_i'^\top\\ =&(1-g^2,\ldots,1-g^2) = \mathbf{s}'.\end{aligned}$$

Since $\mathbf{e}_1, \mathbf{e}_i, \mathbf{e}_i'$ all have different supports, the only way to get $1-g^2$ in each entry, is to have $\mathbf{e}_i = \mathbf{e}_i' = 0$. In fact, any other sum leads to a contradiction:

- If $(1-g^2) + (1-g^{j-i+1}) = 1-g^2$ then $1 = g^{j-i+1}$ and hence $j = i-1$ which contradicts $j > i$.

- If $(1-g^2)+(g^{j-i}-g)=1-g^2$ then $g^{j-i}=g$ and hence $j-i=1$. However, as then $g^j+g^i=g^i(1+g)=1+g$, it follows that $g^i=1$, which contradicts $i\neq 0$.
- If $(1-g^2)+(1-g^{j-i+})+(g^{j-i}-g)=1-g^2$, then $1+g^{j-i}=g^{j-i+1}+g=g(1+g^{j-i})$ and thus $g=1$, which contradicts $\mathbb{E}\neq\mathbb{F}_q^\star$.
- If $(1-g^{j-i+1})+(g^{j-i}-g)=1-g^2$, then $g^{j-i}-g^{j-i+1}=g-g^2$ and hence $g^{j-i}(1-g)=g(1-g)$ and thus $j-i=1$, which is a contradiction again as in the second case.

□

Note that we choose z as prime to work with the easy arithmetic over the finite field $\mathbb{F}_z$. It is well-known that for $z \mid p-1$, there are $\varphi(z)$ many elements of order z in $\mathbb{F}_p$. Since we chose z to be a prime, we get $z-1$ such elements, which are all in $\mathbb{E}$. That is, for any element g of order z, and any element $a\in\mathbb{E}=\{g^i \mid i\in\{1,\dots,z\}\}$ with $a\neq 1$, the order of a is also z, and we could have equivalently picked a to generate $\mathbb{E}$.

Uniqueness of the solution to R-SDP We always consider that an R-SDP instance has been chosen uniformly at random. That is, both the parity-check matrix $\mathbf{H}$ and $\mathbf{e}$ have been generated by sampling from $\mathbb{F}_p^{(n-k)\times n}$ and $\mathbb{E}^n$ uniformly at random. Consequently, the anticipated number of solutions we expect to have, on average, is

$$\frac{z^n}{p^{n-k}}=2^{n\left(\log_2(z)-(1-R)\log_2(p)\right)}.$$

When z and R are such that $\log_2(z)\leq(1-R)\log_2(p)$, we expect to have at most one solution.

Restricted vectors Note that $(\mathbb{E}^n,\star)$ is a commutative, transitive group isomorphic to $(\mathbb{F}_z^n,+)$. The isomorphism is given by

$$\ell:\mathbb{E}^n\to\mathbb{F}_z^n,$$
$$\mathbf{x}=(g^{\ell_1},\dots,g^{\ell_n})\mapsto\ell(\mathbf{x})=(\ell_1,\dots,\ell_n).$$

This representation of vectors in $\mathbb{E}^n$ as vectors in $\mathbb{F}_z^n$ is helpful to shorten the sizes of objects. For the opposite direction of the isomorphism, we use the following abuse of notation

$$\mathbf{a}=g^{\ell(\mathbf{a})}=(g^{\ell(\mathbf{a})_1},\dots,g^{\ell(\mathbf{a})_n}),$$

for some $\ell(\mathbf{a})=(\ell(\mathbf{a})_1,\dots,\ell(\mathbf{a})_n)\in\mathbb{F}_z^n$.

Since any restricted vector $\mathbf{a}\in\mathbb{E}^n$ can be compactly represented as a vector $\ell(\mathbf{a})\in\mathbb{F}_z^n$ we only require $n\log_2(z)$ bits to represent a restricted vector.

Restricted transformations A linear map $\varphi:\mathbb{E}^n\to\mathbb{E}^n$ which acts transitively on $\mathbb{E}^n$ is simply given by component-wise multiplication, i.e., $\varphi(\mathbf{b})=\mathbf{a}\star\mathbf{b}$, for some $\mathbf{a}\in\mathbb{E}^n$. In fact, for any $\mathbf{a}=g^{\ell(\mathbf{a})}$ and any $\mathbf{b}=g^{\ell(\mathbf{b})}$ we can always find $\mathbf{c}=g^{\ell(\mathbf{a})-\ell(\mathbf{b})}$ such that $\mathbf{a}=\mathbf{c}\star\mathbf{b}$.

Let the map φ be the component-wise multiplication with $\mathbf{a}\in\mathbb{E}^n$. Then we can compactly represent φ through the vector $\ell(\mathbf{a})\in\mathbb{F}_z^n$. Additionally, the computation $\varphi(\mathbf{b})=\mathbf{a}\star\mathbf{b}$ is given by an addition in $\mathbb{F}_z^n$; namely $\ell(\mathbf{a})+\ell(\mathbf{b})$. Thus, we only require $n\log_2(z)$ bits to represent restricted transformations.

3.2.3 Restricted Decoding Problem in a Subgroup

We can also further generalize this problem by considering a subgroup $(G,\star)\leq(\mathbb{E}^n,\star)$ as

$$G=\langle\mathbf{a}_1,\dots,\mathbf{a}_m\rangle=\left\{\prod_{i=1}^m\mathbf{a}_i^{u_i}\mid u_i\in\{1,\dots z\}\right\},$$

for some $m<n$. Thus, we can update the R-SDP to the R-SDP(G):

Problem 4. ***Restricted Syndrome Decoding Problem with subgroup*** G ***(R-SDP***(G)***)***
Let $G = \langle \mathbf{a}_1, \ldots, \mathbf{a}_m \rangle$, for $\mathbf{a}_i \in \mathbb{E}^n$, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, and $\mathbf{s} \in \mathbb{F}_p^{n-k}$. Does there exist a vector $\mathbf{e} \in G$ with $\mathbf{e}\mathbf{H}^\top = \mathbf{s}$?

The R-SDP(G) is still an NP-hard problem since the R-SDP is NP-hard. That is, if there exists a polynomial time solver that can solve the problem for any G, then this solver could also solve $G = \mathbb{E}^n$. Notice that R-SDP(G) admits fewer solutions than the more general R-SDP. Consequently, we can modify the criterion to have a unique solution as

$$|G| p^{-(1-R)n} \leq 1,$$

and setting $|G| = z^m$ we obtain $m \log_p(z) \leq (1-R)n$.

Restricted vectors from the subgroup To construct elements $\mathbf{e} \in G$, we can collect all the exponents of the generators $\mathbf{a}_i$ into a matrix. That is, we define the matrix $\mathbf{M}_G \in \mathbb{F}_z^{m\times n}$ as

$$\mathbf{M}_G = \begin{pmatrix} \ell(\mathbf{a}_1)_1 & \cdots & \ell(\mathbf{a}_1)_n \\ \vdots & & \vdots \\ \ell(\mathbf{a}_m)_1 & \cdots & \ell(\mathbf{a}_m)_n \end{pmatrix} = \begin{pmatrix} \ell(\mathbf{a}_1) \\ \vdots \\ \ell(\mathbf{a}_m) \end{pmatrix}.$$

To check whether $|G| = z^m$, it is enough to verify $\mathrm{rank}(\mathbf{M}_G) = m$. In order to construct an element $\mathbf{e} \in G$, we can then simply pick any vector $\mathbf{u} \in \mathbb{F}_z^m$ and compute $\ell(\mathbf{e}) = \mathbf{u}\mathbf{M}_G \in \mathbb{F}_z^n$, since then $\mathbf{e} = g^{\ell(\mathbf{e})}$.

To this end, we introduce the group homomorphism ℓ_G:

$$\ell_G : G \to \mathbb{F}_z^m,$$
$$\mathbf{e} = \mathbf{a}_1^{u_1} \star \cdots \star \mathbf{a}_m^{u_m} \mapsto \ell_G(\mathbf{e}) = (u_1, \ldots, u_m).$$

Thus, to sample $\mathbf{e} \in G$, it is enough to choose $\ell_G(\mathbf{e}) \in \mathbb{F}_z^m$. To represent restricted vectors in the subgroup, we require only $m \log_2(z)$ bits.

Restricted transformations in the subgroup The linear maps $\varphi : G \to G$, which act transitively on G, are still given by component-wise multiplication with another element in G, i.e., for $\mathbf{e} \in G$, $\varphi(\mathbf{e}) = \mathbf{e}' \star \mathbf{e}$. Thus, we can again represent the map by a vector in $\mathbb{F}_z^m$ and the operation on $\mathbf{e} \in G$ as an addition in $\mathbb{F}_z^m$. To represent restricted transformations in the subgroup, we again require just $m \log_2(z)$ bits.

4 A Comprehensive Overview of CROSS

In this chapter, we succinctly describe the CROSS signature scheme. We first introduce the underlying ZK protocol, CROSS-ID, and show that it is a $q2$-Identification protocol with a $q2$-Extractor. Our protocol is essentially derived from the CVE ZK protocol [26] originally proposed for SDP. We adapt such a scheme to the R-SDP setting and apply some standard optimizations to reduce communication costs. We prove that our protocol achieves zero-knowledge, completeness, and soundness, with soundness error $\varepsilon = \frac{p}{2(p-1)}$.

This implies that the Fiat-Shamir transformation of parallel executions of this protocol yields a signature scheme that is EUF-CMA secure.

We then proceed by applying other optimizations that are effective only when parallel executions are considered, e.g., using Merkle and PRNG trees and an unbalanced distribution for the second challenge, aiming to reduce the communication cost and, consequently, the signature size.

We describe the protocol in terms of R-SDP(G), which also encompasses the R-SDP version by simply setting $G = \mathbb{E}^n$, which also implies $m = n$.

4.1 CROSS ZK protocol

The ZK protocol employed in CROSS is described in Figure 2. What sets this scheme aside from CVE is that in our protocol, the prover first samples a transformed error vector $\mathbf{e}' \in G$ and a random $\mathbf{u}' \in \mathbb{F}_p^n$, and only then do they find the transformation $\sigma \in G$ such that $\mathbf{e} = \sigma(\mathbf{e}')$. Notice that $\sigma : G \mapsto G$ is a bijection, so that σ is uniformly random over G provided that $\mathbf{e}'$ is sampled uniformly at random from G. Since $\mathbf{u}'$ is uniformly random over $\mathbb{F}_p^n$, $\mathbf{y} = \mathbf{u}' + \beta\mathbf{e}'$ follows the uniform distribution over $\mathbb{F}_p^n$. Here, $\beta \in \mathbb{F}_p^*$ is the first challenge. Another difference with respect to CVE is that the first response is $h = \mathsf{Hash}(\mathbf{y})$. The second challenge, which we indicate by b, is either 0 or 1. When $b = 1$, one can communicate the seed which was used to sample both $\mathbf{u}'$ and $\mathbf{e}'$: this shows that indeed $\mathbf{y}$ has been generated as the sum of a masking vector and a restricted vector which has been scaled by β. This strategy saves us some communication cost since sending h requires fewer bits than $\mathbf{y}$. When $b = 0$, the prover reveals $\mathbf{y}$, together with σ (which is not a random transformation, hence it cannot be compressed using seeds).

Private Key $\mathbf{e} \in G$
Public Key $G \subseteq \mathbb{E}^n$, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $\mathbf{s} = \mathbf{e}\mathbf{H}^\top \in \mathbb{F}_p^{n-k}$

PROVER		VERIFIER
Sample $\mathsf{Seed} \xleftarrow{\$} \{0;1\}^\lambda$		
Sample $(\mathbf{e}', \mathbf{u}') \xleftarrow{\mathsf{Seed}} G \times \mathbb{F}_p^n$		
Compute $\sigma \in G$ such that $\sigma(\mathbf{e}') = \mathbf{e}$		
Set $\mathbf{u} = \sigma(\mathbf{u}')$		
Compute $\widetilde{\mathbf{s}} = \mathbf{u}\mathbf{H}^\top$		
Set $c_0 = \mathsf{Hash}(\widetilde{\mathbf{s}}, \sigma)$		
Set $c_1 = \mathsf{Hash}(\mathbf{u}', \mathbf{e}')$		
	$\xrightarrow{(c_0,c_1)}$	
	$\xleftarrow{\beta}$	Sample $\beta \xleftarrow{\$} \mathbb{F}_p^*$
Compute $\mathbf{y} = \mathbf{u}' + \beta\mathbf{e}'$		
Set $h = \mathsf{Hash}(\mathbf{y})$		
	$\xrightarrow{h}$	
		Sample $b \xleftarrow{\$} \{0,1\}$
	$\xleftarrow{b}$	
If $b = 0$, set $f := (\mathbf{y}, \sigma)$		
If $b = 1$, set $f := \mathsf{Seed}$	$\xrightarrow{f}$	
		If $b = 0$:
		Compute $\widetilde{\mathbf{y}} = \sigma(\mathbf{y})$ and $\widetilde{\mathbf{s}} = \widetilde{\mathbf{y}}\mathbf{H}^\top - \beta\mathbf{s}$
		Accept if:
		1) $\mathsf{Hash}(\mathbf{y}) = h$
		2) $\mathsf{Hash}(\widetilde{\mathbf{s}}, \sigma) = c_0$
		3) $\sigma \in G$
		If $b = 1$:
		Sample $(\mathbf{e}', \mathbf{u}') \xleftarrow{\mathsf{Seed}} G \times \mathbb{F}_p^n$
		Set $\mathbf{y} = \mathbf{u}' + \beta\mathbf{e}'$
		Accept if:
		1) $\mathsf{Hash}(\mathbf{y}) = h$
		2) $\mathsf{Hash}(\mathbf{u}', \mathbf{e}') = c_1$

Figure 2: CROSS-ID

We briefly comment on how some of the operations in the protocol can be efficiently implemented:

- The group G is represented by a basis $\mathbf{M}_G \in \mathbb{F}_z^{m\times n}$. This can be conveniently provided in the form $\mathbf{M}_G = (\mathbf{I}_m, \mathbf{W})$, with $\mathbf{W} \in \mathbb{F}_z^{m\times(n-m)}$. To sample uniformly at random from G, it is enough to

  1. sample $\mathbf{a} \xleftarrow{\$} \mathbb{F}_z^m$,
  2. obtain the exponents $\mathbf{x} = \mathbf{a}\mathbf{M}_G \in \mathbb{F}_z^n$,

3. obtain the restricted vector/isometry as $\left(g^{x_1}, \ldots, g^{x_n}\right) \in \mathbb{F}_p^n$.

Notice that $\mathbf{a}$ is the ℓ_G representation of the restricted object we have sampled. Using $\mathbf{M}_G$ in systematic form, we have a computational advantage since $\mathbf{x} = (\mathbf{a}, \mathbf{aW})$ and computing $\mathbf{aW}$ requires only $O\big(m(n-m)\big)$ operations over $\mathbb{F}_z$.

- The ℓ_G representation for σ can be easily computed as $\ell_G(\sigma) = (a_1, \ldots, a_m) = \ell_G(\mathbf{e}) - \ell_G(\mathbf{e}')$. The verifier will simply verify that $\ell_G(\sigma) \in \mathbb{F}_z^m$ and will locally regenerate σ by first computing the coefficients $\ell_G(\sigma)\mathbf{M}_G \in \mathbb{F}_z^n$, then using them as the exponents for g.
- The matrix $\mathbf{W}$, as well as the parity-check matrix $\mathbf{H}$, can be sampled from a unique seed $\mathsf{Seed}_{\mathsf{pk}} \in \{0;1\}^{2\lambda}$ which is included in the public key. This way, the public key is $\{\mathbf{s}, \mathsf{Seed}_{\mathsf{pk}}\}$ and has size $(n-k)\log_2(p) + 2\lambda$. We will use $\mathsf{Seed}_{\mathsf{pk}}$ with 2λ bits.
- When relying on R-SDP, there is no need to use G. The seed $\mathsf{Seed}_{\mathsf{pk}}$ is used to sample only $\mathbf{H}$. The exponents of σ can be simply obtained as $\ell(\mathbf{e}) - \ell(\mathbf{e}')$.

We now proceed to show that the protocol in Figure 2 achieves zero-knowledge, completeness, and $(2,2)$-out-of-$(p-1,2)$ special soundness. Consequently, it is a $q2$-Identification protocol where $q = p-1$ and has soundness error $\frac{p}{2(p-1)}$.

Proposition 5 (Completeness)**.** The protocol in Figure 2 is complete.

Proof. We have to show that the honest prover always gets accepted. When $b = 0$, we have

$$\widetilde{\mathbf{y}} = \sigma(\mathbf{y}) = \sigma(\mathbf{u}') + \beta\sigma(\mathbf{e}') = \mathbf{u} + \beta\mathbf{e}.$$

So, it holds that

$$\widetilde{\mathbf{y}}\mathbf{H}^\top - \beta\mathbf{s} = \mathbf{u}\mathbf{H}^\top + \beta\mathbf{e}\mathbf{H}^\top - \beta\mathbf{s} = \widetilde{\mathbf{s}} + \beta\mathbf{s} - \beta\mathbf{s} = \mathbf{u}\mathbf{H}^\top.$$

This indeed corresponds to the syndrome that was used to generate c_0. Finally, we also verify that $\sigma \in G$. When $b = 1$, the prover provides only seeds: since PRNGs are deterministic, the verifier obtains the very same quantities that have been used to generate the commitments. □

Proposition 6 (Zero-Knowledge)**.** The protocol in Figure 2 achieves zero-knowledge.

Proof. We prove that a simulator $\mathcal{S}$ with knowledge of the challenges can easily simulate the interaction $\langle \mathcal{P}, \mathcal{V} \rangle$ between the prover and the verifier. Formally, we show that $\mathcal{S}$ produces a communication tape T^* that is indistinguishable from the T resulting from the interaction between $\langle \mathcal{P}, \mathcal{V} \rangle$. We define two strategies that $\mathcal{S}$ can follow, which will depend uniquely on the value of b (i.e., the second challenge).

- *Strategy for* $b = 0$: The simulator $\mathcal{S}$ finds, with simple linear algebra, a vector $\mathbf{e}^* \in \mathbb{F}_p^n$ with syndrome $\mathbf{s}$, i.e., such that $\mathbf{e}^*\mathbf{H}^\top = \mathbf{s}$. Then, $\mathcal{S}$ selects a random $\sigma^* \in G$ and a vector $\mathbf{u}^* \in \mathbb{F}_p^n$, and computes $\mathbf{u}'^* = \sigma^{*-1}(\mathbf{e}^*)$. Finally, it computes $\widetilde{\mathbf{s}}^* = \mathbf{u}^*\mathbf{H}^\top$ and $c_0 = \mathsf{Hash}(\widetilde{\mathbf{s}}^*, \sigma^*)$. Then, $\mathcal{S}$ computes $\mathbf{y}^* = \mathbf{u}'^* + \beta\mathbf{e}'^*$. It is easy to see that the transcript produced by $\mathcal{S}$ (i.e., the values $\mathbf{y}^*$ and σ^*) follow the same statistical distribution as those of an honestly produced transcript. Indeed, in an honest execution, $\mathbf{y}$ is uniformly random over $\mathbb{F}_p^*$ because $\mathbf{u}'$ is uniformly random over $\mathbb{F}_p^n$. This guarantees that $\mathbf{u}' + \beta\mathbf{e}'$ is uniformly random over $\mathbb{F}_p^n$, and the same holds after applying the transformation σ. Finally, in an honest execution of the protocol, σ is uniformly distributed over G. Indeed, for any $\mathbf{e}' \in G$ there is a unique $\sigma \in G$ such that $\sigma(\mathbf{e}') = \mathbf{e}$. If $\mathbf{e}'$ is uniformly random over G, then σ also follows the same distribution. The commitment which is not verified can be chosen as a random binary string with length 2λ. Under the ROM, this has the same statistical distribution as an honestly computed c_1.
- *Strategy for* $b = 1$: In this case, the simulator simply executes the protocol by sampling the seed and computing c_1 analogously to what the honest prover $\mathcal{P}$ would do. For the other commitment, c_0, it is enough to use a random binary string again.

□

Proposition 7 (Soundness)**.** The protocol in Figure 2 is sound, with soundness error $\varepsilon = \frac{p}{2(p-1)}$.

Proof. We consider an adversary $\mathcal{A}$ that tries to impersonate the prover; that is, they aim to reply correctly to the verifier's challenges. We first sketch two cheating strategies that achieve a success probability of $\varepsilon = \frac{p}{2(p-1)}$, and then we show that these strategies are optimal in the sense that the success probability is maximum and corresponds to the soundness error. We will do this by proving that the protocol is $(2,2)$-out-of-$(p-1,2)$ special sound: from this, the formula for the soundness error follows [4, 5]. Notice that the cheating strategies are not necessary for the proof but are provided for completeness.

Strategy 0: The adversary $\mathcal{A}$ aims to always answer correctly in case $b = 0$, but also guesses β^* in an attempt to be accepted in the case $b = 1$.

For this, the adversary first chooses $\beta^* \in \mathbb{F}_p^*$ and a seed $\mathtt{Seed}$ which is used to sample $\mathbf{u}' \in \mathbb{F}_p^n$ and $\mathbf{e}' \in G$. The adversary also chooses a random $\sigma \in G$ and computes $\mathbf{y}^* = \mathbf{u}' + \beta^* \mathbf{e}'$. The commitment c_1 is prepared as $c_1 = \mathsf{Hash}(\mathbf{u}', \mathbf{e}')$. Now, the adversary computes $\widetilde{\mathbf{s}} = \sigma(\mathbf{y}^*)\mathbf{H}^\top$ and sets $c_0 = \mathsf{Hash}(\widetilde{\mathbf{s}} - \beta^* \mathbf{s}, \sigma)$. Finally, the adversary chooses a vector $\widetilde{\mathbf{e}} \in \mathbb{F}_p^n$ such that $\widetilde{\mathbf{e}}\mathbf{H}^\top = \mathbf{s}$ and a vector $\widetilde{\mathbf{u}} \in \mathbb{F}_p^n$ such that $\widetilde{\mathbf{u}}\mathbf{H}^\top = \sigma(\mathbf{y}^*)\mathbf{H}^\top - \beta^* \mathbf{s}$. The adversary sends c_0, c_1 to the verifier and receives β.

If $\beta = \beta^*$, then the adversary replies with $h = \mathsf{Hash}(\mathbf{y}^*)$. If the verifier asks for $b = 0$, the adversary sends $(\mathbf{y}^*, \sigma)$ and gets accepted since $\mathsf{Hash}(\mathbf{y}^*) = h$, $\sigma \in G$, and

$$c_0 = \mathsf{Hash}(\sigma(\mathbf{y}^*)\mathbf{H}^\top - \beta \mathbf{s}, \sigma) = \mathsf{Hash}(\widetilde{\mathbf{s}} - \beta^* \mathbf{s}, \sigma).$$

If $b = 1$, the adversary replies with the seed to compute $\mathbf{e}'$ and $\mathbf{u}'$. Also in this case, the adversary gets accepted as $h = \mathsf{Hash}(\mathbf{u}' + \beta^* \mathbf{e}')$ and $c_1 = \mathsf{Hash}(\mathbf{u}', \mathbf{e}')$.

If, however, $\beta \neq \beta^*$, the adversary sends a different h. Namely, the adversary computes $\mathbf{y} = \sigma^{-1}(\widetilde{\mathbf{u}}) + \beta\sigma^{-1}(\widetilde{\mathbf{e}})$ and sends $h = \mathsf{Hash}(\mathbf{y})$. If the verifier then asks for $b = 0$, the adversary sends $(\mathbf{y}, \sigma)$ and gets accepted as $h = \mathsf{Hash}(\mathbf{y})$, $\sigma \in G$, and

$$c_0 = \mathsf{Hash}(\sigma(\mathbf{y})\mathbf{H}^\top - \beta \mathbf{s}, \sigma) = \mathsf{Hash}(\widetilde{\mathbf{u}}\mathbf{H}^\top, \sigma) = \mathsf{Hash}(\sigma(\mathbf{y}^*)\mathbf{H}^\top - \beta^* \mathbf{s}, \sigma).$$

If the verifier asks for $b = 1$, the adversary has no chance of success.

Consequently, this strategy has a success probability given by

$$\Pr[b = 0] + \Pr[(b = 1) \wedge (\beta = \beta^*)] = \frac{1}{2} + \frac{1}{2(p-1)} = \frac{p}{2(p-1)}.$$

Strategy 1: The adversary hopes to receive the challenge $b = 1$ but, again, prepares to reply also for $b = 0$ by guessing the value of β. The adversary starts by choosing a $\beta^* \in \mathbb{F}_p^*$. The adversary selects a seed from which they generate $\mathbf{u}' \in \mathbb{F}_p^n$ and $\mathbf{e}' \in G$. The adversary also picks a $\sigma \in G$ and computes $\mathbf{u} = \sigma(\mathbf{u}'), \widetilde{\mathbf{e}} = \sigma(\mathbf{e}') \in G$. The adversary calculates $\widetilde{\mathbf{s}} = \mathbf{u}\mathbf{H}^\top + \beta^* \widetilde{\mathbf{e}}\mathbf{H}^\top - \beta^* \mathbf{s}$. The adversary will send the commitments $c_0 = \mathsf{Hash}(\widetilde{\mathbf{s}}, \sigma)$ and $c_1 = \mathsf{Hash}(\mathbf{u}', \mathbf{e}')$. When the adversary receives $\beta \in \mathbb{F}_q^\star$, the adversary still computes $\mathbf{y} = \mathbf{u}' + \mathbf{e}'\beta$ and sends the Hash h.

If the adversary gets challenged with $b = 1$, they send the seeds of $\mathbf{u}'$ and $\mathbf{e}'$, and will be accepted with probability one. This follows as the verifier uses the seeds to reconstruct $\mathbf{u}'$ and $\mathbf{e}'$, and they can check that $h = \mathsf{Hash}(\mathbf{u}' + \beta \mathbf{e}')$ and $c_1 = \mathsf{Hash}(\mathbf{u}', \mathbf{e}')$.

However, if the adversary is given the challenge $b = 0$, they send $(\mathbf{y}, \sigma)$ and can only get accepted if $\beta = \beta^*$, since then

$$\sigma(\mathbf{y})\mathbf{H}^\top - \beta \mathbf{s} = \mathbf{u}\mathbf{H}^\top + \beta \tilde{\mathbf{e}}\mathbf{H}^\top - \beta \mathbf{s} = \tilde{\mathbf{s}}.$$

Hence, in this case, $c_0 = \mathsf{Hash}(\sigma(\mathbf{y})\mathbf{H}^\top - \beta \mathbf{s}, \sigma)$ and $h = \mathsf{Hash}(\mathbf{y})$, with $\sigma \in G$. Thus, this strategy has a success probability given by

$$\Pr[b = 1] + \Pr[(b = 0) \wedge (\beta = \beta^*)] = \frac{1}{2} + \frac{1}{2(p-1)} = \frac{p}{2(p-1)}.$$

$(2,2)$*-out-of-*$(p-1,2)$ *special soundness* We consider four accepting transcripts T_1, T_2, T_3, T_4, all associated with the same pair of commitments c_0, c_1. The commitment c_0 fixes the pair $(\widetilde{\mathbf{s}}, \sigma)$, while the commitment c_1 fixes the pair $(\mathbf{u}', \mathbf{e}')$. We identify the transcripts by the challenge values, which we denote respectively by $(\beta, 0)$, $(\beta, 1)$, $(\beta^*, 0)$, and $(\beta^*, 1)$. Taking into account the prover's replies, we have that the transcripts are structured as follows:

T_1: $\big(c_0, c_1, \beta, h, \mathbf{y}, \sigma\big)$;

T_2: $\big(c_0, c_1, \beta, h, \mathtt{Seed}\big)$;

T_3: $\big(c_0, c_1, \beta^*, h^*, \mathbf{y}^*, \sigma^*\big)$;

T_4: $\big(c_0, c_1, \beta^*, h^*, \mathtt{Seed}^*\big)$.

We now show that, from the knowledge of these four transcripts, a solution for the R-SDP(G) instance $\{\mathbf{s}, \mathbf{H}\}$ can be easily computed (i.e., in polynomial time). We first focus on T_2 and T_4. Let $\mathbf{u}', \mathbf{e}'$ be the vectors generated from $\mathtt{Seed}$, and $\mathbf{u}'^*, \mathbf{e}'^*$ those generated from $\mathtt{Seed}^*$. Since c_1 is verified in both cases, either hash collisions have been found (i.e., $\mathsf{Hash}\big(\mathbf{u}', \mathbf{e}'\big) = \mathsf{Hash}\big(\mathbf{u}'^*, \mathbf{e}'^*\big)$ but $\mathbf{u}' \neq \mathbf{u}'^*$ and/or $\mathbf{e}' \neq \mathbf{e}'^*$), or $\mathbf{u}' = \mathbf{u}'^*$ and $\mathbf{e}' = \mathbf{e}'^*$. Since also h and h^* are checked, and unless hash collisions have been found, this guarantees that $h = \mathsf{Hash}(\mathbf{y})$, where $\mathbf{y} = \mathbf{u}' + \beta\mathbf{e}'$, and $h^* = \mathsf{Hash}(\mathbf{y}^*)$, where $\mathbf{y}^* = \mathbf{u}'^* + \beta^*\mathbf{e}'^* = \mathbf{u}' + \beta^*\mathbf{e}'$. This implies that $\mathbf{y} - \mathbf{y}^* = \mathbf{e}'(\beta - \beta^*)$. Now, we look at the pair of transcripts T_1 and T_3. Unless hash collisions have been found, we have $\sigma = \sigma^*$,

$$\sigma(\mathbf{y})\mathbf{H}^\top - \beta\mathbf{s} = \widetilde{\mathbf{s}}, \text{ and } \sigma(\mathbf{y}^*)\mathbf{H}^\top - \beta^*\mathbf{s} = \widetilde{\mathbf{s}},$$

from which it follows that

$$\sigma(\mathbf{y} - \mathbf{y}^*)\mathbf{H}^\top = (\beta - \beta^*)\mathbf{s}.$$

Exploiting the relations we derived from the pair (T_2, T_4), we obtain $\mathbf{y} - \mathbf{y}^* = (\beta - \beta^*)\mathbf{e}'$, where $\mathbf{e}'$ is a restricted vector, hence

$$(\beta - \beta^*)\sigma(\mathbf{e}')\mathbf{H}^\top = (\beta - \beta^*)\mathbf{s} \implies \sigma(\mathbf{e}')\mathbf{H}^\top = \mathbf{s}.$$

Since $\sigma, \mathbf{e}'$ have been verified, thus $\sigma, \mathbf{e}' \in G$; then $\sigma(\mathbf{e}') \in G$ means that $\sigma(\mathbf{e}')$ solves R-SDP(G) for the instance $\{\mathbf{H}, \mathbf{s}\}$. □

4.2 Fiat-Shamir Transformation with fixed weight challenges

Since the protocol in Figure 2 can be classified as a $q2$-Identification protocol as in [41], when considering t parallel executions and applying the Fiat-Shamir transformation, we obtain a scheme that achieves EUF-CMA security.

Theorem 8. CROSS, the signature scheme resulting after applying the Fiat-Shamir transform on t parallel executions of a $q2$-Identification protocol, achieves EUF-CMA security.

This follows from the fact that CROSS applies the Fiat-Shamir transform on t parallel executions of a $q2$-Identification protocol and by the arguments from [41].

All the messages that are exchanged in the i-th round are indicated with the apix $^{(i)}$, e.g., $c_0^{(i)}$ and $c_1^{(i)}$ are the commitments while $h^{(i)}$ is the hash of the vector $\mathbf{y}^{(i)}$, computed as $\mathbf{u}'^{(i)} + \beta^{(i)}\mathbf{e}'^{(i)}$. To be consistent with the notation used in Section 3.1, we group all the messages exchanged during the t rounds as follows.

	Round 1		**Round 2**		$\cdots$	**Round t**	
Commitment = (	$c_0^{(1)}$	$c_1^{(1)}$	$c_0^{(2)}$	$c_1^{(2)}$	$\cdots$	$c_0^{(t)}$	$c_1^{(t)}$)
First challenge = (	$\beta^{(1)}$		$\beta^{(2)}$		$\cdots$	$\beta^{(t)}$	)
First response = (	$h^{(1)}$		$h^{(2)}$		$\cdots$	$h^{(t)}$	)
Second challenge = (	$b^{(1)}$		$b^{(2)}$		$\cdots$	$b^{(t)}$	)
Second response = (	$f^{(1)}$		$f^{(2)}$		$\cdots$	$f^{(t)}$	)

We use the Fiat-Shamir transformation as described in Section 3.1. To prevent the use of attacks based on commitments collisions, we make use of a length-2λ salt $\mathtt{Salt}$, as suggested in [27]. A graphical representation of how challenges are generated is shown in Figure 3.

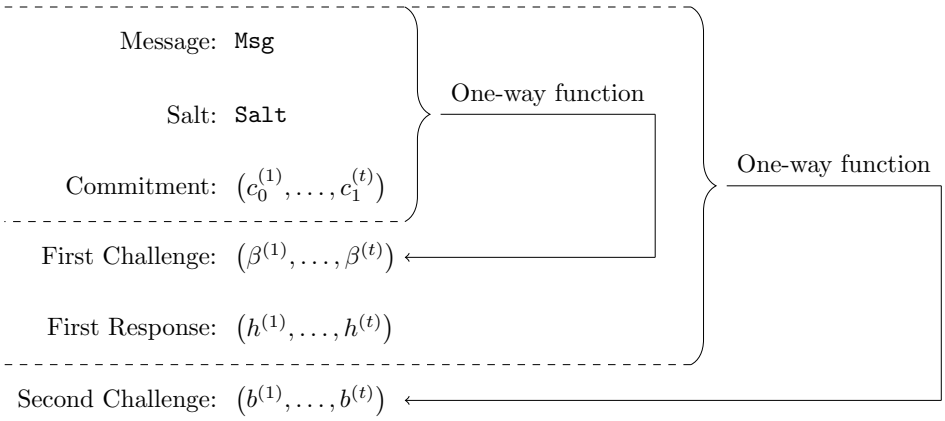

Figure 3: Flowchart representation of challenges generation in the Fiat-Shamir transformation

4.2.1 Fixed Weight Second Challenge

When applying the Fiat-Shamir transformation, we consider that the second challenge $(b^{(1)}, \ldots, b^{(t)})$ has always weight w, i.e., there are exactly w rounds where the verifier asks for $b = 1$, and $t - w$ rounds where $b = 0$. In the case $b = 1$ the prover can just send a length-λ seed and does not need to reveal $\mathbf{y}$ (since it can be recomputed by the verifier). Since the signature size depends (also) on how many rounds use $b = 0$ and how many rounds use $b = 1$, requiring these values to be constant reduces variability in the signature size and, moreover, simplifies constant-time implementations.

Fixing the value of w comes with some consequences:

- When w is close to t, the majority of rounds require only a very small communication cost. In such a case, specific optimizations to further reduce the signature size can be applied. We discuss them in the remainder of this section;
- attackers may exploit the knowledge that the second challenge has constant weight w. In Section 7.3.1, we take this into account and analyze how it affects the cost of forgery attacks.

Depending on the ratio between w and t, we distinguish between three different versions for the CROSS signature scheme. In the first version, which is the one we employ in CROSS-fast variant, we have $w \approx t/2$ and t slightly larger than λ. In the second and third versions, which we employ in CROSS-balanced and CROSS-small, we instead have that w is close to t and that t is much greater than λ.

Both versions use some well-known optimizations to reduce the signature size. Some of them apply to any setting of the protocol (i.e., any pair of values t and w), while others are meaningful only when w is close to t. We begin by consider optimizations of the first type, which are employed in CROSS-fast.

We then focus on the case in which w is close to t and consequently consider optimizations of the latter type; the corresponding signature scheme is employed for CROSS-balanced and CROSS-small.

Notice that all the techniques we use are standard, are employed in several modern signature schemes, and are essentially transparent from a security point of view. The only modification that may warrant some discussion is requiring a fixed weight for the second challenge since (as previously mentioned) this affects the cost of forgeries. This latter version will, consequently, rely on some further optimizations.

We proceed by first describing the CROSS-fast signature scheme, which we derive by applying some optimizations to the straightforward Fiat-Shamir transformation of (t parallel repetitions of) the protocol in Figure 2. Then, in the subsequent section, we show the scheme employed in CROSS-balanced and CROSS-small, in which we consider two further optimizations, namely, the use of a Merkle tree for the commitments $c_0^{(1)}, \ldots, c_0^{(t)}$ and a seed tree for the seeds $\mathtt{Seed}^{(1)}, \ldots, \mathtt{Seed}^{(t)}$.

Let us denote by $c_0 = \mathsf{Hash}(c_0^{(1)}, \ldots, c_0^{(t)})$ and by $c_1 = \mathsf{Hash}(c_1^{(1)}, \ldots, c_1^{(t)})$. Then, we denote by $\mathtt{GenCh}_1$ the one-way function, which takes as inputs $\mathtt{Msg}, \mathtt{Salt}$, and the commitments (c_0, c_1), and outputs the first challenge vector $(\beta^{(1)}, \ldots, \beta^{(t)}) \in (\mathbb{F}_p^\star)^t$. Similarly, we denote by $\mathtt{GenCh}_2$ the one-way function, which takes as inputs $\mathtt{Msg}, \mathtt{Salt}$, the first challenge vector $(\beta^{(1)}, \ldots, \beta^{(t)}, h$ and the commitments (c_0, c_1), and outputs the second challenge vector $(b^{(1)}, \ldots, b^{(t)}) \in \{0, 1\}^t$ of fixed weight w.

4.3 CROSS-fast

The description of the algorithms for signature generation and verification are given in Figures 4 and 5.

Private Key $\mathbf{e} \in G$
Public Key $G \subseteq \mathbb{E}^n$, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $\mathbf{s} = \mathbf{e}\mathbf{H}^\top \in \mathbb{F}_p^{n-k}$
Input Message $\mathtt{Msg}$
Output Signature $\mathtt{Signature}$

SIGNER	VERIFIER
Sample $\mathtt{Salt} \xleftarrow{\$} \{0;1\}^{2\lambda}$	
For $i = 1, \ldots, t$:	
Sample $\mathtt{Seed}^{(i)} \xleftarrow{\$} \{0;1\}^{\lambda}$	
Sample $(\mathbf{e}'^{(i)}, \mathbf{u}'^{(i)}) \xleftarrow{\mathtt{Seed}^{(i)}} G \times \mathbb{F}_p^n$	
Compute $\sigma^{(i)} \in G$ such that $\sigma^{(i)}(\mathbf{e}'^{(i)}) = \mathbf{e}$	
Set $\mathbf{u}^{(i)} = \sigma^{(i)}(\mathbf{u}'^{(i)})$	
Compute $\widetilde{\mathbf{s}}^{(i)} = \mathbf{u}^{(i)}\mathbf{H}^\top$	
Set $c_0^{(i)} = \mathsf{Hash}\big(\widetilde{\mathbf{s}}^{(i)}, \sigma^{(i)}, \mathtt{Salt}, i\big)$	
Set $c_1^{(i)} = \mathsf{Hash}\big(\mathbf{u}'^{(i)}, \mathbf{e}'^{(i)}, \mathtt{Salt}, i\big)$	
Compute $c_0 = \mathsf{Hash}\big(c_0^{(1)}, \ldots, c_0^{(t)}\big)$	
Compute $c_1 = \mathsf{Hash}\big(c_1^{(1)}, \ldots, c_1^{(t)}\big)$	
Compute $c = \mathsf{Hash}\big(c_0, c_1\big)$	
Generate $\big(\beta^{(1)}, \ldots, \beta^{(t)}\big) = \mathsf{GenCh}_1\big(c, \mathtt{Msg}, \mathtt{Salt}\big)$	
For $i = 1, \ldots, t$:	
Compute $\mathbf{y}^{(i)} = \mathbf{u}'^{(i)} + \beta^{(i)}\mathbf{e}'^{(i)}$	
Compute $h^{(i)} = \mathsf{Hash}\big(\mathbf{y}^{(i)}\big)$	
Compute $h = \mathsf{Hash}\big(h^{(1)}, \ldots, h^{(t)}\big)$	
Generate $\big(b^{(1)}, \ldots, b^{(t)}\big) = \mathsf{GenCh}_2\big(c, \beta^{(1)}, \ldots, \beta^{(t)}, h, \mathtt{Msg}, \mathtt{Salt}\big)$	
For $i = 1, \ldots, t$:	
If $b^{(i)} = 0$:	
Set $f^{(i)} := \big(\mathbf{y}^{(i)}, \sigma^{(i)}, c_1^{(i)}\big)$	
Else:	
Set $f^{(i)} := \big(\mathtt{Seed}^{(i)}, c_0^{(i)}\big)$	
Set $\mathtt{Signature} = \big\{\mathtt{Salt}, c, h, \{f^{(i)}\}_{i=1,\ldots,t}\big\}$	
	$\xrightarrow{\mathtt{Signature}}$

Figure 4: The CROSS-fast signature scheme: signature generation

Private Key $\mathbf{e} \in G$
Public Key $G \subseteq \mathbb{E}^n$, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $\mathbf{s} = \mathbf{e}\mathbf{H}^\top \in \mathbb{F}_p^{n-k}$

PROVER		VERIFIER
	$\xrightarrow{\texttt{Signature}}$	
		Generate $\left(\beta^{(1)}, \ldots, \beta^{(t)}\right) = \mathsf{GenCh}_1\left(c, \texttt{Msg}, \texttt{Salt}\right)$
		Generate $\left(b^{(1)}, \ldots, b^{(t)}\right) = \mathsf{GenCh}_2\left(c, \beta^{(1)}, \ldots, \beta^{(t)}, h, \texttt{Msg}, \texttt{Salt}\right)$
		For $i = 1, \ldots, t$:
		If $b^{(i)} = 0$:
		Set $h^{(i)} = \mathsf{Hash}\left(\mathbf{y}^{(i)}\right)$
		Compute $\widetilde{\mathbf{s}}^{(i)} = \sigma^{(i)}(\mathbf{y}^{(i)})\mathbf{H}^\top - \beta^{(i)}\mathbf{s}$
		Set $c_0^{(i)} = \mathsf{Hash}\left(\widetilde{\mathbf{s}}^{(i)}, \sigma^{(i)}, \texttt{Salt}, i\right)$
		Else:
		Sample $(\mathbf{e}'^{(i)}, \mathbf{u}'^{(i)}) \xleftarrow{\texttt{Seed}^{(i)}} G \times \mathbb{F}_p^n$
		Set $c_1^{(i)} = \mathsf{Hash}\left(\mathbf{u}'^{(i)}, \mathbf{e}'^{(i)}, \texttt{Salt}, i\right)$
		Compute $\mathbf{y}^{(i)} = \mathbf{u}'^{(i)} + \beta^{(i)}\mathbf{e}'^{(i)}$
		Compute $h^{(i)} = \mathsf{Hash}(\mathbf{y}^{(i)})$
		Verify $h = \mathsf{Hash}\left(h^{(1)}, \ldots, h^{(t)}\right)$
		Compute $c_0 = \mathsf{Hash}\left(c_0^{(1)}, \ldots, c_0^{(t)}\right)$
		Compute $c_1 = \mathsf{Hash}\left(c_1^{(1)}, \ldots, c_1^{(t)}\right)$
		Verify $c = \mathsf{Hash}\left(c_0, c_1\right)$

Figure 5: The CROSS-fast signature scheme: signature verification

Notice that the only optimizations we have applied are aggregating the commitments and first responses and postponing their verification to the end of the verification algorithm. Indeed, in every round, the verifier possesses $\mathbf{y}^{(i)}$ (either it is received directly from the prover, or it is recomputed locally from $\texttt{Seed}^{(i)}$). Instead of responding with $\left(h^{(1)}, \ldots, h^{(t)}\right)$, the prover more conveniently sends

$$h = \mathsf{Hash}\left(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(t)}\right).$$

To generate the second challenge, the prover uses h (this value is also included in the signature). After executing all rounds, the verifier can locally recompute h. If the verifier's computation matches the given value of h, either a hash collision has occurred or the t vectors $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(t)}$ were indeed valid.
The public key size is

$$|\mathsf{pk}| = (n-k)\lceil \log_2(p) \rceil + 2\lambda.$$

The signature size is

$$|\texttt{Signature}| = \underbrace{6\lambda}_{\texttt{Salt},\ c,\ h} + w \cdot \underbrace{3\lambda}_{\text{Responses for } b^{(i)}=1} + (t-w)\underbrace{\left(2\lambda + n\lceil \log_2(p) \rceil + m\lceil \log_2(z) \rceil\right)}_{\text{Responses for } b^{(i)}=0}. \tag{1}$$

For this, note that the responses $b^{(i)} = 1$ appears w times and $b^{(i)} = 0$ appears $t-w$ times. The response for $b^{(i)} = 1$ consists of $\texttt{Seed}^{(i)}$ of size λ and $c_0^{(i)}$ of size 2λ, whereas for $b^{(i)} = 0$ the response consists of $y^{(i)}$, which needs $n\lceil \log_2(p) \rceil$ bits, $\sigma^{(i)}$, which needs $m\lceil \log_2(z) \rceil$ bits and $c_1^{(i)}$ of size 2λ. When R-SDP is considered, m needs to be replaced with n.

4.4 CROSS-small and CROSS-balanced

We now consider the signature scheme used in CROSS-small and CROSS-balanced. These variants consider the weight w to be very close to t, say, significantly larger than $t/2$. Two further optimizations can be employed in this case; we describe them before introducing the resulting signature scheme.

Using a Seed Tree For each execution of the signing algorithm, we have t seeds $\texttt{Seed}^{(1)}, \ldots, \texttt{Seed}^{(t)}$, which are used to sample $\mathbf{u}'^{(i)}$ and $\mathbf{e}'^{(i)}$ in each round. We generate these seeds employing a tree

structure, having as a root $\texttt{MSeed}||\texttt{Salt}$, where $\texttt{MSeed} \xleftarrow{\$} \{0;1\}^\lambda$ is sampled at the beginning of the signing algorithm. The seeds $\texttt{Seed}^{(1)}, \ldots, \texttt{Seed}^{(t)}$ are the leaves in the base layer of the tree.

Notice that the prover is asked for a seed in all but $t-w$ rounds. Hence, the prover has to reveal all but $t-w$ seeds; the maximum number of tree nodes that need to be revealed is $(t-w)\log_2\left(\frac{t}{t-w}\right)$. So, sending all seeds has an overall communication cost of

$$|\texttt{SeedPath}| = \lambda(t-w)\log_2\left(\frac{t}{t-w}\right). \tag{2}$$

Using a Merkle Tree for Commitments In each round, the verifier can always locally recompute one of the two commitments. Since the second challenge has fixed weight w which is close to t, the verifier will recompute the majority of the commitments $c_1^{(i)}$ and only a few of the commitments $c_0^{(i)}$. For what concerns the commitments $c_1^{(i)}$, the prover can more conveniently commit to a unique hash digest

$$c_1 = \mathsf{Hash}\big(c_1^{(1)}, \ldots, c_1^{(t)}\big).$$

Let $J \subseteq \{1, \ldots, t\}$ be the support of $\big(b^{(1)}, \ldots, b^{(t)}\big)$ (i.e., the set of indices i for which $b^{(j)} = 1$): the verifier will possess all the $c_1^{(i)}$ with $i \in J$, and will miss only those with index $i \notin J$. For each index i, the prover can include $c_1^{(i)}$ in the second response. This way, the overall cost associated with the commitments $c_1^{(1)}, \ldots, c_1^{(t)}$ is

$$|\texttt{Com}(1)| = \underbrace{2\lambda}_{c_1} + \underbrace{2(t-w)\lambda}_{c_1^{(i)},\ i \notin J} = 2\lambda(t-w+1). \tag{3}$$

For the commitments $c_0^{(i)}$, instead the prover can prepare a Merkle tree $\mathcal{T}$ using $c_0^{(1)}, \ldots, c_0^{(t)}$ as the leaves in the base layer. We denote by c_0 the root of $\mathcal{T}$. The verifier can locally recompute all the $c_0^{(i)}$ with $i \notin J$; to certify that the prover has indeed committed to these values, the verifier will ask for the Merkle proofs of $c_0^{(i)}$.

Notice that there are $t-w$ rounds in which the second challenge has value 0. Naively, sending all of these proofs would require $(t-w)\log_2(t)$ hash digests ($\log_2(t)$ digests for each of the $t-w$ leaves for which the proof is required). More conveniently, one can consider that these proofs will have some common paths: indeed, the number of distinct hashes that are needed is not greater than $(t-w)\log_2\left(\frac{t}{t-w}\right)$. Hence, the overall cost associated with the commitments $c_0^{(1)}, \ldots, c_0^{(t)}$ is upper bounded by

$$|\texttt{Com}(0)| = \underbrace{2\lambda}_{c_0} + \underbrace{2\lambda(t-w)\log_2\left(\frac{t}{t-w}\right)}_{\text{Merkle proof for } c_0^{(i)},\ i \notin J} = 2\lambda\left(1 + (t-w)\log_2\left(\frac{t}{t-w}\right)\right). \tag{4}$$

The Resulting Signature Scheme Summarizing all the optimizations that we are considering, the final scheme follows this workflow:

Signing:

1. sample a salt $\texttt{Salt} \xleftarrow{\$} \{0;1\}^{2\lambda}$;
2. sample a master seed $\texttt{MSeed} \xleftarrow{\$} \{0;1\}^\lambda$ and create the seed tree with t leaves $\texttt{Seed}^{(1)}, \ldots, \texttt{Seed}^{(t)}$ in the base layer. Seed number i, i.e., $\texttt{Seed}^{(i)}$, is employed to sample $\mathbf{u}'^{(i)}$ and $\mathbf{e}'^{(i)}$, which are used for round i. Commitments are stored as $c = \mathsf{Hash}(c_0, c_1)$;
3. for round $i = 1, \ldots, t$ compute the restricted transformation $\sigma^{(i)}$ and the commitments $c_0^{(i)}, c_1^{(i)}$ as defined in CROSS-ID. Salt the hash function used to compute commitments with $\texttt{Salt}$ and the round index i;
4. construct the Merkle tree $\mathcal{T}$ with commitments $c_1^{(1)}, \ldots, c_1^{(t)}$;

5. generate the first challenge $\left(\beta^{(1)}, \ldots, \beta^{(t)}\right)$ using the message, the salt, and the commitments;
6. compute $\left(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(t)}\right)$ according to CROSS-ID, and hash all of these vectors into h;
7. generate the second challenge $\mathbf{b} = \left(b^{(1)}, \ldots, b^{(t)}\right) \in \{0;1\}^t$ from the hash of the message, $\texttt{Salt}$, commitments, $\left(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(t)}\right)$, and h, which has fixed Hamming weight w; set J as the support of $\mathbf{b}$, i.e., as the set of indices i for which $b^{(i)} = 1$;
8. compute $\texttt{SeedPath}$ as the ensemble of intermediate seeds in the SeedTree which are needed to recompute all seeds $\texttt{Seed}^{(i)}$, for $i \in J$;
9. set $\texttt{MerkleProof}$ as the Merkle proof for leaves $\{c_0^{(i)}\}_{i \notin J}$;
10. the signature is obtained as

$$\texttt{Signature} = \left\{\texttt{Salt}, c, h, \texttt{SeedPath}, \texttt{MerkleProof}(\mathcal{T}_0), \{\mathbf{y}^{(i)}, \sigma^{(i)}, c_1^{(i)}\}_{i \notin J}\right\}.$$

Verification:

1. generate the first challenge $\left(\beta^{(1)}, \ldots, \beta^{(t)}\right)$ from $\texttt{Msg}$, $\texttt{Salt}$, c;
2. generate the second challenge $\left(b^{(1)}, \ldots, b^{(t)}\right)$ from $\texttt{Msg}$, $\texttt{Salt}$, c, $\beta^{(1)}, \ldots, \beta^{(t)}$, and h;
3. using $\texttt{SeedPath}$, generate the seeds $\{\texttt{Seeds}^{(i)}\}_{i \in J}$;
4. for $i \in J$, recompute $c_1^{(i)}$, $\mathbf{y}^{(i)}$, and $h^{(i)}$;
5. for $i \notin J$, compute $h^{(i)} = \mathsf{Hash}(\mathbf{y}^{(i)})$, and $c_1^{(i)}$;
6. using $\texttt{MerkleProof}$ and $\{c_0^{(i)}\}_{i \notin J}$, recompute c_0;
7. compute $c_1 = \mathsf{Hash}\left(c_1^{(1)}, \ldots, c_1^{(t)}\right)$;
8. verify $c = \mathsf{Hash}\left(c_0, c_1\right)$;
9. verify $h = \mathsf{Hash}\left(h^{(1)}, \ldots, h^{(t)}\right)$.

Full details on how the signing and verification algorithm operate as given in Figures 6 and 7. We have implicitly defined the functions to work with the seed tree (i.e., generate it from the root and recover the path to generate some specific leaves in the base layer), as well as work with Merkle trees. The functions GenCh_1 and GenCh_2 generate the challenges by hashing the input into a length-2λ digest, with which a PRNG gets fed. The function GenCh_1 has co-domain $\left(\mathbb{F}_p^*\right)^t$, while GenCh_2 returns a random vector over $\mathbb{F}_2^t$ with Hamming weight w.

The public key size is

$$|\mathsf{pk}| = (n-k)\lceil \log_2(p) \rceil + 2\lambda.$$

The signature size is

$$|\texttt{Signature}| = \underbrace{6\lambda}_{h,\ c,\ \texttt{Salt}} + \underbrace{\lambda(t-w)\log_2\left(\frac{t}{t-w}\right)}_{\texttt{SeedPath}} + \underbrace{2\lambda\left(1 + (t-w)\log_2\left(\frac{t}{t-w}\right)\right)}_{\texttt{MerkleProof}} + \underbrace{(t-w)\left(\underbrace{2\lambda}_{c_1^{(i)}} + \underbrace{n\lceil\log_2(p)\rceil}_{\mathbf{y}^{(i)}} + \underbrace{m\lceil\log_2(z)\rceil}_{\sigma^{(i)}}\right)}_{f^{(i)},\ i \notin J}. \tag{5}$$

When R-SDP is considered, m needs to be replaced with n.

Private Key $\mathbf{e} \in G$
Public Key $G \subseteq \mathbb{E}^n$, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $\mathbf{s} = \mathbf{e}\mathbf{H}^\top \in \mathbb{F}_p^{n-k}$
Input Message $\mathtt{Msg}$
Output Signature $\mathtt{Signature}$

SIGNER	VERIFIER
Sample $\mathtt{MSeed} \xleftarrow{\$} \{0;1\}^\lambda$, $\mathtt{Salt} \xleftarrow{\$} \{0;1\}^{2\lambda}$	
Generate $\left(\mathtt{Seed}^{(1)}, \ldots, \mathtt{Seed}^{(t)}\right) = \mathsf{SeedTree}(\mathtt{MSeed}, \mathtt{Salt})$	
For $i = 1, \ldots, t$:	
Sample $(\mathbf{e}'^{(i)}, \mathbf{u}'^{(i)}) \xleftarrow{\mathtt{Seed}^{(i)}} G \times \mathbb{F}_p^n$	
Compute $\sigma^{(i)} \in G$ such that $\sigma^{(i)}(\mathbf{e}'^{(i)}) = \mathbf{e}$	
Set $\mathbf{u}^{(i)} = \sigma^{(i)}(\mathbf{u}'^{(i)})$	
Compute $\widetilde{\mathbf{s}}^{(i)} = \mathbf{u}^{(i)}\mathbf{H}^\top$	
Set $c_0^{(i)} = \mathsf{Hash}\left(\widetilde{\mathbf{s}}^{(i)}, \sigma^{(i)}, \mathtt{Salt}, i\right)$	
Set $c_1^{(i)} = \mathsf{Hash}\left(\mathbf{u}'^{(i)}, \mathbf{e}'^{(i)}, \mathtt{Salt}, i\right)$	
Set $\mathcal{T} = \mathsf{MerkleTree}\left(c_0^{(1)}, \ldots, c_0^{(t)}\right)$	
Compute $c_0 = \mathcal{T}.\mathsf{Root}(\,)$	
Compute $c_1 = \mathsf{Hash}\left(c_1^{(1)}, \ldots, c_1^{(t)}\right)$	
Compute $c = \mathsf{Hash}\left(c_0, c_1\right)$	
Generate $\left(\beta^{(1)}, \ldots, \beta^{(t)}\right) = \mathsf{GenCh}_1\left(c, \mathtt{Msg}, \mathtt{Salt}\right)$	
For $i = 1, \ldots, t$:	
Compute $\mathbf{y}^{(i)} = \mathbf{u}'^{(i)} + \beta^{(i)}\mathbf{e}'^{(i)}$	
Compute $h^{(i)} = \mathsf{Hash}\left(\mathbf{y}^{(i)}\right)$	
Compute $h = \mathsf{Hash}\left(h^{(1)}, \ldots, h^{(t)}\right)$	
Generate $\left(b^{(1)}, \ldots, b^{(t)}\right) = \mathsf{GenCh}_2\left(c, \beta^{(1)}, \ldots, \beta^{(t)}, h, \mathtt{Msg}, \mathtt{Salt}\right)$	
Set $J = \left\{i \mid b^{(i)} = 1\right\}$	
Set $\mathtt{SeedPath} = \mathsf{SeedPath}(\mathtt{MSeed}, \mathtt{Salt}, J)$	
For $i \notin J$:	
Set $f^{(i)} := \left(\mathbf{y}^{(i)}, \sigma^{(i)}, c_1^{(i)}\right)$	
Compute $\mathtt{MerkleProofs} = \mathcal{T}.\mathsf{Proofs}(\{1, \ldots, t\} \setminus J)$	
Set $\mathtt{Signature} = \left\{\mathtt{Salt}, c, h, \mathtt{SeedPath}, \mathtt{MerkleProofs}, \{f^{(i)}\}_{i \notin J}\right\}$	
	$\xrightarrow{\mathtt{Signature}}$

Figure 6: The CROSS signature scheme: signature generation

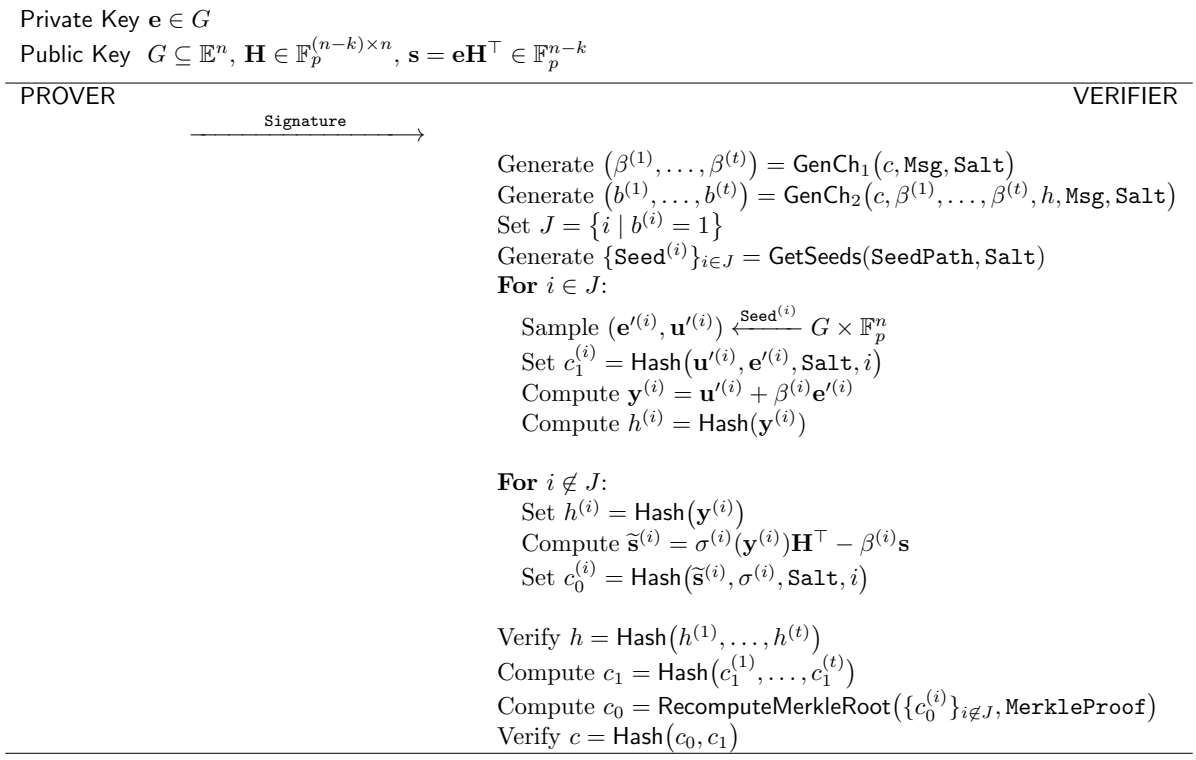

Figure 7: The CROSS signature scheme: signature verification

5 Procedural Description of CROSS

In the following, we provide a procedural description of the CROSS signature scheme primitives: KeyGen, Sign, and Verify. The latter two algorithms correspond to the realization of the signature generation and verification protocols reported in Figure 6 and Figure 7, respectively, while the KeyGen describes how the CROSS signature scheme keypair is generated.

The algorithmic triple presents minor procedural changes depending on whether the private key being employed belongs to $\mathbb{E}^n$ (CROSS-R-SDP), or to a proper subgroup of $\mathbb{E}^n$ (CROSS-R-SDP(G)). We report the algorithmic description with the following convention: portions in black are identical between the two variants, portions in teal are unique to CROSS-R-SDP, while portions in orange are unique to CROSS-R-SDP(G).

For efficiency in computation, we will represent, whenever possible, elements of $\mathbb{E}^n$ as length-n vectors over $\mathbb{F}_z$, and denote them with lowercase boldface Greek letters, e.g., $\boldsymbol{\eta}$. The choice of lowercase boldface Greek letters is made to avoid the lexical ambiguity which would arise between the syntax for a procedure invocation and the $\ell(\mathbf{e})$ notation. The same representation will also be employed for restricted transformations over elements of $\mathbb{E}^n$, such as σ, for which we will adopt the same notation. This notation is mutated by the fact that computing the application of σ to $\mathbf{e} \in \mathbb{E}^n$ corresponds to an element-wise multiplication over $\mathbb{F}_z$ of the components of their representations. Thus, we use σ in this section to denote the previous $\ell_G(\sigma)$. For the case of R-SDP(G), we will represent the length m vectors in $\mathbb{F}_z$ with lowercase Greek letters; we thus have, for instance, $\ell_G(\mathbf{e}'^{(i)}) = \tilde{\zeta}$.

To avoid ambiguity with the left-associative exponentiation operation, we move the round indexes, wherever they are needed, from the round-brackets enclosed superscripts to the subscript of the symbol they qualify, e.g., $\mathtt{Seed}^{(i)} \mapsto \mathtt{Seed}_i$. We will also report all temporary variables not reused across rounds without a subscript.

We will denote bit-strings employing a `monospace` font, while abstract data structures such as arrays and portions of trees will be denoted with a sans serif font. When either a cell of an array or an element of a vector must be singled out, we employ the usual square-bracket notation, and zero-based indexing; e.g., $\mathbf{v}[3]$ represents the fourth element of the vector $\mathbf{v}$. Table 1 summarizes the notation correspondence between protocol and pseudocode.

Table 1: Notation matches between protocol-level description and pseudocode

Protocol	**Pseudocode**	**Semantics**
$\ell_G(\mathbf{e})$	ζ	vector in $\mathbb{F}_z^m$ for $\mathbf{e}$
$\ell(\mathbf{e})$	η	$\zeta\mathbf{M}_G = \ell_G(\mathbf{e})\mathbf{M}_G \in \mathbb{F}_z^n$ for $\mathbf{e}$
$\ell_G(\sigma^{(i)})$	δ_i	vector in $\mathbb{F}_z^m$ for $\sigma^{(i)}$
$\ell(\mathbf{e}'^{(i)})$	$\tilde{\eta}_i$	vector in $\mathbb{F}_z^n$ for $\mathbf{e}'^{(i)}$
$\ell(\sigma^{(i)})$	σ_i	vector in $\mathbb{F}_z^n$ for $\sigma^{(i)}$
$\sigma^{(i)}$	$\mathbf{v}$	trans. on $\mathbb{E}^n$, resp. G
$\mathbf{u}^{(i)}$	$\mathbf{u}$	trans. $\mathbf{u}'^{(i)}$
$\tilde{\mathbf{s}}^{(i)}$	$\tilde{\mathbf{s}}$	syndrome of $\mathbf{u}^{(i)}$
$c_0^{(i)}$	$\mathtt{cmt}_0[i]$	Commitment 0, round i
$c_1^{(i)}$	$\mathtt{cmt}_1[i]$	Commitment 1, round i
$c_0 = \mathcal{T}.\mathtt{root}()$	d_0	root of Merkle tree $\mathcal{T}$ of commitment 0
c_1	d_1	Hash of commitment 1
	d_{01}	Hash of d_0, d_1
	d_m	Hash of message
	d_β	Hash of d_m,d_{01}, $\mathtt{Salt}$
$(\beta^{(1)}, \ldots, \beta^{(t)})$	$\mathtt{beta}$	first challenge
$\mathbf{e}'^{(i)}$	$\tilde{\mathbf{e}}$	trans. $\mathbf{e}$
$\mathbf{y}^{(i)}$	$\mathbf{y}_i$	response to first challenge
h	d_b	Hash of $y^{(1)}, \ldots, y^{(t)}$
$b^{(1)}, \ldots, b^{(t)}$	$\mathbf{b}$	second challenge
$f^{(i)}$	$\mathtt{rsp}_0, \mathtt{rsp}_1$	response in round i

5.1 Key Generation

Algorithm 1: KeyGen()

Data: λ: security parameter,
$g \in \mathbb{F}_p^*$: generator of a subgroup $\mathbb{E}$ of $\mathbb{F}_p^*$ with cardinality z
$\mathbb{E}^n$: restricted subgroup
$\mathbf{M}_G$: $m \times n$ matrix of $\mathbb{F}_z$ elements, employed to generate vectors $\boldsymbol{\eta} \in G \subset \mathbb{E}^n$
Input: None
Output: $\mathtt{pri} : \mathtt{Seed}_{\mathtt{sk}}$: private key seed, 2λ bits long;
$\mathtt{pub} : (\mathtt{Seed}_{\mathtt{pk}}, \mathbf{s})$ public key: $\mathtt{Seed}_{\mathtt{pk}}$ is a 2λ bit seed, to derive the non-systematic portion of a random parity-check matrix $\mathbf{H}$ and the $m \times n$ matrix of $\mathbb{F}_z$ elements $\mathbf{M}_G$;
$\mathbf{s} \in \mathbb{F}_p^{n-k}$ is the syndrome of $\mathbf{e}$ through $\mathbf{H}$

1 $\mathtt{Seed}_{\mathtt{sk}} \xleftarrow{\$} \{0,1\}^{2\lambda}$
2 $(\mathtt{Seede}, \mathtt{Seed}_{\mathtt{pk}}) \leftarrow \text{CSPRNG}\left(\mathtt{Seed}_{\mathtt{sk}}, \{0,1\}^{2\lambda} \times \{0,1\}^{2\lambda}\right)$
3 $(\mathbf{V}, \mathbf{W}) \leftarrow \text{CSPRNG}\left(\mathtt{Seed}_{\mathtt{pk}}, \mathbb{F}_p^{(n-k)\times k} \times \mathbb{F}_z^{m\times(n-m)}\right)$ | $\mathbf{V} \leftarrow \text{CSPRNG}\left(\mathtt{Seed}_{\mathtt{pk}}, \mathbb{F}_p^{(n-k)\times k}\right)$
4 $\mathbf{H} \leftarrow [\mathbf{V} \,|\mathbf{I}_{n-k}]$
$\mathbf{M}_G \leftarrow [\mathbf{W} \mid \mathbf{I}_m]$
5 $\boldsymbol{\zeta} \leftarrow \text{CSPRNG}\left(\mathtt{Seede}, \mathbb{F}_z^m\right)$ | $\boldsymbol{\eta} \leftarrow \text{CSPRNG}\left(\mathtt{Seede}, \mathbb{F}_z^n\right)$
$\boldsymbol{\eta} \leftarrow \boldsymbol{\zeta}\mathbf{M}_G$
6 **for** $j \leftarrow 0$ **to** $n-1$ **do**
7 $\quad \mathbf{e}[j] \leftarrow g^{\boldsymbol{\eta}[j]}$
8 **end**
9 $\mathbf{s} \leftarrow \mathbf{e}\mathbf{H}^\top$
10 $\mathtt{pri} \leftarrow \mathtt{Seed}_{\mathtt{sk}}$
11 $\mathtt{pub} \leftarrow (\mathtt{Seed}_{\mathtt{pk}}, \mathbf{s})$
12 **return** (pri,pub);

The first algorithm, KeyGen (Algorithm 1), has the task to generate uniformly at random a secret key, which is comprised of a restricted vector $\mathbf{e}$ from a restricted subgroup $G \subseteq \mathbb{E}^n$, and a public key, which is comprised of the parity-check matrix $\mathbf{H}$ and the syndrome $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$. In the case of CROSS-R-SDP(G), an additional matrix $\mathbf{M}_G \in \mathbb{F}_z^{m\times n}$, which has all the information for the subgroup G, is required to generate the restricted vector $\mathbf{e}$.

Through the following strategy, we reduce the amount of stored data in the private and public keys to a single seed, and a seed plus the syndrome $\mathbf{s}$, respectively. The keypair generation algorithm, KeyGen, starts by drawing a 2λ bit seed from the systemwide TRNG and considers this as the $\mathtt{Seed}_{\mathtt{sk}}$ only key material which is required to be stored as the private key (line 1) Employing a CSPRNG, $\mathtt{Seed}_{\mathtt{sk}}$, the private key seed is expanded into two different λ bit strings, $\mathtt{Seede}$ and $\mathtt{Seed}_{\mathtt{pk}}$ (line 2). $\mathtt{Seed}_{\mathtt{pk}}$ is employed to generate, through the expansion via a CSPRNG, all the material pertaining to the public matrix $\mathbf{H}$, and for the case of CROSS-R-SDP(G)) the matrix $\mathbf{M}_G$ (lines 3–5). To the end of minimizing the pressure on the CSPRNG, only the nonsystematic portions of $\mathbf{H}$ and $\mathbf{M}_G$ are expanded from the CSPRNG (namely $\mathbf{V}$ and $\mathbf{W}$).

Following the generation of $\mathbf{H}$ and $\mathbf{M}_G$, KeyGen computes the restricted error vector $\mathbf{e}$ in the subgroup G expanding through a CSPRNG the $\mathtt{Seede}$ binary string (line 5). For CROSS-R-SDP, this is done through a straightforward expansion of $\boldsymbol{\eta} = \ell(\mathbf{e})$ from $\mathtt{Seede}$. For CROSS-R-SDP(G), it is enough to sample a random vector $\boldsymbol{\zeta} = \ell_G(\mathbf{e}) \in \mathbb{F}_z^m$ and compute $\boldsymbol{\eta} = \ell(\mathbf{e}) = \ell_G(\mathbf{e})\mathbf{M}_G \in \mathbb{F}_z^n$. Then the restricted error vector $\mathbf{e}$ in the subgroup G is obtained (lines 6-8) computing by

$$\mathbf{e} = (g^{\boldsymbol{\eta}[0]}, \dots g^{\boldsymbol{\eta}[n-1]}) = (g^{\ell(\mathbf{e})_1}, \dots, g^{\ell(\mathbf{e})_n}) \in G.$$

Finally, once the value $\mathbf{e}$ is available the corresponding syndrome $\mathbf{s}$ through $\mathbf{H}$ is computed (line 9). The public key is then constituted by the seed $\mathtt{Seed}_{\mathtt{pk}}$ required to sample $\mathbf{V}$, $\mathbf{W}$, and by the syndrome $\mathbf{s}$. The private key solely consists of the secret seed $\mathtt{Seed}_{\mathtt{sk}}$, from which all the elements can be derived.

5.2 Signature Generation

The SIGN (Algorithm 2) procedure realizes operatively the signature protocol described in Figure 6, receiving as inputs a private key `pri` and the message to be signed, represented as an arbitrary length bit string `Msg`.

The first step in the signature procedure is to expand, starting from the secret seed, both the restricted error vector $\mathbf{e}$, the parity-check matrix $\mathbf{H}$ and, for the case of CROSS-R-SDP(G) only, the matrix $\mathbf{M}_G$ describing G. These operations are the same which are computed during the key generation algorithm (Algorithm 1) in lines 1 to 5. Once the key material is expanded, the SIGN procedure draws a λ bit string `Mseed`, and a 2λ bit string `Salt` from the system TRNG (line 3). These strings are provided as input to the procedure SEEDTREELEAVES which computes a sequence of t seeds $(\texttt{Seed}[0], \ldots, \texttt{Seed}[t-1])$, starting from them (line 4). SEEDTREELEAVES internally computes a binary tree of nodes, each containing a binary string containing 3λ pseudorandom bits concatenated to an integer index of the tree node. The index is determined by enumerating the nodes, level by level, starting from the root and proceeding from left to right at each level. The 3λ pseudorandom bits of each child node are composed of the output of a CSPRNG seeded with the binary string contained in its parent, concatenated with the 2λ bit string `Salt`. The bit string of the root node is obtained concatenating the λ bit string `Mseed`, and a 2λ bit string `Salt`. The first t leaves starting from the left of the generated binary tree are returned by SEEDTREELEAVES as the sequence of t seeds $(\texttt{Seed}[0], \ldots, \texttt{Seed}[t-1])$.

The SIGN (Algorithm 2) procedure then proceeds to the computation of the commitments for the t rounds of the CROSS-ID identification protocol (lines 5–15), employing one of the generated seeds per round. The purpose of the loop at lines (lines 5–15) is to compute the contents of the t-elements sequences cmt_0 and cmt_1, representing the commitments for each CROSS-ID protocol iteration (these correspond to the $c_0^{(j)}$ and $c_1^{(j)}$ values of the protocol level description, for $1 \leq j \leq t$).

To do so, the i-th iteration of the loop expands $\texttt{Seed}[i]$ concatenated with the 2λ bit string `Salt` and the round index i, to sample a restricted error and a randomly drawn $\mathbb{F}_p$ vector. The restricted error is represented as an n-element vector of the exponents of the subgroup $\mathbb{E}$, and denoted as $\boldsymbol{\eta}'_i \in \mathbb{F}_z^n$ at line 6. The $\boldsymbol{\eta}'_i$ is used together with the secret error $\boldsymbol{\eta}$ (also represented as a vector of exponents in $\mathbb{F}_z^n$) to compute a restricted transformation $\boldsymbol{\sigma}_i \in \mathbb{F}_z^n$, such that $\boldsymbol{\sigma}_i + \boldsymbol{\eta}'_i = \boldsymbol{\eta}$ (corresponding to $\ell_G(\boldsymbol{\sigma}_i) + \ell_G(\mathbf{e}'_i) = \ell_G(\mathbf{e})$ in the protocol level notation). This operatively amounts to a component-wise subtraction in $\mathbb{F}_z^n$, as all the operands are represented as n elements vectors of $\mathbb{F}_z^n$ (line 7). To apply the restricted transformation sigma, the SIGN procedure needs to convert its representation into a vector of n elements in $\mathbb{F}_p$. This is done by an element-wise computation of $g^{\boldsymbol{\sigma}_i[j]}$, for all $0 \leq j < n$ (lines 8–10) which yields the temporary value v. The transformation, now represented as a $\mathbb{F}_p$ vector is applied to the previously randomly drawn $\mathbb{F}_p$ vector $\mathbf{u}'_\mathbf{i}$. The application of the transformation is made via component-wise product on $\mathbb{F}_p$ (line 11). Once this is completed, the syndrome $\tilde{\mathbf{s}}$ of $\boldsymbol{u} = \boldsymbol{v} \star \boldsymbol{u}'$ through $\mathbf{H}$ is computed at line 12. The commitments for the current round, i, are finally computed at lines 13 and 14. $\mathsf{cmt}_0[i]$ is obtained as the hash digest of the syndrome $\tilde{\mathbf{s}}$, the transformation, the `Salt` and round number i (represented as byte aligned). In computing $\mathsf{cmt}_0[i]$ we leverage the possibility of compactly representing the transformation $\boldsymbol{\sigma}_i$ for the CROSS-R-SDP(G) case. Indeed, thanks to the linearity of the computation of the multiplication by $\mathbf{M}_G$, we can compute the value $\boldsymbol{\delta}_i$ such that $\boldsymbol{\delta}_i \mathbf{M}_G = \boldsymbol{\sigma}_i$, and replace $\boldsymbol{\sigma}_i$ with $\boldsymbol{\delta}_i$ in the computation of $\mathsf{cmt}_0[i]$. This can be efficiently done right after the generation of $\boldsymbol{\eta}'_i$ in the CROSS-R-SDP(G) case (line 6), when the values $\boldsymbol{\zeta}$ and $\boldsymbol{\zeta}'$ (such that $\boldsymbol{\zeta}_i \mathbf{M}_G = \boldsymbol{\eta}_i$ and $\boldsymbol{\zeta}'_i \mathbf{M}_G = \boldsymbol{\zeta}'_i$) are available. Replacing $\boldsymbol{\sigma}_i$ with $\boldsymbol{\delta}_i$ allows to reduce the amount of data to be digested by hashes, in turn speeding up the computation of $\mathsf{cmt}_0[i]$. $\mathsf{cmt}_1[i]$ is computed as the hash digest of the round seed $\texttt{Seed}[i]$ and `Salt`.

Once all commitments are prepared, the algorithm proceeds to compute the digest to derive the first challenge vector `beta` (β in the protocol level description, renamed to avoid ambiguities with the semantics of greek lowercase letters). The canonical approach of the Fiat-Shamir transform, as described in Figure 3, would obtain the digest which is employed to seed the CSPRNG generating `beta` through the result of computing the hash of the concatenation of the message `Msg`, the `Salt` and all commitments contained in the cmt_0 and cmt_1 commitment sequences. The SIGN procedure optimizes this approach, observing two facts. The first is that our fixed weight challenges have high weight (for the balanced and small version) and therefore, the commitments in the cmt_0, which are revealed when the binary challenge has value 0, are seldom revealed. It is, therefore, useful to compute a Merkle tree (hierarchical) hash of them alone to be able to reveal a less-than-linear (in t) amount of hashes in the signature. The SIGN procedure, therefore, computes the root of a Merkle tree d_0 having cmt_0 elements as leaves (line 16), while it computes a simple hash of the elements of cmt_1 obtaining the digest d_1. In the canonical Fiat-

Shamir approach, both d_0 and d_1 would be included in the signature to be recomputed by the verifier and checked. We save one digest in signature size, computing the digest d_{01} employing as input d_0 and d_1 (line 18) and including only d_{01} in the signature.

The second fact is that including the entire message in the hash input generating the seed which is expanded into beta requires the entire message to be streamed into the computation unit performing the signature. We, therefore, chose to include the digest of the message d_m as the input, allowing the parallel computation of it during the algorithm if more than a unit is available. We therefore compose the input of the hash (line 20) yielding the digest d_β concatenating d_m, d_{01}, $\mathtt{Salt}$.

Expanding d_β using a CSPRNG then yields the first challenge vector $\mathsf{beta} \in (\mathbb{F}_p^*)^t$ (line 21). Once this is done, the SIGN procedure computes the $\mathbf{y}_i$, $0 \leq i < t$, responses by computing the representation over $\mathbb{F}_p$ of $\boldsymbol{\eta}'_i$ (lines 23–25), and subsequently obtaining $\mathbf{y}_i$ as $\boldsymbol{u}'_i + \mathsf{beta}[i]\mathbf{e}'_i$.

For each round, the corresponding response $\mathbf{y}_i$ is computed from the challenge $\mathsf{beta}[i]$, the restricted error $\mathbf{e}'_i$, and $\boldsymbol{u}'_i$.

The hash digest d_b obtained from hashing all the $\mathbf{y}_i$ and d_β will then represent the first response and is included in the signature, after which its expansion with a CSPRNG yields the second challenge vector $\mathbf{b}$ (lines 29-30). The challenge vector $\mathbf{b}$ needs to be sampled from the set of binary strings of length t and weight w. We observe that since the digest d_b is public, the constant weight sampling process does not need to be implemented through a constant time algorithm. This, in turn, reduces the difficulty of obtaining both a correct and secure algorithm. Since the number of zeroes in $\mathbf{b}$ is by far smaller than the number of ones, we sample the positions of the zeroes in the string through extracting from the CSPRNG fed with d_b numbers in the $\{0, n-1\}$ range. If a drawn position already contains a zero, we discard the sampled number and sample a fresh one.

Once the $\mathbf{b}$ fixed weight challenge string is available, it is possible to compute which nodes of the Merkle tree should be included in the signature so that the verifier is able to recompute the d_0 value starting from them and the responses to a zero-valued challenge. This is performed by the MERKLEPROOF procedure (line 31), which takes the fixed-weight $\mathbf{b}$ vector as an input, together with the commitment sequence cmt_0. The procedure includes in the $\mathsf{MerkleProofs}$ data structure all the nodes of the Merkle tree, which are roots of the highest subtrees, such that they do not contain a leaf recomputable by the verifier.

The SIGN procedure then computes the second set of protocol responses. Our optimized approach represents the responses to a zero-valued challenge bit explicitly, keeping them in the sequences rsp_0 and rsp_1. By contrast, since the responses to one-valued challenge bits can be represented in a compact fashion by the seed employed in lines 5–15, we compactly represent all of them as roots of binary subtrees of the seed tree data structure. Indeed, the SEEDTREEPATHS procedure (called at line 32) takes as an input the master seed $\mathtt{Mseed}$ allowing to generate the seed tree and a bitset representing which leaves (indeed, seeds) are to be disclosed to the verifier. The SEEDTREEPATHS determines the nodes to be included in the $\mathsf{SeedPaths}$ data structure picking all the roots of subtrees of the binary tree such that their descendants are uniquely nodes to be disclosed.

The SIGN procedure moves onto filling the rsp_0 and rsp_1 sequences, both having length $t - w$. In particular, the loop at lines 36–42 iterates on the CROSS-ID protocol repetitions (indexed by i) and, for all rounds where the binary challenge is zero-valued (lines 37–41) it includes in the rsp_0 sequence the $\mathbf{y}_i$ vector and a compact representation of the restricted transformation (σ_i for CROSS-R-SDP, δ for CROSS-R-SDP(G)), while storing in the rsp_1 sequence the corresponding commit from the and cmt_1 sequence.

Finally, the signature $\mathtt{Signature}$ is composed concatenating and encoding in a compact fashion (described in Section 9, Packing and Unpacking) the following elements: 2λ-bit $\mathtt{Salt}$, the hash digest of the commitments d_{01}, the hash digest of the first responses d_b, the Merkle tree proof $\mathsf{MerkleProofs}$ of the $\mathsf{cmt}_0[i]$ for challenge bits $\mathbf{b}[i] = 0$, the seed tree path $\mathsf{SeedPath}$ for the rounds in which $\mathbf{b}[i] = 1$, and the response vectors rsp_0 and rsp_1.

Algorithm 2: SIGN($\mathtt{pri}, \mathtt{Msg}$)

Data: λ: security parameter,
$g \in \mathbb{F}_p^*$: generator of a subgroup $\mathbb{E}$ of $\mathbb{F}_p^*$ with cardinality z
$\mathbb{E}^n$: restricted subgroup
$\mathbf{M}_G$: $m \times n$ matrix of $\mathbb{Z}_z$ elements, employed to generate vectors $\boldsymbol{\eta} \in G \subset \mathbb{E}^n$
t: number of iterations of the ZKID protocol
$\mathcal{B}_w^t$: set of all binary strings with length w and Hamming weight t
c: a fixed constant, equal to the number of nodes in the seed tree
$\mathtt{dsc}$: a fixed constant, greater than t employed to obtain domain separation

Input: $\mathtt{pri}$: private key constituted of $\mathtt{Seed}_{\mathtt{sk}} \in \{0,1\}^{2\lambda}$
$\mathtt{Msg}$: message to be signed $\mathtt{Msg} \in \{0,1\}^*$

Output: Signature $\mathtt{Signature}$

Begin

// Key material expansion

$\boldsymbol{\eta}, \boldsymbol{\zeta}, \mathbf{H}, \mathbf{M}_G \leftarrow$ EXPANDPRIVATESEED($\mathtt{Seed}_{\mathtt{sk}}$) | $\boldsymbol{\eta}, \mathbf{H} \leftarrow$ EXPANDPRIVATESEED($\mathtt{Seed}_{\mathtt{sk}}$)

// Computation of commitments

$\mathtt{Mseed} \xleftarrow{\$} \{0,1\}^{\lambda}$, $\mathtt{Salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$

$(\mathtt{Seed}[0], \ldots, \mathtt{Seed}[t-1]) \leftarrow$ SEEDTREELEAVES($\mathtt{Mseed}, \mathtt{Salt}$)

for $i \leftarrow 0$ **to** $t-1$ **do**

$\boldsymbol{\zeta}', \boldsymbol{u}_i' \leftarrow \mathrm{CSPRNG}\left(\mathtt{Seed}[i]||\mathtt{Salt}||i+c, \mathbb{F}_z^m \times \mathbb{F}_p^n\right)$
$\boldsymbol{\delta}_i \leftarrow \boldsymbol{\zeta} - \boldsymbol{\zeta}'$
$\boldsymbol{\eta}_i' \leftarrow \boldsymbol{\zeta}' \mathbf{M}_G$
| $\boldsymbol{\eta}_i', \boldsymbol{u}_i' \leftarrow \mathrm{CSPRNG}\left(\mathtt{Seed}[i]||\mathtt{Salt}||i+c, \mathbb{F}_z^n \times \mathbb{F}_p^n\right)$

$\boldsymbol{\sigma}_i \leftarrow \boldsymbol{\eta} - \boldsymbol{\eta}_i'$

for $j \leftarrow 0$ **to** $n-1$ **do**

$\boldsymbol{v}[j] \leftarrow g^{\boldsymbol{\sigma}_i[j]}$

end

$\boldsymbol{u} \leftarrow \boldsymbol{v} \star \boldsymbol{u}_i'$ // $\star$ is component-wise product

$\tilde{\mathsf{s}} \leftarrow \boldsymbol{u}\mathbf{H}^\top$

$\mathsf{cmt}_0[i] \leftarrow$ HASH($\tilde{\mathsf{s}}||\boldsymbol{\delta}_i||\mathtt{Salt}||i+c+\mathtt{dsc}$) | $\mathsf{cmt}_0[i] \leftarrow$ HASH($\tilde{\mathsf{s}}||\boldsymbol{\sigma}_i||\mathtt{Salt}||i+c+\mathtt{dsc}$)

$\mathsf{cmt}_1[i] \leftarrow$ HASH($\mathtt{Seed}[i]||\mathtt{Salt}||i+c+\mathtt{dsc}$)

end

$\mathsf{d}_0 \leftarrow$ MERKLEROOT($\mathsf{cmt}_0[0], \ldots, \mathsf{cmt}_0[t-1]$)

$\mathsf{d}_1 \leftarrow$ HASH($\mathsf{cmt}_1[0] \;||\; \ldots \;||\; \mathsf{cmt}_1[t-1]$)

$\mathsf{d}_{01} \leftarrow$ HASH($\mathsf{d}_0 \;||\; \mathsf{d}_1$)

// First challenge vector extraction

$\mathsf{d}_m \leftarrow$ HASH(Msg)

$\mathsf{d}_\beta \leftarrow$ HASH($\mathsf{d}_m||\mathsf{d}_{01}||\mathtt{Salt}$)

$\mathsf{beta} \leftarrow \mathrm{CSPRNG}\left(\mathsf{d}_\beta, (\mathbb{F}_p^*)^t\right)$

// Computation of first round of responses

for $i \leftarrow 0$ **to** $t-1$ **do**

for $j \leftarrow 0$ **to** $n-1$ **do**

$\mathbf{e}'[j] \leftarrow g^{\boldsymbol{\eta}_i'[j]}$

end

$\mathbf{y}_i \leftarrow \boldsymbol{u}_i' + \mathsf{beta}[i]\mathbf{e}_i'$

end

28 // Second challenge vector extraction
29 $\mathsf{d}_b \leftarrow \textsc{Hash}(\mathbf{y}_0 || \ldots || \mathbf{y}_{t-1} || \mathsf{d}_\beta)$
30 $\mathbf{b} \leftarrow \text{CSPRNG}\left(\mathsf{d}_b, \mathcal{B}^t_{(w)}\right)$
// Computation of second round of responses
31 $\mathsf{MerkleProofs} \leftarrow \textsc{MerkleProof}((\mathsf{cmt}_0[0], \ldots, \mathsf{cmt}_0[t-1]), \mathbf{b})$
32 $\mathsf{SeedPath} \leftarrow \textsc{SeedTreePaths}(\mathtt{Mseed}, \mathbf{b})$
// Signature composition
33 $\mathsf{rsp}_0 \leftarrow (\mathbb{F}_p^n \times \mathbb{F}_z^m)^{t-w}$ | $\mathsf{rsp}_0 \leftarrow (\mathbb{F}_p^n \times \mathbb{F}_z^n)^{t-w}$
34 $\mathsf{rsp}_1 \leftarrow (\{0,1\}^\lambda)^{t-w}$ // empty array
35 $j \leftarrow 0$
36 **for** $i \leftarrow 0$ **to** $t-1$ **do**
37 **if** $(\mathbf{b}[i] = 0)$ **then**
// $\mathsf{cmt}_0[i]$ is recomputed by the verifier, $\mathsf{cmt}_1[i]$ must be sent
38 $\mathsf{rsp}_0[j] \leftarrow (\mathbf{y}_i, \boldsymbol{\delta}_i)$ | $\mathsf{rsp}_0[j] \leftarrow (\mathbf{y}_i, \boldsymbol{\sigma}_i)$
39 $\mathsf{rsp}_1[j] \leftarrow \mathsf{cmt}_1[i]$
40 $j \leftarrow j + 1$
41 **end**
42 **end**
43 $\mathtt{Signature} \leftarrow \mathtt{Salt} \,||\, \mathsf{d}_{01} \,||\, \mathsf{d}_b \,||\, \mathsf{MerkleProofs} \,||\, \mathsf{SeedPath} \,||\, \mathsf{rsp}_0 \,||\, \mathsf{rsp}_1$
// all Signature components are encoded as binary strings
44 **return** Signature
45 **end**

5.3 Signature Verification

The VERIFY procedure implements the verification procedure described in Figure 7. The procedure takes as input the public key `pub`, the message on which the signature should be verified `Msg` and the `Signature`. The procedure outputs a single Boolean value, `True` or `False`, depending on whether the signature is valid or not.

The first step of the signature verification procedure is the expansion of the seed contained in the public key, $\mathtt{Seed}_{\mathtt{pk}}$ into the parity-check matrix $\mathbf{H}$, and, for CROSS-R-SDP(G), the matrix $\mathbf{M}_G$ (lines 2-4). Once the key material is available, the procedure moves onto the recomputation of the challenge vectors beta and $\mathbf{b}$. To this end, $\mathbf{b}$ is recomputed through the same CSPRNG technique employed to generate it during the signature operation, seeding the CSPRNG with d_b (line 8). The input to the CSPRNG d_β (line 6) required to generate beta is obtained hashing the concatenation of the message digest d_m (obtained at line 5) with the commitments digest d_{01} and the `Salt` contained in the signature.

The VERIFY procedure then regenerates the seeds required to recompute the values of $\mathsf{cmt}_1[i]$ for all the protocol iterations i where the binary challenge vector $\mathbf{b}$ has value 1 (line 9). This is performed by the REBUILDSEEDTREELEAVES procedure, which takes the SeedPath data structure, the challenge vector $\mathbf{b}$ and the `Salt`. The procedure computes, starting from the information contained in $\mathbf{b}$ the roots of the highest subtrees containing only leaves to be regenerated, and places the nodes contained in SeedPath into them. It subsequently computes, top-down, all the required leaves. Having generated the sequence $(\mathtt{Seed}[0], \ldots, \mathtt{Seed}[t-1])$, for which only revealed seeds contain valid values, the VERIFY procedure now computes the values in the cmt_0 and cmt_1, sequences, as well as the values $\mathbf{y}_i$ of the first responses, depending on the challenge bit $\mathbf{b}[i]$ (lines 10 - 31).

If $\mathbf{b}[i] = 1$ (lines 12-19), the VERIFY procedure re-computes the value of $\mathsf{cmt}_1[i]$ by hashing $\mathsf{Seed}[i]$ with the `Salt` (line 13). Subsequently, the procedure expands $\mathsf{Seed}[i]$, obtained as a leaf of the seed tree, to recompute the $\mathbf{e}'$ vector (line 14 –17), performing the same expansion via CSPRNG done by the signature procedure, depending on the CROSS variant (random sampling plus translation from the exponent representation into the vector-over-$\mathbb{F}_p$ one). Similarly, the value of $\mathbf{u}'$ is recovered from a

CSPRNG expansion. Finally, VERIFY computes the values of the first response $\mathbf{y}_i$ using the reconstructed first challenge $\mathsf{beta}[i]$ and $\mathbf{e}', \mathbf{u}'$ (line 18).

If $\mathbf{b}[i] = 0$, the VERIFY procedure re-computes the value of $\mathsf{cmt}_0[i]$. Therefore, it is first verified that the transformation contained in rsp_0, represented as n elements of $\mathbb{F}_z$ in $\boldsymbol{\sigma}$ for CROSS-R-SDP, or as m elements of $\mathbb{F}_z$ in $\boldsymbol{\delta}$ for CROSS-R-SDP(G), is a valid element of the restricted subgroup G, checking if all the values in the encoded vector of either $\boldsymbol{\sigma}$ or $\boldsymbol{\delta}$ are are in the appropriate range $\{0, \ldots, z-1\}$ (line 21), and if needed, reconstructing $\boldsymbol{\sigma}$ from $\boldsymbol{\delta}$ via a multiplication by $\mathbf{M}_G$. If so, the transformation is applied to the provided $\mathbf{y}_i$, which is afterward used to compute the syndrome $\tilde{\mathbf{s}}$ (lines 25–26). Hashing the syndrome $\tilde{\mathbf{s}}$, the transformation ($\boldsymbol{\sigma}$ or $\boldsymbol{\delta}$) and the `Salt` yields the digest of $\mathsf{cmt}_0[i]$ (line 27). The corresponding value of the digest of $\mathsf{cmt}_1[i]$ is copied from the rsp_1 sequence (line 28).

The VERIFY procedure is now able to recompute the digests d_0 and d_1 (lines 33-34): we denote the recomputed values as d'_0 and d'_1. The digest d'_0 is obtained via the RECOMPUTEMERKLEROOT procedure, which receives the sequence of commitments cmt_0, where part of them were recomputed and part of them received as the MerkleProofs data structure in the signature. The procedure performs a hierarchical hash of the said commitments, obtaining d'_0 (line 33). The d'_1 digest is obtained performing a hash of the cmt_1 sequence (line 34). Finally, d'_0 and d'_1 are employed as the inputs to a hash call, to obtain d'_{01} (line 35), which, provided that the signature is valid, should be matching its counterpart d_{01} contained in the signature itself. The last computation performed by the VERIFY procedure is the recomputation of the digest d_b (line 36), (we denote the variable holding the recomputed value d'_b) obtained hashing together both the receive $\mathbf{y}_i$ as part of the rsp_0 sequence, and the ones which were computed at line 18, in increasing order of the value of i. The VERIFY procedure determines whether both d'_{01} matches d_{01} and d'_b matches d_b (line 37): if this is the case, the signature is valid and VERIFY returns `True`, otherwise it returns `False`.

Algorithm 3: CROSS-VERIFY($\mathtt{pub}, \mathtt{Msg}, \mathtt{Signature}$)

Data: λ: security parameter,
$g \in \mathbb{F}_p^*$: generator of a subgroup $\mathbb{E}$ of $\mathbb{F}_p^*$ with cardinality z
$\mathbb{E}^n$: restricted subgroup
$\mathbf{M}_G$: $m \times n$ matrix of $\mathbb{F}_z$ elements, employed to generate vectors $\boldsymbol{\eta} \in G \subset \mathbb{E}^n$
t: number of iterations of the ZKID protocol
$\mathcal{B}_w^t$: set of all binary strings with length w and Hamming weight t
c: a fixed constant, equal to the number of nodes in the seed tree
$\mathtt{dsc}$: a fixed constant, greater than t employed to obtain domain separation

Input: $\mathtt{pub}$: $(\mathtt{Seed}_{\mathtt{pk}}, \mathbf{s})$ public key: $\mathtt{Seed}_{\mathtt{pk}}$ is a 2λ bit seed to derive the non-systematic portion of a random parity-check matrix $\mathbf{H}$ and the $m \times n$ matrix of $\mathbb{F}_z$ elements $\mathbf{M}_G$
$\mathtt{Msg}$: message to verify the signature on; $\mathtt{Msg} \in \{0,1\}^*$
$\mathtt{Signature}$: signature obtained encoding as binary the tuple $(\mathtt{Salt}, \mathsf{d}_{01}, \mathsf{d}_b, \mathsf{MerkleProofs}, \mathsf{SeedPath}, \mathsf{rsp}_0, \mathsf{rsp}_1)$

Output: a Boolean value, $\{\mathtt{True}, \mathtt{False}\}$, indicating if the signature is verified or not

Begin
  // Key material expansion
  $(\mathbf{V}, \mathbf{W}) \leftarrow \text{CSPRNG}\left(\mathtt{Seed}_{\mathtt{pk}}, \mathbb{F}_p^{(n-k)\times k} \times \mathbb{F}_z^{m\times(n-m)}\right)$ | $\mathbf{V} \leftarrow \text{CSPRNG}\left(\mathtt{Seed}_{\mathtt{pk}}, \mathbb{F}_p^{(n-k)\times k}\right)$
  $\mathbf{H} \leftarrow [\mathbf{V} \mid \mathbf{I}_{n-k}]$
  $\mathbf{M}_G \leftarrow [\mathbf{W} \mid \mathbf{I}_m]$
  // Challenge recomputation
  $\mathsf{d}_m \leftarrow \text{HASH}(\mathsf{Msg})$
  $\mathsf{d}_\beta \leftarrow \text{HASH}(\mathsf{d}_m || \mathsf{d}_{01} || \mathtt{Salt})$
  $\mathsf{beta} \leftarrow \text{CSPRNG}\left(\mathsf{d}_\beta, (\mathbb{F}_p^*)^t\right)$
  $\mathbf{b} \leftarrow \text{CSPRNG}\left(\mathsf{d}_b, \mathcal{B}_w^t\right)$
  $(\mathtt{Seed}[0], \ldots, \mathtt{Seed}[t-1]) \leftarrow \text{REBUILDSEEDTREELEAVES}(\mathsf{SeedPath}, \mathbf{b}, \mathtt{Salt})$
  $j \leftarrow 0$
  for $i \leftarrow 0$ **to** $t-1$ **do**
    if $(\mathbf{b}[i] = 1)$ **then**
      $\mathsf{cmt}_1[i] \leftarrow \text{HASH}(\mathtt{Seed}[i] || \mathtt{Salt} || i + c + \mathtt{dsc})$
      $\boldsymbol{\zeta}', \boldsymbol{u}_i' \leftarrow \text{CSPRNG}\left(\mathtt{Seed}[i] || \mathtt{Salt} || i + c, \mathbb{F}_z^m \times \mathbb{F}_p^n\right)$; $\boldsymbol{\eta}' \leftarrow \boldsymbol{\zeta}' \mathbf{M}_G$ | $\boldsymbol{\eta}', \boldsymbol{u}_i' \leftarrow \text{CSPRNG}\left(\mathtt{Seed}[i] || \mathtt{Salt} || i + c, \mathbb{F}_z^n \times \mathbb{F}_p^n\right)$
      for $j \leftarrow 0$ **to** $n-1$ **do**
        $\mathbf{e}'[j] \leftarrow g^{\boldsymbol{\eta}'[j]}$
      end
      $\mathbf{y}_i \leftarrow \mathbf{u}' + \mathsf{beta}[i]\mathbf{e}'$
    end
    else
      $(\mathbf{y}_i, \boldsymbol{\delta}_i) \leftarrow \mathsf{rsp}_0[j]$; verify $\boldsymbol{\delta}_i \in G$; $\boldsymbol{\sigma}_i \leftarrow \boldsymbol{\delta} \mathbf{M}_G$ | $(\mathbf{y}_i, \boldsymbol{\sigma}_i) \leftarrow \mathsf{rsp}_0[j]$; verify $\boldsymbol{\sigma}_i \in G$
      for $j \leftarrow 0$ **to** $n-1$ **do**
        $\mathbf{v}[j] \leftarrow g^{\boldsymbol{\sigma}[j]}$
      end
      $\mathbf{y}' \leftarrow \mathbf{v} \star \mathbf{y}_i$
      $\tilde{\mathbf{s}} \leftarrow \mathbf{y}' \mathbf{H}^\top - \mathsf{beta}[i]\mathbf{s}$
      $\mathsf{cmt}_0[i] \leftarrow \text{HASH}(\tilde{\mathbf{s}} || \boldsymbol{\delta}_i || \mathtt{Salt} || i + c + \mathtt{dsc})$ | $\mathsf{cmt}_0[i] \leftarrow \text{HASH}(\tilde{\mathbf{s}} || \boldsymbol{\sigma}_i || \mathtt{Salt} || i + c + \mathtt{dsc})$
      $\mathsf{cmt}_1[i] \leftarrow \mathsf{rsp}_1[j]$
      $j \leftarrow j + 1$
    end
  end

```
32
33    d'_0 ← RecomputeMerkleRoot(cmt_0, MerkleProofs, b)
34    d'_1 ← Hash(cmt_1[0] || ... || cmt_1[t − 1])
35    d'_01 ← Hash(d'_0 || d'_1)
36    d'_b ← Hash(y_0||...||y_{t−1})
37    if (d_01 = d'_01 and d_b = d'_b) then
38        return True
39    end
40    return False
41 end
```

Note that the values of $\boldsymbol{\sigma}$, published in the signature and hashed, are in redundant-double zero representation for efficiency reasons. No loss in signature size takes place. Furthermore, whenever vectors over $\mathbb{F}_p$ or $\mathbb{F}_z$ are used as inputs to hash functions in Algorithm 2 and Algorithm 3, we represent them in their bit-packed form to reduce pressure on the hash function.

6 Design Rationale

This section explains the design rationale behind CROSS. We address each choice we make for the signature scheme.

Code-Based. Within the NIST quantum-resistant standardization process, three digital signature schemes are already chosen for standardization: CRYSTALS-Dilithium, FALCON, and SPHINCS+. As these schemes are hash-based and lattice-based, it is preferable to have also signature schemes based on other hard problems. Seeing the recent breaks in multivariate and isogeny-based cryptography, it seems prudent to standardize several signature schemes relying on different primitives. Hard problems from coding theory, such as syndrome decoding and codeword finding for random codes, have worst-case hardness guarantees, coming from the NP-completeness proof in [14, 10]. Furthermore, despite a significant amount of research effort devoted to the task, no subexponential solver, neither classical nor quantum, is available for such problems [50]. The decoding problem for restricted error vectors has also been proven to be NP-hard [53], and the best-known decoders [23, 8] show an even larger computational complexity than for the classical decoding problem.

Zero-Knowledge Protocol. Leveraging the Fiat-Shamir transform on a ZK protocol gives the signature scheme high-security guarantees. We can provide an EUF-CMA security proof of CROSS, and the long history of classical code-based ZK protocols establishes them as safe solutions for modern signature schemes.

Restricted Errors. Standard code-based ZK protocols rely on the hardness of the Hamming-metric decoding problem. However, their ZK protocols suffer from large communication costs. The bulk of this comes from having to communicate permutations. The introduction of restricted errors to ZK protocols allows us to circumvent the need for costly permutations while at the same time maintaining the NP-hardness of the underlying problem [53]. Additionally, the restriction allows us to represent the restricted vectors and their transformations compactly.

Choice of $\mathbb{E}$. One could also use a different restriction on the entries of the error vector. However, to reduce the communication cost within the ZK protocol (and therefore also the signature size of the resulting signature scheme), the particular form of $\mathbb{E}$ is crucial. Having that each entry of a restricted vector is given by g^ℓ, it is enough to send ℓ. In other words, we are exploiting the fact that $(\mathbb{E}^n, \star)$ is isomorphic to $(\mathbb{F}_z^n, +)$. Having that the group $(\mathbb{E}^n, \star)$ is transitive allows us to represent also the transformations in a compact way; in fact, this form of communication requires only $n \log_2(z)$ bits.

Choice of G. Since the syndrome equations are linear under addition, having a multiplicative subgroup of $\mathbb{E}^n$ will not harm the security of the underlying problem. To consider the subgroup $(G, \star) \subset (\mathbb{E}^n, \star)$ allows us to further reduce the signature sizes, as any vector $\mathbf{e}$ (and thus also its associated transformations) can be represented using only $m \log_2(z)$ bits.

Choice of CROSS-ID. There are already several code-based ZK protocols, though these rely on the Hamming-metric syndrome decoding problem. One can replace this problem with the R-SDP in each ZK protocol and reduce the signature sizes significantly. For example, using the Hamming-metric SDP in GPS [37] requires a signature of size 24 kB to attain a security level of 128 bits. The corresponding R-SDP version requires only 14 kB signature sizes; R-SDP(G) needs just 12 kB, half the size for the same security.

We have considered several such ZK protocols and analyzed the performance of their R-SDP variants in terms of signature sizes, which are the main bottleneck of code-based signature schemes stemming from ZK protocols. We have found that the CROSS-ID protocol attains the best performance. This is mainly due to the large number of hashes required in GPS [37] and BG [19], which seem to be the bottleneck for computational efficiency.

Choice of Ambient Space. The R-SDP and the ZK protocols can easily be formulated over a general finite field $\mathbb{F}_q$, where q is a prime power. However, we restrict ourselves to prime fields to thwart possible vulnerabilities concerning subfield attacks [44].

7 Security

This section provides a thorough security analysis of CROSS.

- We provide the EUF-CMA security proof in Theorem 8. This is derived from [41], and the properties of the underlying ZK protocol, i.e., (2,2)-special-soundness and being a $q2$-Identification scheme.
- The hardness of the underlying problem is shown in Theorem 3. R-SDP is relatively new but closely related to the classical syndrome decoding problem and the subset sum problem, which have been studied extensively [40]. This allows us to tailor the best-known solvers of the related problems to our new setting. In Section 7.1 we give a conservative estimate of the time complexity of these algorithms. The estimation scripts are available at `https://www.cross-crypto.com/resources`.
- In Section 7.2 we discuss the possibility of solving R-SDP by Gröbner bases. These algebraic attacks are inferior to the combinatorial ones discussed in Section 7.1.
- We provide forgery attacks in Section 7.3.1 on weighted challenges by adapting the attack in [43].
- Finally, in Section 8, we show that the chosen set of parameters attains the claimed security levels, reporting the finite-regime security estimates.

7.1 Hardness of Underlying Problem and Generic Solvers

The fastest known generic solvers for the syndrome decoding problem in the Hamming metric are Information Set Decoding (ISD) algorithms [48, 28, 16, 13]. The best-known algorithms for the subset sum problem are introduced in [40, 12]. These solvers have been adapted to R-SDP in [23] for the particular case of $z \in \{2, 4, 6\}$ and in [8] for arbitrary values of z. These works have shown that the cost of solving the R-SDP depends on the particular structure of $\mathbb{E}$ and have identified weaker instances, such as small z, or even z.

We will quickly recall here the main points and which structures of $\mathbb{E}$ should be avoided. Indeed, several choices can lead to a somewhat easier problem. For instance, in an extension field $\mathbb{F}_{p^m}$ for some prime p and integer m, there are several choices of $\mathbb{E}$ where one can consider solving a simpler problem: $\mathbb{E} = \mathbb{F}_p$ is an obvious example. More generally, picking $\mathbb{E}$ contained in a relatively small subfield can lead to the vulnerability from [44]. To avoid this possibility, we restrict our consideration to prime fields.

As another suboptimal choice, one can choose rather large values for p and $\mathbb{E} = \{0, 1\}$. Thus, solvers for subset sum problems may be used [12], where one adds some elements to the search space. To circumvent possible speedups from such techniques, we restrict ourselves to error sets $\mathbb{E}$ of relatively large size.

We have therefore excluded these weaker instances and analyzed the decoding cost for our two instances:

1. R-SDP with $p = 127, z = 7$,
2. R-SDP(G) with $p = 509$ and $z = 127$.

We will quickly recall the famous Stern/Dumer algorithm and elaborate more on the representation technique approach (adapted from [13]) using larger search spaces (as in [12]) since this algorithm depends heavily on the additive structure of the chosen $\mathbb{E}$.

7.1.1 Generic Solvers for the R-SDP

To estimate the R-SDP's complexity, we provide combinatorial solvers based on the fastest known algorithms for solving the classical syndrome decoding problem [48, 13] and hard knapsacks [40, 12].

Let us quickly recall the partial Gaussian elimination step [35], which is used in all modern ISD algorithms. Given the parity-check matrix $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, an information set I is chosen and $\mathbf{H}$ is brought into quasi-systematic form. For this, let I' be a set of size $k+\ell$, which contains the information set I and transform $\mathbf{H}$ as

$$\mathbf{UHP} = \widetilde{\mathbf{H}} = \begin{pmatrix} \mathbf{I}_{n-k-\ell} & \mathbf{H}_1 \\ 0 & \mathbf{H}_2 \end{pmatrix},$$

where $\mathbf{U} \in \mathbb{F}_p^{(n-k)\times(n-k)}$ is an invertible matrix and $\mathbf{P} \in \mathbb{F}_p^{n\times n}$ is a permutation matrix. This inherently splits the unknown error vector $\mathbf{e}$ into the positions indexed by I' and $(I')^C$, i.e., $\mathbf{eP}^\top = (\mathbf{e}_1, \mathbf{e}_2)$. Thus, we get the system of two equations

$$\begin{aligned} \mathbf{e}_1 + \mathbf{e}_2\mathbf{H}_1^\top &= \mathbf{s}_1 \text{ and} \\ \mathbf{e}_2\mathbf{H}_2^\top &= \mathbf{s}_2, \end{aligned}$$

where $\mathbf{e}_1, \mathbf{e}_2$ are full-weight vectors with entries in $\mathbb{E}$ of length $n-(k+\ell)$ and $k+\ell$. To solve the system, one enumerates solutions $\mathbf{e}_2$ of the second equation $\mathbf{e}_2\mathbf{H}_2^\top = \mathbf{s}_2$ and checks for each one if the remaining $\mathbf{e}_1 = \mathbf{s}_1 - \mathbf{e}_2\mathbf{H}_1^\top$ completes it to a valid, i.e., restricted, solution. In the following, we discuss two advanced methods for the enumeration of $\mathbf{e}_2$.

Algorithm based on Collision Search We start with an algorithm that uses a meet-in-the-middle strategy to enumerate the solutions. This approach was applied to hard knapsacks by Horowitz and Sahni [39] and adopted for the syndrome decoding problem by Stern and Dumer [48, 28]. For the adaption to R-SDP, to which we refer to as *Stern/Dumer*, we define the lists

$$\begin{aligned} \mathcal{L} &\coloneqq \left\{ (\mathbf{x}_1, (\mathbf{x}_1, \mathbf{0})\mathbf{H}_2^\top) \mid \mathbf{x}_1 \in \mathbb{E}^{\left\lfloor \frac{k+\ell}{2} \right\rfloor} \right\} \text{ and} \\ \mathcal{L}' &\coloneqq \left\{ (\mathbf{x}_2, \mathbf{s}_2 - (\mathbf{0}, \mathbf{x}_2)\mathbf{H}_2^\top) \mid \mathbf{x}_2 \in \mathbb{E}^{\left\lceil \frac{k+\ell}{2} \right\rceil} \right\}, \end{aligned}$$

which contain $|\mathcal{L}| = z^{\left\lfloor \frac{k+\ell}{2} \right\rfloor}$ and $|\mathcal{L}'| = z^{\left\lceil \frac{k+\ell}{2} \right\rceil}$ elements, respectively. One uses a collision search to find suitable $\mathbf{e}_2 = (\mathbf{x}_1, \mathbf{x}_2)$ and extends them to solutions of the complete problem, as discussed in the partial Gaussian elimination step.

Theorem 9. The discussed collision-based solver *Stern/Dumer*, which is tailored to full-weight R-SDP, uses $M_{\text{Stern}}(p, n, k, z)$ bits of memory, which can be lower-bounded as

$$M_{\text{Stern}}(p, n, k, z) \geq |\mathcal{L}| \cdot \left\lfloor \frac{k+\ell}{2} \right\rfloor \cdot \log_2(z).$$

The number of binary operations of the collision-based algorithm tailored to full-weight R-SDP can be bounded from below as

$$C_{\text{Stern}}(p, n, k, z) \geq \min_{0\leq\ell\leq n-k} \frac{C + C' + C_{\text{coll}}}{1 + z^n p^{k-n}} \log_2(M_{\text{Stern}}(p, n, k, z)),$$

where C, C' and C_{coll} are bounded as

$$\begin{aligned} C &\geq |\mathcal{L}| \cdot \left(\left\lfloor \tfrac{k+\ell}{2} \right\rfloor \cdot \log_2(z) + \ell \cdot \log_2(p) \right), \\ C' &\geq |\mathcal{L}'| \cdot \left(\left\lceil \tfrac{k+\ell}{2} \right\rceil \cdot \log_2(z) + \ell \cdot \log_2(p) \right), \\ C_{\text{coll}} &\geq |\mathcal{L}| \cdot |\mathcal{L}'| \cdot p^{-\ell} \cdot (k+\ell) \log_2(p). \end{aligned}$$

Proof. To perform the collision search, the algorithm has to store the smaller list among $\mathcal{L}$ and $\mathcal{L}'$. Since this list contains dense vectors of length $\left\lfloor \frac{k+\ell}{2} \right\rfloor$ with entries in $\mathbb{E}$, at least $\left\lfloor \frac{k+\ell}{2} \right\rfloor \log_2(z)$ bits are required per list element. This gives the bound on the memory cost $M_{\text{Stern}}(p, n, k, z)$.

Let us consider the algorithm's time complexity $C_{\text{Stern}}(p, n, k, z)$. As usual, the complexity of finding any solution is given by the cost of finding a particular one divided by the number of solutions. Here, the average number of solutions is tightly upper-bounded as $1 + z^n p^{k-n}$.

When enumerating the solutions of the small instance, one must first store the error vectors $\mathbf{x}_1$ associated with list $\mathcal{L}$ in positions depending on the corresponding syndrome $\mathbf{x}_1\mathbf{H}_1^\top$. The error vectors have a size of $\frac{k+\ell}{2} \cdot \log_2(z)$ bits and the syndromes a size of $\ell \cdot \log_2(p)$ bits. Hence, this requires at least $|\mathcal{L}| \cdot \left(\left\lfloor \frac{k+\ell}{2} \right\rfloor \cdot \log_2(z) + \ell \cdot \log_2(p) \right)$ binary operations.

Next, the syndromes $\mathbf{s}_2 - \mathbf{x}_2\mathbf{H}_2^\top$ of the error vectors $\mathbf{x}_2$ associated with list $\mathcal{L}'$ are calculated. Again, due to the size of the objects, this requires at least $|\mathcal{L}'| \left(\left\lceil \frac{k+\ell}{2} \right\rceil \cdot \log_2(z) + \ell \cdot \log_2(p) \right)$ binary operations.

Solutions $\mathbf{e}_2$ of the small instance are obtained by performing a collision search, i.e., $\mathbf{x}_1\mathbf{H}_1^\top = \mathbf{s}_2 - \mathbf{x}_2\mathbf{H}_2^\top$. On average, $|\mathcal{L}| \cdot |\mathcal{L}'| \cdot p^{-\ell}$ collisions are found. For each collision, one checks whether $\mathbf{e}_2 = (\mathbf{x}_1, \mathbf{x}_2)$ extends to a solution $\mathbf{e}$ to the complete problem. For this, one has to calculate at least one syndrome symbol of the complete instance, which is the sum of $k + \ell$ elements of $\mathbb{F}_p$. Hence, this step requires at least $|\mathcal{L}| \cdot |\mathcal{L}'| \cdot p^{-\ell} \cdot (k + \ell) \log_2(p)$ binary operations.

Finally, the memory access cost is modeled with the conservative logarithmic cost model [29, 6], that is, the cost per iteration is increased by a factor $\log_2(M_{\text{Stern}}(p, n, k, z))$. □

Algorithm based on the Representation Technique We now analyze a more elaborate multi-level algorithm inspired by [40, 12, 13, 38]. This algorithm uses representations from a sum partition instead of the above set partition. That is one writes the solution to the smaller instance $\mathbf{e}_2 = \mathbf{e}^{(1)} + \mathbf{e}^{(2)}$, where $\mathbf{e}^{(1)}$ and $\mathbf{e}^{(2)}$ are suitably chosen vectors of length $k + \ell$, the supports of which may overlap. Then, there are multiple pairs $(\mathbf{e}^{(1)}, \mathbf{e}^{(2)})$, which follow a chosen distribution and sum to $\mathbf{e}_2$. Since it is sufficient to obtain a single copy of $\mathbf{e}_2$ to solve the problem, it is sufficient to enumerate only a fraction of all possible pairs $(\mathbf{e}^{(1)}, \mathbf{e}^{(2)})$.

For this section, we introduce the notation $\mathbb{E}_0$ to denote $\mathbb{E} \cup \{0\}$.

To minimize the number of vectors that must be enumerated and, hence, the computational complexity, we tailor the representation technique to the restricted case. That is, we do not only construct lists with $\mathbf{e}^{(i)} \in \mathbb{E}_0^{k+\ell}$, but in $(\mathbb{E}_0 \cup \mathbb{D})^{k+\ell}$, where $\mathbb{D} \subseteq \{a - b \mid a, b \in \mathbb{E}\}$ is a carefully chosen set, which allows for an increased number of representations. We denote by $z_{\mathbb{D}}$ the size of the chosen $\mathbb{D}$.

To determine the number of representations of an error vector as a sum of vectors in $(\mathbb{E}_0 \cup \mathbb{D})^{k+\ell}$, we quantify the additive structure of $\mathbb{E}$ and $\mathbb{D}$ in the following. For this, we determine the number of possibilities to write an element $a \in \mathbb{E}$ as $b + c$ with $b, c \in \mathbb{E}$ and the number of possibilities to write it as $b + c'$ with $b \in \mathbb{E}, c' \in \mathbb{D}$. These quantities are denoted by

$$\alpha_{\mathbb{E}}(a) \coloneqq |\{b \in \mathbb{E} \mid \exists c \in \mathbb{E} : b + c = a\}|,$$
$$\alpha_{\mathbb{D}}(a) \coloneqq |\{b \in \mathbb{E} \mid \exists c \in \mathbb{D} : b + c = a\}|.$$

Since for our choice of $\mathbb{E}$ these quantities do not depend on the choice of $a \in \mathbb{E}$, we simply write $\alpha_{\mathbb{E}}$, respectively $\alpha_{\mathbb{D}}$.

Example 10. For $g = 2$ of order $z = 7$ in $\mathbb{F}_{127}$, we have $\mathbb{E} = \{1, 2, 4, 8, 16, 32, 64\}$ and $\alpha_{\mathbb{E}} = 1$, since for any $2^i \in \mathbb{E}$ it holds that $2^{i-1} \in \mathbb{E}$ (and $2^{i-1} + 2^{i-1} = 2^i$). Further, we pick

$$\mathbb{D} = \{a - b \mid a, b \in \mathbb{E}\} \setminus \mathbb{E}_0 = \{2^{i_1} - 2^{i_1+i_2} \mid i_1 \in \{0, \ldots, 6\}, i_2 \in \{1, \ldots, 5\}\},$$

which contains $z_{\mathbb{D}} = 35$ elements. For any element $2^i \in \mathbb{E}$ there exists five elements $c \in \mathbb{D}$ such that $c + 2^i \in \mathbb{E}$, thus $\alpha_{\mathbb{D}} = 5$. More generally, for the chosen p, z, there is a $\widetilde{\mathbb{D}}$ of size $z_{\widetilde{\mathbb{D}}} = z \cdot s$ with $a \in \{1, \ldots, 5\}$ such that $\alpha_{\widetilde{\mathbb{D}}} = s$.

We have introduced all preliminaries for counting the number of representations, which is given in the following lemma.

Lemma 11. Let $\mathbf{e} \in (\mathbb{E}_0 \cup \mathbb{D})^{k+\ell}$ have v_i entries from $\mathbb{E}$ and d_i entries from $\mathbb{D}$. Further, we let $\nu_{i+1} = v_{i+1} - \frac{v_i}{2}$ and $\delta_{i+1} = d_{i+1} - \frac{d_i}{2}$. Then, there are

$$r = \binom{v_i}{v_{i+1}} \binom{v_{i+1}}{2\nu_{i+1}} \alpha_{\mathbb{E}}^{2\nu_{i+1}} \cdot \binom{v_i/2 - \nu_{i+1}}{\delta_{i+1}}^2 \cdot \alpha_{\mathbb{D}}^{2\delta_{i+1}} \binom{d_i}{d_i/2}$$

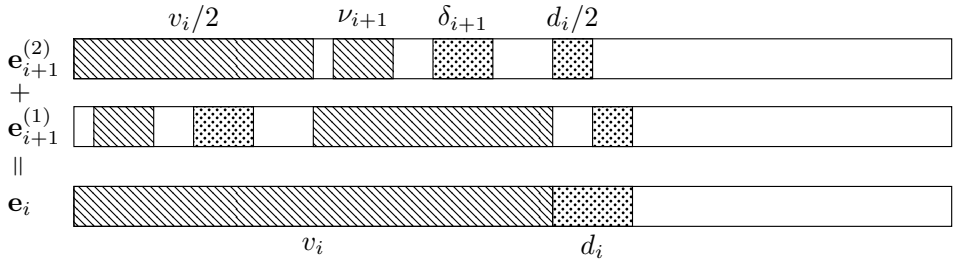

Figure 8: Counting the number of representations on level i.

possibilities for picking $\mathbf{e}^{(1)}, \mathbf{e}^{(2)} \in (\mathbb{E}_0 \cup \mathbb{D})^{k+\ell}$ such that $\mathbf{e}^{(1)}, \mathbf{e}^{(2)}$ each have v_{i+1} entries in $\mathbb{E}$, d_{i+1} entries in $\mathbb{D}$, and $\mathbf{e}^{(1)} + \mathbf{e}^{(2)} = \mathbf{e}$.

Proof. The counting of the number of representations is depicted in Figure 8. For the v_i entries of $\mathbf{e}$ living in $\mathbb{E}$, we choose $v_{i+1} = v_i/2 + \nu_{i+1}$ entries in $\mathbf{e}^{(1)}$ and distribute inside these v_{i+1} entries the $2\nu_{i+1}$ overlaps with entries of $\mathbf{e}^{(2)}$ in $\mathbb{E}$. By definition, there are $\alpha_{\mathbb{E}}^{2\nu_{i+1}}$ ways of choosing the $2\nu_{i+1}$ entries. Then, out of the non-selected $v_i - v_{i+1}$ entries of $\mathbf{e}$ in $\mathbb{E}$, we choose δ_{i+1} many entries of $\mathbf{e}^{(1)}$ for overlaps with entries in $\mathbf{e}^{(2)}$. This step is repeated for $\mathbf{e}^{(2)}$, since there is also the same choice for $\mathbf{e}^{(2)}$ to overlap with entries of $\mathbf{e}^{(1)}$ in $\mathbb{E}$. By definition, there are again $\alpha_{\mathbb{D}}^{2\delta_{i+1}}$ choices for these entries. Finally, we split the d_i entries of $\mathbf{e}$ living in $\mathbb{D}$ into $d_i/2$ entries of $\mathbf{e}^{(1)}$ living in $\mathbb{D}$, which then also fixes the remaining $d_i/2$ entries of $\mathbf{e}^{(2)}$ in $\mathbb{D}$.

□

We now describe how the multi-level algorithm proceeds in the case of four levels. We also tried more levels, however, increasing the number of levels further did not yield an improved finite regime performance. On level i, the solver uses list with v_i elements from $\mathbb{E}$ and d_i elements from $\mathbb{D}$. The compositions of the levels are connected via

$$\begin{array}{llll} v_0 = k + \ell, & v_1 = v_0/2 + \nu_1, & v_2 = v_1/2 + \nu_2, & v_3 = v_2/2, \\ d_0 = 0, & d_1 = d_0/2 + \delta_1, & d_2 = d_1/2 + \delta_2, & d_3 = d_2/2, \end{array}$$

where $\ell, \nu_1, \nu_2, \delta_1$ and δ_2 are internal parameters which can be optimized. The parameter ℓ denotes the redundancy of the small instance due to the partial Gaussian elimination, and ν_i and δ_i correspond to the "overlapping" number of entries in $\mathbb{E}$, respectively in $\mathbb{D}$ on level i.

Then, according to Lemma 11, the number of representations for level 1, i.e., r_1, and the number of representation for level 0, i.e., r_0, are given by

$$r_1 = \binom{v_1}{v_2}\binom{v_2}{2\nu_2}\alpha_{\mathbb{E}}^{2\nu_2}\binom{v_1 - v_2}{\delta_2}^2\alpha_{\mathbb{D}}^{2\delta_2}\binom{d_1}{d_1/2},$$
$$r_0 = \binom{v_0}{v_1}\binom{v_1}{2\nu_1}\alpha_{\mathbb{E}}^{2\nu_1}\binom{v_0 - v_1}{\delta_1}^2\alpha_{\mathbb{D}}^{2\delta_1}.$$

- On the third and last level, the algorithm prepares the base lists $\mathcal{L}_3$. The elements of the base lists are vectors of length $\frac{k+\ell}{2}$ which contain v_3 elements of $\mathbb{E}$ and d_3 elements of $\mathbb{D}$. The base lists have the same size, being
$$L_3 = \binom{(k+\ell)/2}{v_3, d_3} z^{v_3} z_{\mathbb{D}}^{d_3},$$
where $\binom{(k+\ell)/2}{v_3,d_3} = \binom{(k+\ell)/2}{v_3+d_3} \cdot \binom{v_3+d_3}{v_3}$ denotes the trinomial coefficient.
- On the second level, two base lists are merged into a list by performing a concatenation merge on ℓ_1 symbols. We refer to the resulting list as $\mathcal{L}_2$, which contains vectors of length $k + \ell$ with v_2 elements of $\mathbb{E}$ and d_2 elements of $\mathbb{D}$. The lists $\mathcal{L}_2$ have sizes
$$L_2 = \binom{k+\ell}{v_2, d_2} z^{v_2} z_{\mathbb{D}}^{d_2} p^{-\ell_1},$$
where $\ell_1 = \log_p(r_1)$ guarantees that one representation of the final solution in $\mathcal{L}_2$ survives the merge on average.

- On the first level, the algorithm creates lists by performing a representation merge of two level-2 lists on ℓ_0 syndrome symbols. We refer to the resulting list as $\mathcal{L}_1$, which contains vectors of length $k+\ell$ with v_1 elements of $\mathbb{E}$ and d_1 elements of $\mathbb{D}$. The lists $\mathcal{L}_1$ have size

$$L_1 = \binom{k+\ell}{v_1, d_1} z^{v_1} z_{\mathbb{D}}^{d_1} p^{-\ell_0-\ell_1},$$

  where $\ell_0 = \log_p(r_0) - \ell_1$ guarantees that one representation of the final solution in $\mathcal{L}_1$ survives the merge on average.

- On level 0, a final representation merge on the remaining $\ell - \ell_1 - \ell_0$ syndrome symbols gives a solution of the small instance, i.e., vectors $\mathbf{e}_2$ of length $k+\ell$ with entries solely from $\mathbb{E}$ that satisfy $\mathbf{e}_2\mathbf{H}_2^\top = \mathbf{s}_2$.

Theorem 12. The discussed representation-based solver *BJMM* tailored to R-SDP uses $M_{\text{BJMM}}(p,n,k,z)$ bits of memory, which can be lower-bounded as

$$M_{\text{BJMM}}(p,n,k,z) \geq \max_{i\in\{3,2,1\}} \{L_i(v_i \log_2(z) + d_i \log_2(z_{\mathbb{D}}))\}.$$

The computational complexity of the algorithm can be bounded from below as

$$C_{\text{BJMM}}(p,n,k,z) = \min_{\ell,\nu_1,\nu_2,\delta_1,\delta_2} \left\{ \frac{C_3 + C_2 + C_1 + C_0}{1 + z^n p^{k-n}} \log_2(M_{\text{BJMM}}(p,n,k,z)) \right\},$$

where C_i denotes the cost associated with level i, which are given as

$$\begin{aligned}
C_3 &\geq 2 \cdot L_3(\ell_1 \log_2(p) + v_3 \log_2(z) + d_3 \log_2(z_{\mathbb{D}})),\\
C_2 &\geq 2 \cdot L_2(\ell_0 \log_2(p) + v_2 \log_2(z) + d_2 \log_2(z_{\mathbb{D}})),\\
C_1 &\geq 2 \cdot L_2^2 p^{-\ell_0} \log_2(p),\\
C_0 &\geq L_1^2 p^{-(\ell-\ell_0-\ell_1)} \log_2(p).
\end{aligned}$$

Proof. To perform the collision search, the BJMM algorithm has to store at least one of the lists on levels 3, 2 and 1. Note that the final list does not require to be stored since as soon as we have found a solution to the smaller instance, we can check if it expands to a solution to the original problem. Therefore, the memory cost of the solver can be lower-bounded by the minimum size of these lists. On level 3, i.e., for the base lists $\mathcal{L}_3$, each element requires at least $(v_3 \log_2(z) + d_3 \log_2(z_{\mathbb{D}}))$ bits of memory. Similarly, each element of $\mathcal{L}_2$ requires at least $(v_2 \log_2(z) + d_2 \log_2(z_{\mathbb{D}}))$ bits, and each element of $\mathcal{L}_1$ requires at least $(v_1 \log_2(z) + d_1 \log_2(z_{\mathbb{D}}))$ bits. This gives the bound on the memory cost $M_{\text{BJMM}}(p,n,k,z)$.

Let us now consider the time complexity of the BJMM algorithm.

One begins on the third level by constructing base lists $\mathcal{L}_3$. Similar to Stern/Dumer, one has to construct at least two such lists to be able to perform the first concatenation merge. For each element, which has size of at least $(v_3 \log_2(z) + d_3 \log_2(z_{\mathbb{D}}))$ bits, one calculates as a partial syndrome in $\mathbb{F}_p^{\ell_1}$. This gives the lower bound on the cost C_3.

On the second, level one performs the concatenation merge, which on average results in $|\mathcal{L}_2| = |\mathcal{L}_3|^2 p^{-\ell_1}$ collisions. For each collision, one obtains an error vector, which has the size of at least $(v_2 \log_2(z) + d_2 \log_2(z_{\mathbb{D}}))$ bits, and calculates a partial syndrome in $\mathbb{F}_p^{\ell_0}$. Again, this step has to be performed at least twice to continue to lower levels. Hence, we obtain the bound on C_2.

On the first level, one performs a representation merge between lists $\mathcal{L}_2$ on the small instance ℓ_0 syndrome symbols. This representation merge yields on average $|\mathcal{L}_2|^2 p^{\ell_0}$ collisions. Taking into account early abort techniques [16], we conservatively estimate the cost per collision as a single field addition: at least one element of the error vectors has to be added to determine whether the sum of the vectors can be a well-formed solution, i.e., a restricted vector. Considering that this step needs to be performed twice, we obtain the lower bound on C_1.

One performs a final representation merge between two lists of level 1 on the remaining $\ell - \ell_0 - \ell_1$ syndrome symbols of the small instance, to find solutions for the small instance. This representation

merge yields on average $|\mathcal{L}_1|^2 p^{-(\ell-\ell_0-\ell_1)}$ collisions. Again, we conservatively estimate the cost per collision as a single field addition.

Finally, the memory access cost is modeled with the conservative logarithmic cost model [29, 6], that is, the cost per iteration is increased by a factor $\log_2(M_{\text{BJMM}}(p,n,k,z))$. □

Shifting $\mathbb{E}$ One can modify the R-SDP instance to be solved. For this, the restricted error vector $\mathbf{e} \in \mathbb{E}^n$ is shifted by a vector $\mathbf{x} \in \mathbb{F}_p^n$. Since the syndrome $\mathbf{s}_\mathbf{x}$ of $\mathbf{x}$ through $\mathbf{H}$ is known, the problem now becomes to find $(\mathbf{e}+\mathbf{x})\mathbf{H}^\top = \mathbf{s}+\mathbf{s}_\mathbf{x}$. For solvers, the case of $\mathbf{x} = (-x,\dots,-x)$ with $x \in \mathbb{E}$ is of interest, as this introduces more zeroes in the new solution. Hence, we consider this variant in the following. Let us denote $\mathbb{E}$ shifted into direction $x \in \mathbb{E}$ as

$$\mathbb{E}_x \coloneqq \{a - x \mid a \in \mathbb{E}\} \setminus \{0\}.$$

Then, after shifting by $\mathbf{x} = (-x,\dots,-x)$, the modified R-SDP instance asks to find the vector $\widetilde{\mathbf{e}} = \mathbf{e} + \mathbf{x} \in (\mathbb{E}_x \cup \{0\})^n$ such that $\widetilde{\mathbf{e}}\mathbf{H}^\top = \mathbf{s} + \mathbf{s}_\mathbf{x}$. This shifted instance can be solved using a slight modification of the Stern-like and the BJMM-like algorithm introduced above. Since the entries of $\mathbf{e}$ are picked independently, the Hamming weight of the modified instance follows a binomial distribution, i.e., we have

$$\Pr(\mathrm{wt}_H(\widetilde{\mathbf{e}}) = w) = \frac{\binom{n}{w}(z-1)^w}{z^n} \quad \forall w \in \{0,\dots,n\}.$$

In particular, the weight of $\widetilde{\mathbf{e}}_2$, i.e., the shifted error restricted to the small instance, is also binomially distributed. Therefore, it is sufficient to enumerate solutions of the small instance with weight v_0, where $0 \le v_0 \le k+\ell$, in order to succeed with probability

$$\Pr(\mathrm{wt}_H(\widetilde{\mathbf{e}}_2) = v_0) = \binom{k+\ell}{v_0}(z-1)^{v_0} z^{-k-\ell}.$$

The total cost of the solver is then the cost of a single iteration divided by the success probability. For the Stern-like solver, the required number of iterations compensates the decreased cost per iteration and thus shifting does not provide a speed up. Hence, the complexity of shifted Stern is again as in Theorem 9. A BJMM-like solver, however, can benefit from the zeros, since intermediate lists anyways use error vectors which are not of full weight. The complexity per iteration is given in Theorem 12. It only remains to analyse the structure of the shifted errors in $\mathbb{E}_x$ and the supplementary elements in $\mathbb{D}_x \subseteq \{b-a \mid a,b \in \mathbb{E}_x\} \setminus (\mathbb{E}_x \cup \{0\})$.

In the following, we perform this analysis for the particular case of $p = 127$ and $z = 7$, which are the parameters used in the R-SDP variant of CROSS.

Example 13. Consider again the the error set $\mathbb{E} = \{1,2,4,8,16,32,64\} \subset \mathbb{F}_{127}$ with size $z = 7$ and additivity $\alpha_\mathbb{E} = 1$. Shifting by $x = 1$, one creates error entries that are either zero or live in the modified error set

$$\mathbb{E}_1 = \{1,3,7,15,31,63\}.$$

In the given example, unlike its parent, the modified error set does not possess an additive structure. This holds for shifting by $x = 1$ and for any $x \in \mathbb{E}$.

Previous to shifting, $\mathbb{E}$ had a difference set $\mathbb{D}$ of size $z_\mathbb{D} = 35$ and additivity $\alpha_\mathbb{D} = 5$. Shifting preserves this additivity of $\mathbb{D}$. Hence, one can build from the $\mathbb{D}$ corresponding to $\mathbb{E}$ a $\mathbb{D}_x$ which fits $\mathbb{E}_x$. This is done by shifting the elements of $\mathbb{D}$ and neglecting those that would represent zeros (which are excluded from $\mathbb{E}_x$). In our case, we obtain $\mathbb{D}_x$ of size $z_{\mathbb{D}_x} = 30$ with $\alpha_{\mathbb{D}_x} = 5$.

The performances achieved by the Stern-like and the BJMM-like solvers are depicted in Figure 9. For the comparison, we have chosen $p = 127$ and $z = 7$, as is the case for the R-SDP version of CROSS. For fixed code rate $R \approx 0.59$, the lower bound on the number of required binary operations is plotted over the code length n. This choice of parameters gives a unique solution on average. We observe that the Stern-like algorithm performs better than BJMM but slightly worse than shifted BJMM. For $k = 76$ and thus $n = 127$, all algorithms require more than 2^{143} operations. Hence, these parameters achieve the security level of NIST I. Similarly, a cost of 2^{207}, which corresponds to NIST III, is achieved for $n = 187$ and a cost of 2^{272}, which corresponds to NIST V, is achieved for $n = 251$. Scripts for reproducing the figure are available at https://www.cross-crypto.com/resources.

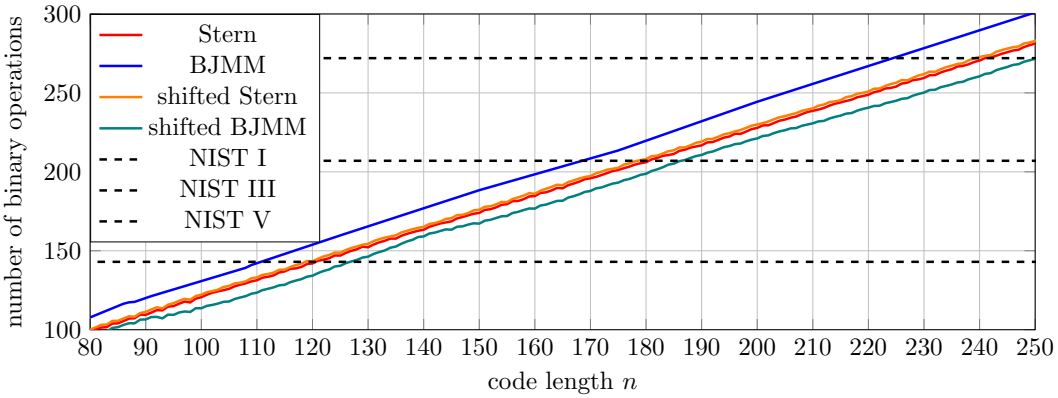

Figure 9: Comparison of the finite regime performance of the Stern-like algorithm and the BJMM-like algorithm. Depicted is the respective lower bound on the number of binary operations for $p = 127$, $z = 7$, $R \approx 0.59$ and varying code length n.

7.1.2 Generic Solvers for the R-SDP(G)

In this section, we extend the discussion of the computational hardness of R-SDP to R-SDP(G). For this, we analyze to which extent the knowledge of G, or equivalently $\mathbf{M}_G$, can be utilized by an attacker.

Not Considering G A first naive approach to solving R-SDP(G) would be to enumerate the solutions of the corresponding R-SDP, completely dismissing G. Then, for each solution $\mathbf{e} \in \mathbb{E}^n$ of the syndrome equation $\mathbf{eH}^\top = \mathbf{s}$, one can check validity, i.e., whether $\mathbf{e} \in G$. Since we dismiss G entirely, we are solving an instance where we have many more solutions than over G. In fact, the number of solutions over $\mathbb{E}$ is given by $z^n p^{k-n}$, out of which at most $z^m p^{k-n} + 1$ solve the original R-SDP(G) instance.

Example 14. For the parameter choices, $p = 509$, $z = 127$, $m = 25$ and $n = 55$, $k = 36$, we get $2^{213.5}$ solutions over $\mathbb{E}$, while we expect at most 15.7 solutions in G on average.

To enumerate the solutions of the parity-check equations in $\mathbb{E}^n$, one can use the algorithms described above or an adaption of Wagner's algorithm [52] to R-SDP. Wagner's algorithm is a multi-level variant of Stern's algorithm that is particularly efficient in finding solutions in regimes where multiple solutions exist. All of these algorithms do, however, require sorting and storing of lists that hold at least as many elements as solutions over $\mathbb{E}^n$. Hence, the enormous number of solutions, together with the memory access costs and further overheads, guarantee the security level.

As the discussion above shows, suitably chosen parameters guarantee that disregarding G leads to costs above the security levels. Hence, we now discuss methods that solve R-SDP(G) and use the knowledge of G.

Considering G A basic approach for this would be to go through all elements in G, of which there are z^m many. Then, for every element of $\mathbf{e} \in G$, one checks whether the parity-check equations are fulfilled. Clearly, the computational cost of such an attack can be easily pushed beyond the security level by choosing m large enough, i.e., $m > \lambda \log_z(2)$.

Therefore, we now consider a combination of the two mentioned approaches. In this hybrid approach, one reduces the number of solutions over $\mathbb{E}$, which are enumerated using the knowledge of G. Since a BJMM-like solver cannot utilize the structure of G in its sum partitions, we discuss a method for solving R-SDP(G) via a collision search.

Let us denote by $\mathbf{M}_H \in \mathbb{F}_z^{(n-m)\times n}$ a parity-check matrix of $\langle \mathbf{M}_G \rangle$. The attacker begins by searching for a public key with a subgroup G, for which $\langle \mathbf{M}_H \rangle$ contains two subcodes with the following properties.

- The first subcode is of dimension d_a and has a support J_a of size $j_a = |J_a|$. Hence, this first subcode is generated by $(\mathbf{0}\,\mathbf{B}_a\,\mathbf{0})$ with $\mathbf{B}_a \in \mathbb{F}_z^{d_a \times j_a}$.
- The second subcode is of dimension d_b and has a support J_b of size $j_b = |J_b|$. Hence, the second subcode is generated by $(\mathbf{0}\,\mathbf{B}_b\,\mathbf{0})$ with $\mathbf{B}_b \in \mathbb{F}_z^{d_b \times j_b}$.

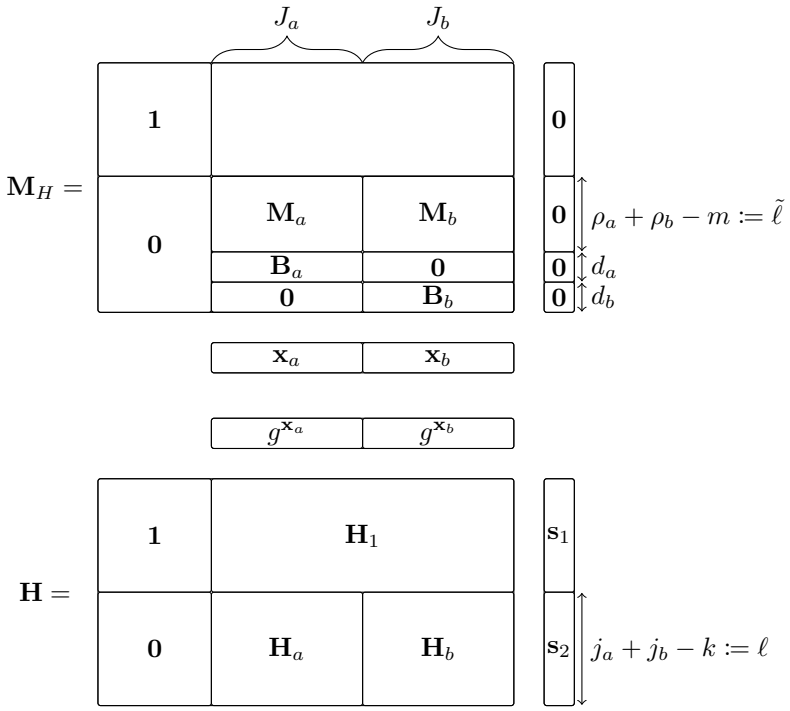

Figure 10: Illustration of Submatrix Stern/Dumer for R-SDP(G).

- The supports J_a and J_b are disjoint.

Due to the obvious connection to the codeword finding problem, the decisional version of this problem can be shown to be NP-complete itself [8, Theorem 2]. The best-known approach to solving the computational version is given by information set decoding, which can, however only succeed if such subcodes indeed exist for a given G. According to [47, Theorem 1], a subcode of dimension d and support size j exists with probability

$$P(j,d) \leq \min\left\{ \binom{n}{j} (z^d - 1)^{j-d} \begin{bmatrix} n-m \\ d \end{bmatrix}_z \begin{bmatrix} n \\ d \end{bmatrix}_z^{-1}, 1 \right\}.$$

Hence, the probability that both subcodes exist in $\langle \mathbf{M}_H \rangle$ can be upper-bounded as $P(j_a, d_a) \cdot P(j_b, d_b)$. It follows that $(P(j_a, d_a) \cdot P(j_b, d_b))^{-1}$ many groups G have to be considered on average to find one which allows for suitable subcodes.

Once the two subcodes are obtained, the attacker can use them in an improved solver, which is illustrated in Figure 10 and explained in the following. Using partial Gaussian elimination, one begins by bringing $\mathbf{M}_H$ and $\mathbf{H}$ into quasi-systematic form with respect to $J_a \cup J_b$. That is we obtain a smaller instance indexed by $I' = J_a \cup J_b$, which is of size $k + \ell = |J_a \cup J_b| = j_a + j_b$. Let us define the following two lists

$$\mathcal{L}_a := \left\{ (\mathbf{x}_a, \mathbf{x}_a \mathbf{M}_a^\top, g^{\mathbf{x}_a} \mathbf{H}_a^\top) \mid \mathbf{x}_a \in \ker(\mathbf{B}_a) \right\},$$
$$\mathcal{L}_b := \left\{ (\mathbf{x}_b, -\mathbf{x}_b \mathbf{M}_b^\top, \mathbf{s}_2 - g^{\mathbf{x}_b} \mathbf{H}_b^\top) \mid \mathbf{x}_b \in \ker(\mathbf{B}_b) \right\}.$$

These lists contain $\log_z(|\mathcal{L}_a|) = j_a - d_a =: \rho_a$ and $\log_z(|\mathcal{L}_b|) = j_b - d_b =: \rho_b$ elements.

In order to calculate the number of collisions between $\mathcal{L}_a$ and $\mathcal{L}_b$, let us have a closer look at the dimensions of the submatrices given in Figure 10. Since the width of the small instance is $j_a + j_b$, one can see that $\mathbf{H}_a \in \mathbb{F}_p^{(j_a + j_b - k) \times j_a}$ and $\mathbf{H}_b \in \mathbb{F}_p^{(j_a + j_b - k) \times j_b}$. We define, as usual, $\ell := j_a + j_b - k$ as the height of this small instance. For $\mathbf{M}_H$, the total height of the small instance is given by $j_a + j_b - m$. This implies that $\mathbf{M}_a$ and $\mathbf{M}_b$ are of height $j_a + j_b - m - d_a - d_b = \rho_a + \rho_b - m$. Let us define $\tilde{\ell} = \max\{0, \rho_a + \rho_b - m\}$. Then, each element of $\mathcal{L}_a$, respectively of $\mathcal{L}_b$, lives in $\mathbb{F}_z^{j_a} \times \mathbb{F}_z^{\tilde{\ell}} \times \mathbb{F}_p^{\ell}$, respectively in $\mathbb{F}_z^{j_b} \times \mathbb{F}_z^{\tilde{\ell}} \times \mathbb{F}_p^{\ell}$. Since the first part is an element of $\ker(\mathbf{B}_a)$, respectively of $\ker(\mathbf{B}_b)$, it can be represented using ρ_a, respectively ρ_b, symbols of $\mathbb{F}_z$. We refer to the combination of the second and third part as the *label* of the element. As the following statement shows, the labels can be used to find valid combinations

in $\mathcal{L}_a \times \mathcal{L}_b$: $(\mathbf{x}_a, \mathbf{x}_b) \in \ker(\mathbf{B}_a) \times \ker(\mathbf{B}_b)$ results in a solution of the small instance if and only if the corresponding labels match:

$$\begin{aligned} \mathbf{x}_a\mathbf{M}_a^\top = -\mathbf{x}_b\mathbf{M}_b^\top &\iff (\mathbf{x}_a, \mathbf{x}_b)(\mathbf{M}_a, \mathbf{M}_b)^\top = \mathbf{0}, \\ g^{\mathbf{x}_a}\mathbf{H}_a^\top = \mathbf{s}_2 - g^{\mathbf{x}_b}\mathbf{H}_b^\top &\iff (g^{\mathbf{x}_a}, g^{\mathbf{x}_b})(\mathbf{H}_a, \mathbf{H}_b)^\top = \mathbf{s}_2. \end{aligned}$$

Hence, solutions of the small instance can be found via a collision search which is performed on the labels of the list elements. Since the labels live in $\mathbb{F}_z^{\tilde{\ell}} \times \mathbb{F}_p^{\ell}$, this search yields on average

$$\frac{z^{\rho_a} \cdot z^{\rho_b}}{z^{\max(\rho_a+\rho_b-m,\,0)} \cdot p^{j_a+j_b-k}} \coloneqq \frac{z^{\rho_a} \cdot z^{\rho_b}}{z^{\tilde{\ell}} \cdot p^{\ell}}$$

collisions, which are extended to the complete instance.

Theorem 15. The discussed collision-based solver *Submatrix Stern/Dumer*, which is tailored to R-SDP(G), uses $M(p, n, k, z, m)$ bits of memory, which can be lower-bounded as

$$M(p, n, k, z, m) \geq \min\{|\mathcal{L}_a| \cdot \rho_a, |\mathcal{L}_b| \cdot \rho_b\} \cdot \log_2(z),$$

where $|\mathcal{L}_a| = z^{\rho_a}$ and $|\mathcal{L}_b| = z^{\rho_b}$. The number of binary operations can be bounded from below as

$$C(p, n, k, z, m) \geq \min_{J_a, J_b} \left\{ \frac{C_a + C_b + C_{\text{coll}}}{1 + z^m p^{k-n}} \log_2\left(M(p, n, k, z, m)\right) + \frac{1}{P(j_a, d_a) \cdot P(j_b, d_b)} \right\},$$

where C_a, C_b and C_{coll} are bounded as

$$\begin{aligned} C_a &\geq |\mathcal{L}_a| \cdot \left(\rho_a \cdot \log_2(z) + \tilde{\ell} \cdot \log_2(z) + \ell \cdot \log_2(p)\right), \\ C_b &\geq |\mathcal{L}_b| \cdot \left(\rho_b \cdot \log_2(z) + \tilde{\ell} \cdot \log_2(z) + \ell \cdot \log_2(p)\right), \\ C_{\text{coll}} &\geq |\mathcal{L}_a| \cdot |\mathcal{L}_b| \cdot z^{-\tilde{\ell}} \cdot p^{-\ell}(j_a + j_b) \log_2(p), \end{aligned}$$

with $\ell = j_a + j_b - k$ and $\tilde{\ell} = \max\{0, \rho_a + \rho_b - m\}$.

Proof. The cost is obtained in a similar way as the cost of the standard R-SDP Stern/Dumer algorithm given in Theorem 9. We state it for completeness.

To perform the collision search, the algorithm has to store the smaller list among $\mathcal{L}_a$ and $\mathcal{L}_b$. Without loss of generality, we assume in the following that this is $\mathcal{L}_a$. Since this list contains elements of $\ker(\mathbf{B}_a)$, at least $\rho_a \log_2(z)$ bits are required per list element. This gives the bound on the memory cost $M(p, n, k, z)$.

Let us consider the algorithm's time complexity $C(p, n, k, z)$. As usual, the complexity of finding any solution is given by the cost of finding a particular one divided by the number of solutions. Here, the average number of solutions is tightly upper-bounded as $1 + z^m p^{k-n}$.

When enumerating the solutions of the small instance, one must first store the vectors $\mathbf{x}_a$ associated with list $\mathcal{L}_a$ in positions depending on the corresponding label $(\mathbf{x}_a\mathbf{M}_a^\top, g^{\mathbf{x}_a}\mathbf{H}_a^\top)$. The vectors can be represented using $\rho_a \cdot \log_2(z)$ bits and the label has a size of $\tilde{\ell} \cdot \log_2(z) + \ell \cdot \log_2(p)$ bits with $\ell = j_a + j_b - k$ and $\tilde{\ell} = \max\{0, \rho_a + \rho_b - m\}$. Hence, this requires at least

$$|\mathcal{L}_a| \cdot \left(\rho_a \cdot \log_2(z) + \tilde{\ell} \cdot \log_2(z) + \ell \cdot \log_2(p)\right)$$

binary operations.

Next, the labels $(-\mathbf{x}_b\mathbf{M}_b^\top, \mathbf{s}_2 - g^{\mathbf{x}_b}\mathbf{H}_b^\top)$ of the of the elements in list $\mathcal{L}_b$ are calculated. The required number of binary operations is computed as

$$|\mathcal{L}_b| \left(\rho_b \cdot \log_2(z) + \tilde{\ell} \cdot \log_2(z) + \ell \cdot \log_2(p)\right),$$

which results from the size of the involved objects.

Solutions $\mathbf{e}_2$ of the small instance are obtained by performing a collision search. On average, $|\mathcal{L}_a| \cdot |\mathcal{L}_b| \cdot z^{-\tilde{\ell}} p^{-\ell}$ collisions are found. For each collision, one checks whether $\mathbf{e}_2 = (g^{\mathbf{x}_a}, g^{\mathbf{x}_b})$ extends to a solution $\mathbf{e}$ to the original problem. For this, one has to calculate at least one syndrome symbol of the original instance, which is the sum of $j_a + j_b$ elements of $\mathbb{F}_p$. Hence, this step requires at least

$$|\mathcal{L}_a| \cdot |\mathcal{L}_b| \cdot z^{-\tilde{\ell}} \cdot p^{-\ell} \cdot (j_a + j_b) \log_2(p)$$

binary operations.

Finally, the memory access cost is modeled with the conservative logarithmic cost model [29, 6], that is, the cost per iteration is increased by a factor $\log_2(M(p, n, k, z))$. □

Example 16. For the parameter choices, $p = 509, z = 127, m = 25, n = 55$ and $k = 36$, a one-dimensional subcode with support size 19 exists with probability 0.45. Further, we assume that the attacker uses a four-dimensional subcode with support size 23 which exists with probability $2^{-116.9}$. Then, the algorithm given in Theorem 15 requires at least $2^{143.6}$ binary operations and a memory of $2^{132.7}$ bit.

7.2 Gröbner Basis Approach

Recall that in the R-SDP, Problem 2, given $g \in \mathbb{F}_p^*$ of prime order z, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $\mathbf{s} \in \mathbb{F}_p^{n-k}$, and $\mathbb{E} = \{g^i \mid i \in \{1, \ldots, z\}\} \subset \mathbb{F}_p^*$, one aims at deciding whether there exists $\mathbf{e} \in \mathbb{E}^n$ such that $\mathbf{e}\mathbf{H}^\top = \mathbf{s}$. We look at the complexity of solving the R-SDP using Gröbner basis methods.

The R-SDP is equivalent to deciding whether the system

$$\begin{cases} \mathbf{x}\mathbf{H}^\top = \mathbf{s} \\ x_i^z = 1 \qquad \text{for } i \in \{1, \ldots, n\} \end{cases}$$

has a solution, since the equations $x_i^z = 1$ for $i \in \{1, \ldots, n\}$ force any potential solution to belong to the set $\mathbb{E}^n$.

Gröbner bases can always be used to solve systems of equations, but are rarely effective. We consider the F5 Algorithm [9] to compute Gröbner bases. The complexity of F5 for homogeneous polynomials is

$$O\left((n-k) d_{reg} \binom{n + d_{reg} - 1}{d_{reg}}^{\omega}\right) \sim O\left(\binom{n + d_{reg}}{d_{reg}}^{\omega}\right)$$

where d_{reg} is the degree of regularity and ω the exponent in the complexity of matrix multiplication.

Some recent lines of research (see [22, 25, 36]) focus on the complexity of d_{reg} concerning similar systems of random linear equations. These results show that d_{reg} growth is linear in n. This leads us to assume that d_{reg} is also linear in n for our systems.

This leads to the complexity of computing the Gröbner bases for the system to be exponential, since

$$\binom{n + d_{reg}}{d_{reg}}^{\omega} = 2^{n \cdot (1+c) \cdot h_2((1+c)^{-1}) \cdot \omega + o(n)},$$

where we used a standard approximation for the binomial coefficient and $c > 0$ comes from the linear growth of d_{reg}. Already small values of c imply that the complexity of computing the Gröbner bases exceeds the cost of the presented combinatorial solvers.

Some experimental results back up this line of theory. Given the parameters $p = 127$, $z = 7$, $k = \frac{1}{2}n$, we obtain Table 2. The degree of regularity d_{reg} and the CPU Time data in seconds have been computed using MAGMA [24]. Both computations were run on a single core of an Apple M1 Pro processor. Performing an exponential fitting, we obtain the plot in Figure 11 with the vertical axis represented in log plot. The data is shown in blue, and is clearly exponential in nature, while the red interpolation takes the form $5.83 \times 10^{-8} e^{1.93n}$. With a coefficient of determination R^2 equal to 1.000, we can say that the exponential interpolation represents the data almost perfectly.

Due to this exponential growth in time required, we anticipate that an adversary using Gröbner bases to attack R-SDP will face an insurmountable computational obstacle at practical parameters.

n	d_{reg}	**CPU time (s)**
4	13	0.000
6	14	0.007
8	16	0.242
10	18	13.171
12	19	625.829

Table 2: Gröbner basis computation data.

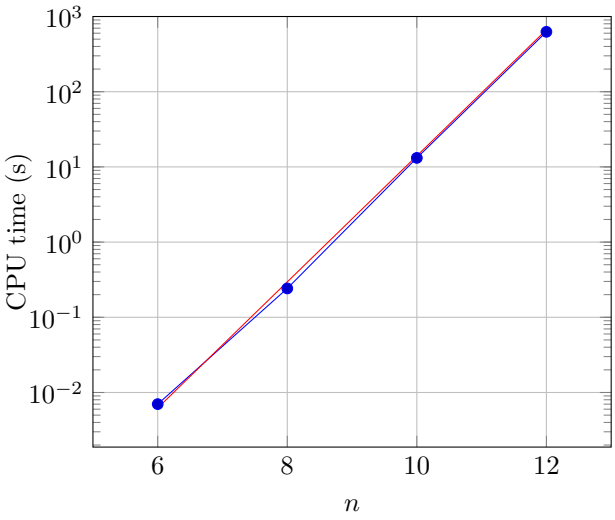

Figure 11: Exponential fitting of data.

7.3 Analysis of the Algorithm with Respect to Known Attacks

We have shown that CROSS achieves EUF-CMA security in Theorem 8 following the approach of [41]. We have seen in Section 7.1 that the underlying problems R-SDP and R-SDP(G) are NP-hard and provide the cost of generic decoders.

We point out that publishing the generators $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{E}^n$ for the subgroup G does not give rise to algebraic attacks. In fact, algebraic attacks that exploit the small order of the entries of $\mathbf{e}$ cannot be mounted straightforwardly, as the multiplicative structure of $\mathbb{E}$ is incompatible with the additive linearity of the syndrome computation.

For the following, we present two forgery attacks, adapted from [43]. The former is well known for weighted challenges, while the latter is a new attack. We will adapt our parameters considering these attacks.

7.3.1 Forgery Attacks

In this section, we describe two forgeries. We conservatively estimate the cost of these forgeries in terms of CROSS operations. In our analysis, one elementary operation corresponds to simulating several of the instructions that the prover would perform. As we argue in Section 8, this allows us to easily (and conservatively) assess the cost of such attacks so that the recommended CROSS instances meet the NIST security categories.

The first forgery we describe is relatively intuitive and consists of applying (for each round) one of the strategies we described in the proof of Proposition 7. The forgery is successful if, for all rounds, either β or b (or both) are guessed correctly; the cost of this attack is derived in the following Proposition.

Proposition 17. We consider the forgery that, for each round, first guesses both $\beta^{(i)}$ and $b^{(i)}$, then follows the cheating Strategy 0 and 1 defined in the proof for Proposition 7. For the second challenges

$b^{(1)}, \ldots, b^{(t)}$, this attack runs in average time $O\left(\frac{1}{P(t,w,p)}\right)$, where

$$P(t,w,p) = \sum_{w'=0}^{\min\{w,t-w\}} \frac{\binom{w}{w'}\binom{t-w}{w'}}{\binom{t}{w}} \left(\frac{1}{p-1}\right)^{2w'}.$$

Proof. The forgery is successful if all rounds are accepted, that is, if for each round either $\beta^{(i)}$ or $b^{(i)}$ (or both) have been guessed correctly. The average number of tests is given by the reciprocal of the probability that, for each round, both challenge values are correctly guessed. Conservatively, we do not consider the cost of each test. Still, we lower bound the cost of the forgery by using the average number of tests before the adversary's guesses are valid for each round. Let us consider t rounds and a fixed weight challenge $\mathbf{b} = \left(b^{(1)}, \ldots, b^{(t)}\right)$ the vector of challenges, with $b^{(i)} \in \{0;1\}$ being the challenge for the i-th round. When $\mathbf{b}$ has weight w, there will exist w many rounds with $b^{(i)} = 1$ and $t-w$ many rounds with $b^{(i)} = 0$. The adversary can now select the w rounds of $b^{(i)} = 1$. If the adversary chooses a challenge in one round correctly, this round will be accepted. However, if the adversary picks a challenge wrong, there is still the possibility of having chosen β correctly. Thus, let us assume that w' many rounds out of the w guessed $b = 1$-rounds are wrong, for $w' \in \{0, \ldots, \min\{w, t-w\}\}$. Then this also implies that in the guessed $b = 0$-rounds, there are w' mistakes, which means that in $2w'$ rounds, the adversary had to have guessed β correctly. This gives an overall cheating probability of

$$\sum_{w'=0}^{\min\{w,t-w\}} \frac{\binom{w}{w'}\binom{t-w}{w'}}{\binom{t}{w}} \left(\frac{1}{p-1}\right)^{2w'}.$$

□

We now describe another forgery inspired by the attack in [43] to 5-pass schemes. The attack makes use of the fact that the second challenge is generated after the first challenge, and, furthermore, it is possible to generate multiple second challenges without modifying the commitments or the first challenge value. This way, one can split the forgery into two separate phases, where the overall cost is given by the sum of the two associated costs.

Proposition 18. We consider the following procedure:

1) sample $\mathtt{Salt} \xleftarrow{\$} \{0;1\}^{2\lambda}$, $\mathtt{MSeed} \xleftarrow{\$} \{0;1\}^{\lambda}$, generate seeds $\mathtt{Seed}^{(1)}, \ldots, \mathtt{Seed}^{(t)}$ using the PRNG tree;

2) guess values $\widetilde{\beta}^{(1)}, \ldots, \widetilde{\beta}^{(t)}$ for the first challenge;

3) guess values $\widetilde{b}^{(1)}, \ldots, \widetilde{b}^{(t)}$ for the second challenge (consider that $\left(\widetilde{b}^{(1)}, \cdots, \widetilde{b}^{(t)}\right)$ has weight w);

4) for each $i = 1, \ldots, t$, do:

   4.1) sample $\mathbf{u}'^{(i)} \in \mathbb{F}_p^n$ and $\mathbf{e}'^{(i)} \in G$ using $\mathtt{Seed}^{(i)}$;

   4.2) choose an arbitrary $\sigma^{(i)} \in G$;

   4.3) compute $\mathbf{y}^{*(i)} = \mathbf{u}'^{(i)} + \widetilde{\beta}^{(i)}\mathbf{e}'^{(i)}$;

   4.4) compute $\widetilde{\mathbf{s}}^{(i)} = \sigma^{(i)}(\mathbf{y}^{*(i)})\mathbf{H}^{\top}$;

   4.5) set $c_0^{(i)} = \mathsf{Hash}\left(\widetilde{\mathbf{s}}^{(i)} - \widetilde{\beta}^{(i)}\mathbf{s}, \sigma^{(i)}, \mathtt{Salt}, i\right)$;

   4.6) set $c_1^{(i)} = \mathsf{Hash}(\mathbf{u}'^{(i)}, \mathbf{e}'^{(i)}, \mathtt{Salt}, i)$;

5) compute c_0 as the root of the tree $\mathsf{MerkleTree}\left(c_0^{(1)}, \ldots, c_0^{(t)}\right)$ and $c_1 = \mathsf{Hash}\left(c_1^{(1)}, \ldots, c_1^{(t)}\right)$; generate $\left(\beta^{(1)}, \ldots, \beta^{(t)}\right) = \mathsf{GenCh}_1(c_0, c_1, \mathtt{Msg}, \mathtt{Salt})$;

6) let $S = \left\{ i \in \{1, \cdots, t\} \,\middle|\, \widetilde{\beta}^{(i)} = \beta^{(i)} \right\}$; if $|S| \geq t^*$, proceed. Otherwise, restart from step 1);

7) for each round $i \in S$ (i.e., such that $\beta^{(i)} = \widetilde{\beta}^{(i)}$), set $\mathbf{y}^{(i)} = \mathbf{y}^{*(i)}$ and $h^{(i)} = \mathsf{Hash}(\mathbf{y}^{(i)})$;

8) for each round $i \notin S$ (i.e., such that $\beta^{(i)} \neq \widetilde{\beta}^{(i)}$), do:

8.1) if $\widetilde{b}^{(i)} = 0$:

S0.1) choose $\widetilde{\mathbf{e}}^{(i)} \in \mathbb{F}_p^n$ such that $\widetilde{\mathbf{e}}^{(i)}\mathbf{H}^\top = \mathbf{s}$;

S0.2) choose $\widetilde{\mathbf{u}}^{(i)} \in \mathbb{F}_p^n$ such that $\widetilde{\mathbf{u}}^{(i)}\mathbf{H}^\top = \sigma(\mathbf{y}^{*(i)})\mathbf{H}^\top - \beta^{(i)}\mathbf{s}$;

S0.3) set $\mathbf{y}^{(i)} = \widetilde{\mathbf{u}}^{(i)} + \beta^{(i)}\widetilde{\mathbf{e}}^{(i)}$ and $h^{(i)} = \mathsf{Hash}(\mathbf{y}^{(i)})$;

8.2) if $\widetilde{b}^{(i)} = 1$:

S1.1) set $\mathbf{y}^{(i)} = \mathbf{y}^{*(i)}$ and $h^{(i)} = \mathsf{Hash}(\mathbf{y}^{(i)})$;

9) compute h and generate $\left(b^{(1)}, \ldots, b^{(t)}\right) = \mathsf{GenCh}_2(c_0, c_1, \beta^{(1)}, \ldots, \beta^{(2)}, h, \mathtt{Msg}, \mathtt{Salt})$;

10) if $\left(b^{(1)}, \ldots, b^{(t)}\right)$ and $\left(\widetilde{b}^{(1)}, \ldots, \widetilde{b}^{(t)}\right)$ are equal for all indices $i \notin S$, proceed. Otherwise, restart from step 8);

11) for each $i = 1, \ldots, t$: if $b^{(i)} = 0$, set $f^{(i)} = \{\sigma^{(i)}, \mathbf{y}^{(i)}\}$, otherwise set $f^{(i)} = \mathtt{Seed}^{(i)}$.

The above algorithm is a forgery running on average time

$$O\left(\min_{t^* \in \{0,\ldots,t\}} \left\{ \frac{1}{P_\beta(t, t^*, p)} + \frac{1}{P_b(t, t^*, w, p)} \right\}\right),$$

where

$$P_\beta(t, t^*, p) = \sum_{j=t^*}^{t} \binom{t}{j} \left(\frac{1}{p-1}\right)^j \left(1 - \frac{1}{p-1}\right)^{t-j},$$

$$P_b(t, t^*, w, p) = \sum_{j=t^*}^{t} \frac{\binom{t}{j}\left(\frac{1}{p-1}\right)^j \left(1 - \frac{1}{p-1}\right)^{t-j}}{P_\beta(t, t^*, p)} \sum_{w^*=0}^{\min\{j,w\}} \frac{\binom{j}{w^*}^2 \binom{t-j}{w-w^*}}{\binom{t}{w}^2}.$$

Proof. The strategies followed by the forgery are, essentially, a rewriting of the ones in the proof for Proposition 7. The algorithm iterates over the first loop (steps 1–6) until the choices on the first challenge are valid for at least t^* rounds. Once this is obtained, the algorithm freezes the commitments, and the first challenge then starts making attempts until the second challenge is correctly generated. This is the purpose of steps 7–10. For each attempt, the algorithm uses fresh values $\widetilde{\mathbf{e}}^{(i)}$: this leads to a different h and, consequently, to new values for the second challenge $\left(b^{(1)}, \ldots, b^{(t)}\right)$. By doing this, the commitments prepared in the initial loop remain valid. This procedure gets repeated until the second challenge amends the situation; namely, in every round where the attacker did not guess the correct value for the first challenge, the value for the second challenge must be correct.

The total cost of the attack is the sum of the costs for the two phases. The probability that the initial guess $\left(\widetilde{\beta}^{(1)}, \ldots, \widetilde{\beta}^{(t)}\right)$ is valid, i.e., that it matches in at least t^* positions with $\left(\beta^{(1)}, \ldots, \beta^{(t)}\right)$, is

$$P_\beta(t, t^*, p) = \sum_{j=t^*}^{t} \binom{t}{j} \left(\frac{1}{p-1}\right)^j \left(1 - \frac{1}{p-1}\right)^{t-j}.$$

Consequently, the average cost for the first loop is $O\left(\frac{1}{P_\beta(t,t^*,p)}\right)$.

We now consider the second loop. Let S denote the set of indices i for which $\beta^{(i)} = \widetilde{\beta}^{(i)}$ and its complement by S^C. In the following, we will indicate $j = |S|$; notice that[3]

$$\Pr[|S| = j] = \frac{\binom{t}{j}\left(\frac{1}{p-1}\right)^j \left(1 - \frac{1}{p-1}\right)^{t-j}}{P_\beta(t, t^*, p)}$$

Let $\mathbf{b} = \left(b^{(1)}, \ldots, b^{(t)}\right)$ and $\widetilde{\mathbf{b}} = \left(\widetilde{b}^{(1)}, \ldots, \widetilde{b}^{(t)}\right)$, and indicate by $\mathbf{b}_S$ (resp., $\widetilde{\mathbf{b}}_S$) the vector formed by the coordinates of $\mathbf{b}$ which are indexed by S (resp., $\widetilde{\mathbf{b}}_S$). Analogously, we denote by $\mathbf{b}_{S^C}$ (resp., $\widetilde{\mathbf{b}}_{S^C}$) the vector formed by the coordinates of $\mathbf{b}$ (resp., $\widetilde{\mathbf{b}}$) which are not indexed by S. For the second loop to

[3] Formally this is the conditional probability, given that $|S| \geq t^*$; to avoid burdening the (already involved) notation, we do not indicate it explicitly.

halt, $\mathbf{b}$ must be such that $\mathbf{b}_{S^C} = \widetilde{\mathbf{b}}_{S^C}$. Let w^* denote the number of 1-guesses for the rounds indexed by S; that is, w^* is the Hamming weight of $\widetilde{\mathbf{b}}_S$. Notice that

$$\Pr\left[\mathrm{wt}(\widetilde{\mathbf{b}}_S) = w^*\right] = \frac{\binom{j}{w^*}\binom{t-j}{w-w^*}}{\binom{t}{w}}.$$

The probability that a generated $\mathbf{b}$ is valid, i.e., $\mathbf{b}_{S^C} = \widetilde{\mathbf{b}}_{S^C}$, is

$$\begin{aligned}\Pr\left[\mathbf{b} \text{ is valid} \,\middle|\, \mathrm{wt}(\widetilde{\mathbf{b}}_S) = w^*\right] &= \frac{\left|\left\{\mathbf{b} \in \{0,1\}^t \,\middle|\, \mathrm{wt}(\mathbf{b}) = w,\ \mathbf{b}_{S^C} = \widetilde{\mathbf{b}}_{S^C}\right\}\right|}{\binom{t}{w}} \\ &= \frac{\left|\left\{\mathbf{b} \in \{0,1\}^t \,\middle|\, \mathrm{wt}(\mathbf{b}_S) = w^*,\ \mathbf{b}_{S^C} = \widetilde{\mathbf{b}}_{S^C}\right\}\right|}{\binom{t}{w}} \\ &= \frac{\binom{j}{w^*}}{\binom{t}{w}}.\end{aligned}$$

An example of a valid $\mathbf{b}$ is reported below, for example, with $t^* = 5$ and $w^* = 3$.

	First t^* rounds					Last $t - t^*$ rounds							
Guessed $\widetilde{\beta}^{(i)}$	2	71	16	23	4	5	98	121	46	29	82	$\cdots$	45
Guessed $\widetilde{b}^{(i)}$	0	1	0	1	1	1	1	0	1	1	1	$\cdots$	0
Actual $\beta^{(i)}$	2	71	16	23	4	7	120	99	21	7	124	$\cdots$	3
Actual $b^{(i)}$	1	1	0	0	1	1	1	0	1	1	1	$\cdots$	0
	Weight w^*					Weight $w - w^*$							

Putting everything together, we have that a test for $\mathbf{b}$ is valid with an average probability

$$\begin{aligned}P_b(t, t^*, w, p) &= \sum_{j=t^*}^{t} \Pr[|S| = j] \cdot \sum_{w^*=0}^{\min\{j,w\}} \Pr[\mathrm{wt}(\widetilde{\mathbf{b}}_S) = w^*] \cdot \Pr\left[\mathbf{b} \text{ is valid} \,\middle|\, \mathrm{wt}(\widetilde{\mathbf{b}}_S = w^*)\right] \\ &= \sum_{j=t^*}^{t} \frac{\binom{t}{j}\left(\frac{1}{p-1}\right)^j\left(1 - \frac{1}{p-1}\right)^{t-j}}{P_\beta(t, t^*, p)} \sum_{w^*=0}^{\min\{j,w\}} \frac{\binom{j}{w^*}^2\binom{t-j}{w-w^*}}{\binom{t}{w}^2}.\end{aligned}$$

The overall cost of the attack is estimated by summing the costs for both phases and optimizing over t^*, that is

$$\min_{t^* \in \{0,\ldots,t\}} \left\{\frac{1}{P_\beta(t, t^*, p)} + \frac{1}{P_b(t, t^*, w, p)}\right\}.$$

□

8 Parameters and Description of Expected Security Strength

Our first concern for the parameter choices is with regard to the security of the system. Following that, we aimed to select values for the field size p, the number of elements in $\mathbb{E}^n$ (i.e. z^n), and the number of elements in G (i.e. z^m), such that their arithmetic is efficient. These parameters also had to take into account the trade-off between signature size and speed. To this end, the parameter-finding process was split into two phases:

i) choose code and restriction parameters, n, k, p and z, m, respectively, so that the best R-SDP/R-SDP(G) solvers require a computational effort matching the one to break AES with a 128, 192, or 256-bit keys (code available at https://www.cross-crypto.com/resources) and

ii) determine the optimal length t and weight w of the fixed-weight challenge vector $\mathbf{b}$.

The first phase of the parameter selection process was comprised of searching for all prime values p satisfying $17 \leq p \leq 2,477$ for all the admissible sizes of $\mathbb{E} \subset \mathbb{F}_p^*$ with z prime and code rates R in $\{0.3, 0.35, 0.4, \ldots, 0.7\}$, such that the smallest value of n solving R-SDP/R-SDP(G) with optimal m on a k-dimensional code in $\mathbb{F}_p^n$ with rate R exceeds 2^λ. This employs the computational complexities reported in Section 7.1. The second phase of the parameter selection involved computing for each value of t between λ and $1,100$, the set of (t, w) pairs such that the complexity of the best forgery attack against CROSS exceeds λ, and, for each value w, the corresponding value of t is minimum. The reason for this choice of an upper limit for t follows from striving to limit the amount of repetitions of the CROSS-ID protocol, to preserve computational efficiency. This exploration resulted in the exhaustive computation of large sets of parameter tuples, one for each of the NIST security categories. Each set contained parameter tuples which are equivalent to the security standpoint, both when considering attacks against the underlying R-SDP/R-SDP(G), and when considering forgery attacks through guessing.

We proceeded then pruning the parameter sets according to efficiency considerations. Given the structurally small size of the CROSS private and public keys (a single seed, and a seed plus a syndrome, respectively, as described in Section 5), we selected the signature size as the main space parameter, to balance the trade-offs. Indeed, the common alternative of considering signature and public key sizes does not alter the final results. We consider in this phase, as a proxy of the execution time, the number of rounds t; essentially, both the signature and verification time in CROSS are proportional to it, albeit through different multiplicative factors.

The first steps in pruning are to consider the entire parameter tuple set for each NIST category and evaluate the impact of the base field choice on the final signature size.

We employed $p = 127$ and $z = 7$ for the R-SDP variant of CROSS. While incurring a slight penalty to signature size, this choice allows us to have much more efficient arithmetic in modulo p and modulo z. Furthermore, all values of both $\mathbb{F}_p$ and $\mathbb{F}_z$ are efficiently representable within a single byte. Indeed both primes are Mersenne primes, which allows for an efficient modular reduction without the use of a divisor functional unit. For the R-SDP(G) variant of CROSS, we could not employ the same arithmetic choice due to the value of z being too small. We therefore have selected $p = 509$, as $\mathbb{F}_p^*$ admits a subgroup with cardinality $z = 127$, which in turn allows us to employ efficient Mersenne arithmetic in one of the two fields over which we compute.

Finally, we proceed to the pruning of the parameter sets by the code rate and number of rounds. Concerning the trade-off between speed and signature size, there is a phenomenon of diminishing returns in increasing the number of iterations t to reduce the signature size. The final outcome of the parameter selection procedure is the set of parameters reported in Table 3. For each NIST security category, we propose three parameter sets, selecting three optimization corners: computational speed in the signature and verification procedures, a balanced version which aims for stability, and signature size. We also report, for ease of comparison, the figures from the SPHINCS$^+$ signature scheme, which NIST has selected for standardization, and has been indicated as a bar for comparisons. Our proposed parameter sets obtain smaller keypairs than SPHINCS$^+$, although with a slightly larger public key. However, the public key size never exceeds 121 bytes, and never exceeds the length of an equivalent (pre-quantum) security level RSA keypair. Our signature sizes for CROSS-R-SDP(G), when tuned for small signature sizes, are within a $\pm$ 6% range with respect to the ones of SPHINCS$^+$, with the signature for NIST security category 1 being slightly smaller. The signature size for CROSS-R-SDP(G), when tuned for fast signature is always smaller than the corresponding SPHINCS$^+$one.

We note that the reported signature sizes are slightly different from the ones obtainable from rounding to the nearest integer, the estimates coming from Equation (5). This is due to the fact that $(t - w)\log_2\left(\frac{t}{t-w}\right)$ is a simple, but sometimes loose, estimate on the number of nodes to be sent in both the Seed CSPRNG tree, and the Merkle tree in Algorithm 3. Furthermore, our choice to encode each element of the sequences rsp_0 bit-packing it separately (according to in Section 9) results in a minimal loss in encoding efficiency. We note that if byte alignment is not an implementation concern, this encoding efficiency loss can be fully removed without any change to the security or functionality of CROSS.

8.1 R-SDP Variant of CROSS

Parameter Choice 1 The parameter choice $p = 127$, $z = 7$, $n = 127$, $k = 76$ attains the claimed security level NIST category 1, i.e., 128 AES gates, roughly 143 bits. The best-performing solver is the shifted BJMM-like algorithm introduced in Section 7. As illustrated in Example 13, $\mathbb{E}_1$ with $\alpha_{\mathbb{E}_1} = 0$ and $\mathbb{D}_1$ of size $z_{\mathbb{D}_x} = 30$ with $\alpha_{\mathbb{D}_1} = 5$ were used. Optimizing the parameters of the algorithm results in

Table 3: Parameter choices, keypair and signature sizes recommended for both CROSS-R-SDP and CROSS-R-SDP(G), assuming NIST security categories 1, 3, and 5, respectively. Keypair and signature sizes of the SPHINCS$^+$ signature standard are also provided for the sake of comparison.

Algorithm and Security Category	Optim. Corner	p	z	n	k	m	t	w	Pri. Key Size (B)	Pub. Key Size (B)	Signature Size (B)
CROSS-R-SDP **1**	**fast**	127	7	127	76	-	163	85	16	61	19152
	balanced	127	7	127	76	-	252	212	16	61	12912
	small	127	7	127	76	-	960	938	16	61	10080
CROSS-R-SDP **3**	**fast**	127	7	187	111	-	245	127	24	91	42682
	balanced	127	7	187	111	-	398	340	24	91	28222
	small	127	7	187	111	-	945	907	24	91	23642
CROSS-R-SDP **5**	**fast**	127	7	251	150	-	327	169	32	121	76298
	balanced	127	7	251	150	-	507	427	32	121	51056
	small	127	7	251	150	-	968	912	32	121	43592
CROSS-R-SDP(G) **1**	**fast**	509	127	55	36	25	153	79	16	38	12472
	balanced	509	127	55	36	25	243	206	16	38	9236
	small	509	127	55	36	25	871	850	16	38	7956
CROSS-R-SDP(G) **3**	**fast**	509	127	79	48	40	230	123	24	59	27404
	balanced	509	127	79	48	40	255	176	24	59	23380
	small	509	127	79	48	40	949	914	24	59	18188
CROSS-R-SDP(G) **5**	**fast**	509	127	106	69	48	306	157	32	74	48938
	balanced	509	127	106	69	48	356	257	32	74	40134
	small	509	127	106	69	48	996	945	32	74	32742
SPHINCS$^+$-1	**fast**	-	-	-	-	-	-	-	64	32	16796
	small	-	-	-	-	-	-	-	64	32	8080
SPHINCS$^+$-3	**fast**	-	-	-	-	-	-	-	96	48	35664
	small	-	-	-	-	-	-	-	96	48	17064
SPHINCS$^+$-5	**fast**	-	-	-	-	-	-	-	128	64	49216
	small	-	-	-	-	-	-	-	128	64	29792

$\ell = 33$ and $v_0 = 72$, which implies a success probability of $2^{-22.7}$. Further, $\delta_1 = 4$ and $\delta_2 = \nu_1 = \nu_2 = 0$ are used. These parameters imply that a conservative lower bound on the memory requirement is given by 2^{116} bit. The total time complexity is conservatively lower-bounded as at least 2^{143} binary operations. For the Stern-like algorithm, $\ell = 20$ achieves the best possible performance, which requires at least 2^{149} binary operations and at least 2^{141} bits of memory.

We chose three pairs (t, w), namely $(163, 85)$, $(252, 212)$ and $(960, 938)$, where t denotes the number of rounds and w the weight of the second challenge. The (t, w) were chosen such that the forgery attack in Proposition 18 has a cost of 128 bits. Since the operations in the forgery attack have a higher cost than AES gates, we attain the NIST category 1.

Parameter Choice 3 The parameter choice $p = 127$, $z = 7$, $n = 187$, $k = 111$ attains the claimed security level NIST category 3, i.e., 192 AES gates, roughly 207 bits. The best-performing solver is the shifted BJMM-like algorithm introduced in Section 7. As illustrated in Example 13, $\mathbb{E}_1$ with $\alpha_{\mathbb{E}_1} = 0$ and $\mathbb{D}_1$ of size $z_{\mathbb{D}_x} = 30$ with $\alpha_{\mathbb{D}_1} = 5$ were used. Optimizing the parameters of the algorithm results in $\ell = 45$ and $v_0 = 104$, which implies a success probability of $2^{-29.7}$. Further, $\delta_1 = 4$ and $\delta_2 = \nu_1 = \nu_2 = 0$ are used. These parameters imply that a conservative lower bound on the memory requirement is given by 2^{169} bit. The total time complexity is conservatively lower-bounded as at least 2^{207} binary operations. For the Stern-like algorithm, $\ell = 28$ achieves the best possible performance, which requires at least 2^{213} binary operations and at least 2^{201} bits of memory.

We chose three pairs (t, w), namely $(245, 127)$, $(398, 340)$ and $(945, 907)$, where t denotes the number of rounds and w the weight of the second challenge. The (t, w) were chosen such that the forgery attack in Proposition 18 has a cost of 192 bits. Since the operations in the forgery attack have a higher cost than AES gates, we attain the NIST category 3.

Parameter Choice 5 The parameter choice $p = 127$, $z = 7$, $n = 251$, $k = 150$ attains the claimed security level NIST category 5, i.e., 256 AES gates, roughly 272 bits. The best-performing solver is the shifted BJMM-like algorithm introduced in Section 7. As illustrated in Example 13, $\mathbb{E}_1$ with $\alpha_{\mathbb{E}_1} = 0$ and $\mathbb{D}_1$ of size $z_{\mathbb{D}_x} = 30$ with $\alpha_{\mathbb{D}_1} = 5$ were used. Optimizing the parameters of the algorithm results in $\ell = 67$ and $v_0 = 152$, which implies a success probability of 2^{-29}. Further, $\delta_1 = 8$ and $\delta_2 = \nu_1 = \nu_2 = 0$ are used. These parameters imply that a conservative lower bound on the memory requirement is given by 2^{254} bit. The total time complexity is conservatively lower-bounded as at least 2^{273} binary operations. For the Stern-like algorithm, $\ell = 38$ achieves the best possible performance, which requires at least 2^{281} binary operations and at least 2^{271} bits of memory.

We chose three pairs (t, w), namely $(327, 169)$, $(507, 427)$ and $(968, 912)$, where t denotes the number of rounds and w the weight of the second challenge. The (t, w) were chosen such that the forgery attack in Proposition 18 has a cost of 256 bits. Since the operations in the forgery attack have a higher cost than AES gates, we attain the NIST category 5.

8.2 R-SDP(G) Variant of CROSS

Parameter Choice 1 The parameter choice $p = 509$, $z = 127$, $n = 55$, $k = 36$, and $m = 25$ attains the claimed security level NIST category 1, i.e., 128 AES gates, roughly 143 bits. The best-performing solver is the submatrix Stern/Dumer algorithm introduced in Section 7.1. It uses a one-dimensional subcode of support 19 and a four-dimensional subcode of support 23. Then, the algorithm requires at least 2^{143} binary operations and 2^{132} bits of memory.

We chose three pairs (t, w), namely $(153, 79)$, $(243, 206)$ and $(871, 850)$, where t denotes the number of rounds and w the weight of the second challenge. The (t, w) were chosen such that the forgery attack in Proposition 18 has a cost of 128 bits. Since the operations in the forgery attack have a higher cost than AES gates, we attain the NIST category 1.

Parameter Choice 3 The parameter choice $p = 509$, $z = 127$, $n = 79$, $k = 48$, and $m = 40$ attains the claimed security level NIST category 3, i.e., 192 AES gates, roughly 207 bits. The best-performing solver is the submatrix Stern/Dumer algorithm introduced in Section 7.1. It uses a one-dimensional subcode of support 28 and a two-dimensional subcode of support 30. Then, the algorithm requires at least 2^{210} binary operations and 2^{196} bits of memory.

We chose three pairs, (t, w), namely $(230, 123)$, $(255, 176)$ and $(949, 914)$, where t denotes the number of rounds and w the weight of the second challenge. The (t, w) were chosen such that the forgery attack in Proposition 18 has a cost of 192 bits. Since the operations in the forgery attack have a higher cost than AES gates, we attain the NIST category 3.

Parameter Choice 5 The parameter choice $p = 509$, $z = 127$, $n = 106$, $k = 69$, and $m = 48$ attains the claimed security level NIST category 5, i.e., 256 AES gates, roughly 271 bits. The best-performing solver is the submatrix Stern/Dumer algorithm introduced in Section 7.1. It uses a one-dimensional subcode of support 37 and a four-dimensional subcode of support 41. Then, the algorithm requires at least 2^{272} binary operations and 2^{259} bits of memory.

We chose three pairs (t, w), namely $(306, 157)$, $(356, 257)$ and $(996, 945)$, where t denotes the number of rounds and w the weight of the second challenge. The (t, w) were chosen such that the forgery attack in Proposition 18 has a cost of 256 bits. Since the operations in the forgery attack have a higher cost than AES gates, we attain the NIST category 5.

9 Implementation Techniques

Choice for CSPRNGs and Hash Functions CROSS requires two auxiliary primitives: a CSPRNG and a cryptographic hash function. Instances of the CSPRNG are used to sample uniformly algebraic objects, such as vectors and matrices composed of elements from $\mathbb{F}_p$ and $\mathbb{F}_z$, or to derive other seeds in a hierarchical manner, via the seed-tree construction [18].

Instances of the hash functions are used to construct a Merkle tree of the commitments, to compute the digests from which challenges are sampled and to compute the commitments themselves. In the following, we give the rationale behind our choice of concrete auxiliary primitives to instantiate our CSPRNG and hash function as stated in Table 4.

Table 4: Symmetric primitives used in CROSS for each NIST category.

NIST category	CSPRNG	Hash Function
1	SHAKE-128	SHAKE-128 with a 256 bit output
3	SHAKE-256	SHAKE-256 with a 384 bit output
5	SHAKE-256	SHAKE-256 with a 512 bit output

For the CSPRNG, we performed a comparative benchmark of AES-CTR-DRBG [11] and SHAKE, the extendable output function standardized in NIST FIPS 202 [46]. Our approach was to benchmark the two primitives in CROSS and consider, for the AES-CTR-DRBG, both a software implementation of the AES block cipher and the use of Intel AES-NI ISA extensions. Our benchmark results have shown that the SHAKE extendable output functions yield better overall performances compared to the use of AES-CTR-DRBG. Therefore, we opted for SHAKE-128 for NIST security category 1 and SHAKE-256 for NIST security categories 3 and 5.

We considered, as concrete cryptographic hash functions, the NIST standard SHA-2 (standardized in [45]), SHA-3 and SHAKE (standardized in [46]), employing digest sizes of 2λ for each security level. As a consequence, we decided to select FIPS-202 based primitives over SHA-2 to benefit from a smaller executable code size in memory-constrained devices such as microcontrollers and reduced area consumption in FPGA/ASIC implementations, thanks to the possibility of sharing the SHA-3/SHAKE inner state logic between the CSPRNG and the hash function. Furthermore, FIPS-202 primitives are designed according to criteria which minimize the Boolean degree of the round function, allowing for greater degree of protection against power side-channel attacks. Finally, we note that NIST has already expressed a preference towards no longer using SHA-2 by electing not to standardize the Kyber variants which employed it.

Our benchmarks obtained a small execution time gain by employing SHA-2 (in the few percentage points range) over SHA-3. Considering the fact that the security margin of the scheme is constrained by the collision resistance of the chosen underlying hash function, we considered the use of SHAKE128 with a 256 bit outputs for category 1, and SHAKE256 with 384 and 512 bit outputs for categories 3 and 5, respectively. The selected SHAKE functions share the same collision resistance of SHA-3 instances with the same output length (as stated in [46]), while processing the input information faster (thanks to their larger *rate* parameter). These considerations led us to choose a truncated-output SHAKE as our cryptographic hash function; this further improves on the required code complexity in software implementations and reduces the number of dedicated hardware components for hashing and random number generation to a single SHAKE128/SHAKE256 module. We note that, in order to preserve the domain separation between SHAKE and SHA-3, which is built-in in the primitive definitions in FIPS-202, we employ the presence of the integer index i appended to their inputs, adding a domain separation constant. Given that the value of the index, which is encoded over 16 bits, never exceeds 2^{15} we add 2^{15} to the indices of the instances of SHAKE employed as hash functions.

Finally, in order to simplify constant time implementations, we chose to compute the amount of randomness which should be extracted from the CSPRNGs when generating randomly the appropriate objects in such a way that the rejection sampling processes we perform fail with a probability $2^{-\lambda}$. We provide in the submission package a Python script which computes such values automatically for all our parameter sets.

Packing and Unpacking The public key's syndrome $\mathbf{s}$ and the response vectors rsp_0, which are part of the signature, consist of elements in $\mathbb{F}_p$ or $\mathbb{F}_z$. For the chosen values of p and z, the maximum number of bits needed to store values in $\mathbb{F}_p$ (respectively $\mathbb{F}_z$) does not require a number of bits that is a multiple of eight. It is reasonable to store these values bit-packed to reduce signature and public key size. For the R-SDP variant of CROSS, we therefore need $\lceil (n-k) \cdot 7/8 \rceil$ bytes for the syndrome $\mathbf{s}$, $\lceil n \cdot 7/8 \rceil$ bytes per $\mathbf{y}$ in rsp_0, and $\lceil n \cdot 3/8 \rceil$ bytes per $\boldsymbol{\delta}$ in rsp_0. For the R-SDP(G) variant of CROSS, we need $\lceil (n-k) \cdot 9/8 \rceil$ bytes for the syndrome $\mathbf{s}$, $\lceil n \cdot 9/8 \rceil$ bytes per $\mathbf{y}$ in rsp_0, and $\lceil n \cdot 7/8 \rceil$ bytes per $\boldsymbol{\sigma}$ in rsp_0. The bit-packed pattern for $\mathbb{F}_p$ elements in the R-SDP variant of CROSS and $\mathbb{F}_z$ elements in the R-SDP(G) variant of CROSS is shown in Figure 12, while the bit-packed pattern for $\mathbb{F}_p$ elements in the R-SDP(G) variant of CROSS is depicted in Figure 13. Finally, Figure 14 shows the bit-packed pattern for $\mathbb{F}_z$ elements in the R-SDP variant of CROSS.

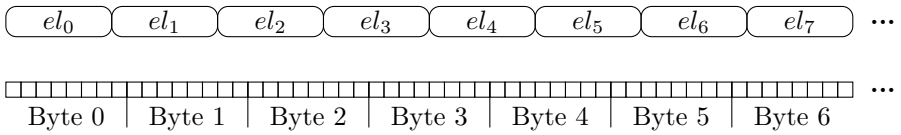

Figure 12: Packing of elements with $p = 127$ and $z = 127$, $\mathbf{s} = \{el_0, ..., el_{n-k}\}$, $\boldsymbol{y} = \{el_0, ..., el_n\}$ and $\boldsymbol{\sigma} = \{el_0, ..., el_n\}$

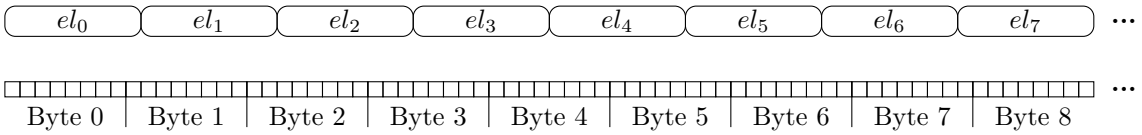

Figure 13: Packing of elements with $p = 509$, $\mathbf{s} = \{el_0, ..., el_{n-k}\}$, $\mathbf{y} = \{el_0, ..., el_n\}$

Efficient arithmetic for $\mathbb{F}_7$ and $\mathbb{F}_{127}$ Implementing CROSS requires, besides the auxiliary CSPRNG and hash function, a set of arithmetic primitives which act on collections of either $\mathbb{F}_p$ or $\mathbb{F}_z$ elements. The simple nature of the arithmetic operations allows for a straightforward constant time implementation. In particular, vector additions, vector subtractions, and point-wise vector multiplications are realized by countable loops, with a compile-time determined trip-count. Similarly, matrix-vector multiplications by either $\mathbf{H}$ or $\mathbf{M}_G$ are characterized by countable nested loops sharing the data-independent execution time of the vector operations.

The only arithmetic operation which may be affected by a variable time implementation is the computation which, given a vector $\boldsymbol{\eta} = [\mathbf{v}[0], \ldots, \mathbf{v}[n-1]]$ in $\mathbb{F}_z$, computes the vector $\mathbf{e} = [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$ in $\mathbb{F}_p$ such that for all $0 \leq i < n$ we have $\mathbf{e}[i] = g^{\mathbf{v}[i]}$, where g is the generator of the restricted subgroup $\mathbb{E}$. A straightforward implementation would employ a square and multiply strategy, which is affected by timing side-channel vulnerabilities. To avoid this issue, we resorted to two different techniques, depending on whether $z = 7$ or $z = 127$, which are the only two values which we need to treat. In the $z = 7$ case, we have that $p = 127$, and therefore its elements can be stored in a single byte, encoded as in natural binary encoding. As a consequence, it is possible to fit the entire look-up table for the seven values $\{g^0, g^1, \ldots, g^6\}$ in a single, 64-bit register. A look-up in this single-register-sized table takes constant time as the entire table is loaded, regardless of the value being looked up. In the $z = 127$ case, we have that $p = 509$. As a consequence, for software implementations, two bytes are required to represent an $\mathbb{F}_p$ element, and the table-based approach cannot be applied in the same straightforward fashion, as in the aforementioned case. To this end, we implement the g^i operation through a square-and-multiply approach where all the values $\{g^{2^0}, g^{2^1}, \ldots, g^{2^6}\}$ mod p are precomputed constants, which are composed through a single arithmetic expression where each power of two is selected via an arithmetic predicated expression. The modular reductions are performed tree-wise to reduce their number to a minimum.

A final note on the arithmetic employed to implement computations on both $\mathbb{F}_7$ and $\mathbb{F}_{127}$ concerns the runtime data representation. We work, in both cases, performing reductions modulo 8 and 128 respectively, thus resulting in a double representation of the zero value (as 0 and 7 for $\mathbb{F}_7$, and as 0 and 127 for $\mathbb{F}_{127}$). This, in turn, effectively reduces the cost of the modular reductions to, at most, two shift and add operations. The values with the double-zero representation are then normalized via a constant time arithmetic expression before emission.

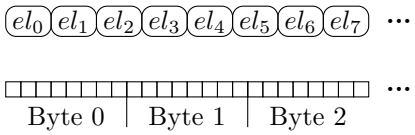

Figure 14: Packing of elements with $z = 7$, $\boldsymbol{\delta} = \{el_0, ..., el_n\}$

Merkle tree and proof To reduce the signature size, we implement a Merkle tree structure for the cmt_0 hashes and include its root d_0, as well as a challenge-dependent Merkle proof MerkleProofs in the signature. For t leaves, a Merkle tree contains $2t-1$ nodes that can be stored in a one-dimensional array, where each entry corresponds to a hash value. For a balanced tree where $t = 2^k$ for some $k \in \mathbb{N}$, the leaves are located at the last t entries, and traversing through the tree is straightforward, as each level i in the tree contains precisely half of the nodes of level $i+1$. For CROSS, the number of leaves corresponds to the number of rounds t that are computed. As seen from Table 3, the choice of t will not lead to balanced tree structures. Implementing a balanced structure merely for the sake of simplicity of tree traversal would lead to a significant amount of unused memory. Therefore, it has been decided to implement unbalanced trees and use some small additional arrays that store the extra information required to compute the parent or child node indices. The positions of leaves inside the Merkle tree array also depend on the unbalanced tree's specific structure, so we implement another array of size t to store these indices. It has to be mentioned that these arrays can be pre-computed offline for each choice of t and, thus, would have no impact on performance. Therefore, dedicated procedural descriptions of these setup algorithms, SETUPTREE() and LEAFINDICES(), are not included in the specification document. Their basic idea is, however, to construct the tree by iteratively traversing through the right children (starting from the root) and finding balanced sub-trees on the left-child nodes. Accumulating the number of nodes in each tree level (stored in **num**) then allows one to compute the level offsets **off** and leaf indices **idx**.

The computation of the Merkle tree and its root is shown in Algorithm 4. Taking the commitment hashes cmt_0, it returns the Merkle tree $\mathcal{T}$ and its corresponding root d_0. In the first step, the arrays **off**, **num**, and **idx** are computed and store the information on the tree structure as described above (again, these could be pre-computed for any choice of t). The commitment values are placed at their corresponding positions in the tree. Finally, starting at the last node in the tree, the parent nodes are computed by hashing their siblings. Two additional variables ctr and lvl are used to choose the correct value from the offset array **off**. Their updating procedure is abstracted as UPDATECTR() for compactness. It decrements the variable lvl by one if ctr reaches the number of nodes per level as stored in **num**. The root d_0 is the first digest in the tree $\mathcal{T}$.

Algorithm 4: MERKLEROOT(cmt_0)

Input: $\mathsf{cmt}_0[0], \ldots, \mathsf{cmt}_0[t-1]$: commitments representing the leaves.
Output: d_0: The root of the Merkle tree.
$\mathcal{T}$: The Merkle tree structure.
Data: $\mathcal{T}$: The Merkle tree structure comprises $2t-1$ nodes, where each node is a hash digest of its children.
idx: A vector of t entries storing the indices of the leaves.
num: A vector of $\log_2(t)+1$ entries storing the number of nodes per tree level.
off: A vector of $\log_2(t)+1$ entries storing offsets for the index computation of parent and child nodes.

```
// Can be pre-computed at compile time, only depend on t
off, num ← SETUPTREE()
idx ← LEAFINDICES(off)
// Place commitments on leaves
for i ← 0 to t do
    T[idx[i]] ← cmt_0[i]
end
// Hash child nodes to create digest of parent node
lvl ← log_2(t) − 1
ctr ← 0
for i ← 2t − 2 to 1 by −2 do
    p ← (i − 1)/2 + off[lvl]
    T[p] ← HASH(T[i − 1] || T[i])
    ctr, lvl ← UPDATECTR(ctr, lvl, num)
end
d_0 ← T[0]
return d_0, T
```

Algorithm 5 shows the generation of the Merkle proof MerkleProofs and works as follows. In a first step, the helper arrays **off**, **num**, and **idx** are computed as before. As the Merkle proof should only be generated for a subset of commitments $\mathsf{cmt}_0[i]$ for which the challenge bit $\boldsymbol{b}[i] = 0$, an additional flag tree $\mathcal{T}'$ is used to indicate the corresponding subset of commitments. Therefore, $\mathcal{T}'$ is initialized to all 0, and only the entries for nodes for which the proof should be computed are set to 1, as shown in lines 3 to 8. From line 12 onward, the algorithm traverses through the tree and sets all the entries in $\mathcal{T}'$ to 1 as soon as at least one of its children has been labeled accordingly. Finally, if only one of the two child nodes has been computed (or is a leaf for which the proof is created), its uncomputed sibling is added to the Merkle proof MerkleProofs. Notice that if both siblings are labeled as computed, nothing needs to be added to the proof, as the corresponding parent node can already be computed from its children.

Algorithm 5: MerkleProofs($\mathcal{T}, \mathsf{cmt}_0, \mathbf{b}$)

Input: $\mathcal{T}$: The Merkle tree computed during MerkleRoot.
$\mathsf{cmt}_0[0], \ldots, \mathsf{cmt}_0[t-1]$: commitments representing the leaves.
$\mathbf{b}$: The challenge vector of size t.
Output: MerkleProof: The vector containing the Merkle proof nodes.
Data: $\mathcal{T}$: The Merkle tree structure comprises $2t-1$ nodes, where each node is a hash digest of its children.
$\mathcal{T}'$: A binary flag tree of $2t-1$ nodes. A node labeled with 1 is computed and, thus, is not required in the proof.
idx: A vector of t entries storing the indices of the leaves.
num: A vector of $\log_2(t)+1$ entries storing the number of nodes per tree level.
off: A vector of $\log_2(t)+1$ entries storing offsets for the index computation of parent and child nodes.

```
   // Can be pre-computed at compile time, only depend on t
 1 off, num ← setupTree()
 2 idx ← leafIndices(off)
   // Label T' with leaf nodes for which proof is created
 3 T' ← {0}
 4 for i ← 0 to t − 1 do
 5     if b[i] = 0 then
 6         T'[idx[i]] ← 1
 7     end
 8 end
 9 lvl ← log2(t) − 1
10 ctr ← 0
11 len ← 0
12 for i ← 2t − 2 to 1 by −2 do
13     p ← (i − 1)/2 + off[lvl]
14     T'[p] ← T'[i − 1] ∨ T'[i]
       // Right child computed, left child not so add it to proof
15     if T'[i] = 1 ∧ T'[i − 1] = 0 then
16         MerkleProofs[len] ← T[i − 1]
17         len ← len + 1
18     end
       // Left child computed, right child not so add it to proof
19     if T'[i] = 0 ∧ T'[i − 1] = 1 then
20         MerkleProofs[len] ← T[i]
21         len ← len + 1
22     end
23     ctr, lvl ← updateCtr(ctr, lvl, num)
24 end
25 return MerkleProofs
```

The final algorithm RecomputeMerkleRoot(), upon taking the commitment values cmt_0, the Merkle Proof MerkleProofs, and the challenge vector $\mathbf{b}$, re-computes the root of the Merkle tree $\mathcal{T}$ and

returns its root d'_0 for verification. The procedure is described in Algorithm 6, and as before, uses a flag tree $\mathcal{T}'$ to indicate computed nodes.

In lines 3 to 9, $\mathcal{T}$ and $\mathcal{T}'$ are initialized by placing the commitments $\mathsf{cmt}_0[i]$ that were re-computed by the verifier on the tree $\mathcal{T}$ and by labeling the corresponding entries in $\mathcal{T}'$ as computed. When traversing through the tree afterward, positions are skipped where both siblings are not computed (and are thus unused). Otherwise, if at least one sibling has been computed, the corresponding node is either taken from the tree (if computed) or from MerkleProofs (if not computed) and placed in a temporary vector $\mathbf{h}$ that stores the hash input. This input is then hashed, and the digest is placed on the Merkle tree $\mathcal{T}$ at the position of the parent node. This parent node is then also labeled as 1 in the flag tree $\mathcal{T}'$. After iterating through the tree, the root d'_0 to be returned is stored at the first position in $\mathcal{T}$.

Algorithm 6: RecomputeMerkleRoot($\mathsf{cmt}_0, \mathsf{MerkleProofs}, \mathbf{b}$)

Input: $\mathsf{cmt}_0[0], \ldots, \mathsf{cmt}_0[t-1]$: commitments representing the leaves.
MerkleProofs: Merkle proof nodes provided in the signature.
$\mathbf{b}$: The challenge vector of size t.

Output: d'_0: The recomputed root of the Merkle tree.

Data: $\mathcal{T}$: The Merkle tree structure comprised of $2t-1$ nodes, where each node is a hash digest of its children.
$\mathcal{T}'$: A binary flag tree of $2t-1$ nodes. A node labeled with 1 is computed and, thus, is not required from the proof.
idx: A vector of t entries storing the indices of the leaves.
num: A vector of $\log_2(t)+1$ entries storing the number of nodes per tree level.
off: A vector of $\log_2(t)+1$ entries storing offsets for the index computation of parent and child nodes.

```
// Can be pre-computed at compile time, only depend on t
off, num ← setupTree()
idx ← leafIndices(off)
// Fill T and label T' with leaf nodes that were computed by the verifier
T' ← {0}, T ← {0}
for i ← 0 to t − 1 do
    if b[i] = 0 then
        T'[idx[i]] ← 1
        T[idx[i]] ← cmt_0[i]
    end
end
lvl ← log_2(t) − 1
ctr ← 0
len ← 0
for i ← 2t − 2 to 1 by −2 do
    if T'[i] = 0 ∧ T'[i − 1] = 0 then
        ctr, lvl ← updateCtr(ctr, lvl, num)
        continue
    end
    // Take first node from tree if valid, else from proof.
    if T'[i] = 1 then
        h[1] ← T[i]
    end
    else
        h[1] ← MerkleProofs[len]
        len ← len + 1
    end
    // Take second node from tree if valid, else from proof.
    if T'[i − 1] = 1 then
        h[0] ← T[i − 1]
    end
    else
        h[0] ← MerkleProofs[len]
        len ← len + 1
    end
    p ← (i − 1)/2 + off[lvl]
    T[p] ← Hash(h)
    T'[p] ← 1
    ctr, lvl ← updateCtr(ctr, lvl, num)
end
d'_0 ← T[0]
return d'_0
```

Table 5: Computation time expressed in clock cycles for all CROSS primitives and variants. Measurements collected via `rtdscp` on an AMD Ryzen 5 Pro 3500U , clocked at 2.1GHz. The computer was running Debian GNU/Linux 12. Computation time benchmarks for SPHINCS$^+$ signature scheme "simple" variant (the "robust" is slower) and the RSA signature taken from eBACS [15], software package version `supercop-20230530`, running on the Intel Core i3-10110U "comet" machine

NIST Cat.	Parameter Set	KeyGen (Mcycles)	Sign (Mcycles)	Verify (Mcycles)
1	CROSS-R-SDP-f	0.04	1.28	0.78
	CROSS-R-SDP-b	0.04	2.38	1.44
	CROSS-R-SDP-s	0.04	8.96	5.84
	CROSS-R-SDP-(G)-f	0.02	0.94	0.55
	CROSS-R-SDP-(G)-b	0.02	1.85	1.09
	CROSS-R-SDP-(G)-s	0.02	6.54	3.96
3	CROSS-R-SDP-f	0.08	2.75	1.69
	CROSS-R-SDP-b	0.08	4.97	2.89
	CROSS-R-SDP-s	0.08	12.20	6.80
	CROSS-R-SDP-(G)-f	0.04	2.04	1.21
	CROSS-R-SDP-(G)-b	0.04	2.63	1.53
	CROSS-R-SDP-(G)-s	0.04	9.67	5.61
5	CROSS-R-SDP-f	0.14	4.93	3.04
	CROSS-R-SDP-b	0.14	8.26	5.00
	CROSS-R-SDP-s	0.14	15.69	9.37
	CROSS-R-SDP-(G)-f	0.07	3.93	2.32
	CROSS-R-SDP-(G)-b	0.07	4.99	2.96
	CROSS-R-SDP-(G)-s	0.07	14.12	7.73
SPHINCS+	(1-f-SHAKE)	4.68	153.1	9.16
SPHINCS+	(1-s-SHAKE)	149.63	2,407.09	3.81
SPHINCS+	(3-f-SHAKE)	6.84	194.89	15.48
SPHINCS+	(3-s-SHAKE)	144.57	3,313.34	4.67
SPHINCS+	(5-f-SHAKE)	12.33	291.77	12.29
SPHINCS+	(5-s-SHAKE)	597.97	6,939.57	9.36
RSA	3,072 (SHA-2)	1,145.5	8.78	0.095

10 Detailed Performance Analysis

We benchmarked the performance of CROSS on an AMD Ryzen 5 Pro 3500U , clocked at 2.1GHz, with 8GiB of DDR4. The computer was running Debian GNU/Linux 12, and the benchmark binaries were compiled with `gcc` 12.2.0 (Debian 12.2.0-14). The computation times are measured in clock cycles, the clock cycle count has been gathered employing the `rtdscp` instruction, which performs instruction fencing. All numbers of clock cycles reported were obtained as the average of 10k runs of the same primitive. All the timings for CROSS were taken with respect to the current AVX2 optimized implementation.

We report in Table 5 the required number of clock cycles to compute the KeyGen, Sign and Verify signature algorithms. We also report, to provide the means of a comparison, the number of clock cycles taken to run the SPHINCS$^+$ signature scheme, as provided by the eBACS platform [15], benchmarking reference. The timings for SPHINCS$^+$ refer to its current best optimized version. For CROSS the "f" letter in the parameter set denotes a "fast" optimization corner, "b" the balanced one and "s" denotes a short (signature) optimization corner. SPHINCS$^+$ does not offer a balanced option.

The reported timings show how CROSS achieves significantly better computational performance than SPHINCS$^+$: in particular, considering NIST security category 1 as an example, the speed-optimized parameter sets for CROSS-R-SDP yield an $\approx 119\times$ faster signature primitive, and an $\approx 11\times$ faster verification primitive. CROSS-R-SDP(G) parameter configurations bring the speed advantage to $\approx$

Table 6: Main memory footprints of the runtime-memory optimized version of CROSS

NIST Cat.	Parameter Set	Reference Size (kiB)	Compact Size (kiB)	Gain (x)
1	CROSS-R-SDP-f	152.38	28.54	5.34
	CROSS-R-SDP-b	218.89	23.34	9.38
	CROSS-R-SDP-s	696.38	26.77	26.01
	CROSS-R-SDP-(G)-f	111.20	20.66	5.38
	CROSS-R-SDP-(G)-b	163.46	18.30	8.93
	CROSS-R-SDP-(G)-s	548.62	23.85	23.00
3	CROSS-R-SDP-f	334.92	57.88	5.79
	CROSS-R-SDP-b	342.26	52.35	6.54
	CROSS-R-SDP-s	1213.26	46.46	26.11
	CROSS-R-SDP-(G)-f	240.54	38.73	6.21
	CROSS-R-SDP-(G)-b	259.32	35.60	7.28
	CROSS-R-SDP-(G)-s	875.84	38.27	22.89
5	CROSS-R-SDP-f	596.95	98.71	6.05
	CROSS-R-SDP-b	857.85	76.21	11.26
	CROSS-R-SDP-s	1639.82	73.74	22.24
	CROSS-R-SDP-(G)-f	424.21	65.26	6.50
	CROSS-R-SDP-(G)-b	475.33	57.46	8.27
	CROSS-R-SDP-(G)-s	1231.35	56.95	21.62

162× for signing, and gain a factor of 16.5× for verification. Furthermore, the signature-size optimized parameters for CROSS still provide faster signature timings with respect to the SPHINCS$^+$ parameter sets optimized for fast operation, for all NIST security categories.

Finally, one noteworthy point, is that the CROSS signature algorithm is faster than RSA for the equivalent (pre-quantum) security parameters for RSA, when considering the fast and balanced options for RSDP and all options for RSDP-(G).

To provide a concrete grounding for practical use, we observe that CROSS-R-SDP, for NIST security category 1 achieves sub-millisecond signature creation (0.61ms) and verification (0.37) ms on the platform we employed for the benchmarks. CROSS-R-SDP(G) performs even better, signing in 0.44ms, and verifying in 0.26ms.

Concerning the computational load of CROSS-R-SDP, about $\approx 60\%$ of the time taken by the signature primitive is spent computing either hashes or CSPRNGs. This computational load profile is essentially the same during verification, as a result, in both cases, of the optimization of the arithmetic operations with AVX2 vector instructions.

Concerning embedded systems oriented implementations, a stringent metric is the one of the main memory footprint. Indeed on microcontroller platforms the required stack size (plus eventually global variables, if present) can prevent the use of a cryptographic primitive. To provide a first datapoint, we optimized CROSS to reduce the required stack size, recomputing mathematical objects which are generated from CSPRNGs whenever they are needed by the computation, while keeping only the seeds themselves in memory. Similarly, the leaves of the seed tree are computed performing an in-order visit of the data structure, reducing the amount of data being kept in main memory. Table 6 reports the figures of the compact memory footprint version of CROSS with respect to the ones required by the reference implementation. As it can be seen, all parameter sets for CROSS can be fit in the main memory of the STM32F407VG microcontroller (which is endowed with 192kiB of memory, split into a 64 and a 128 kiB bank), currently employed by the the `pqm4` benchmarking project.

11 Known Answer Test Values

Known Answer Tests (KAT) have been generated and are a separate archive. The submission package contains facilities (in the Additional Implementation folder) to regenerate them, following the instructions in the `README` file. We include the SHA-2-512 digests of the KAT requests and responses in the following.

```
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_106_32742.req
ce5070783658e2946efbcbe404b612aac1307bed929dec7fc501b2a6e9a34d5fa2db644fd8e29b1c7adcc26bd4f805fef399e02282dd6cf7ad73ea746195eea1  PQCsignKAT_106_32742.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_106_40134.req
02c37076b2d46559db548b93728c69042d42ed5fb87b98883acb75b408de4f6acdd5d0b3d050ffe907b3c24a199b99bc424905ef5b4303adea24fab4eb784c65  PQCsignKAT_106_40134.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_106_48938.req
79b8d74e80046ebe4fdadccb6c7ef92986b2f6860268dc77ef1d783c33b7ce73bdf44338b99294471181588c4963e311d0dbfa413416edec70cf7b5a397ff702  PQCsignKAT_106_48938.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_115_23642.req
61f8d0f0e35cb95072d8af1b796d13d27c0e2eb4c3596fbe0a68bb01ec63d6221cb9ab52856461f1c1af392f09adae0f25f0546a49afdd00cafc50f9d69f5993  PQCsignKAT_115_23642.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_115_28222.req
b226f3fef34df53f36cfe62b3e0651c9b53ca388764bcb0a2150488fb1e8929fe26374905ff83a0fb4097651a4df679b5b04c3d1e843cf9343ac024f0831a82f  PQCsignKAT_115_28222.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_115_42682.req
c1fb628fda6e11597dbc6f2a5bb1611eae2b956a7f5a62f80e9a840f759fc356f005e8879c3219ed3c23a5b824eaa8d08cabecd0da07b6138cc9a10c6d3b9d8c  PQCsignKAT_115_42682.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_153_43592.req
c5bc6704a223de954a0e55ef5a0fe9a24f87f8fc67410615469a58d8f81d9440c2157c4cfdc9e93c75e8cd5c41bd95669c7a3db4b16b30f666e7b1457c1b95e9  PQCsignKAT_153_43592.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_153_51056.req
d9afb26987ec1dde80b5a0c6031a29f6a6b16164ea7c12df166ca184a61278658a0d863161ccbb597c733e73b0d6fabfb2fa47e4552efbeecb2e437da392734b  PQCsignKAT_153_51056.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_153_76298.req
a25d61b5ee2fbc7948bdf0459e9d056fd5e535555ddff195c982a878c8de316bb905501eab556dc4e33c18136c4df766265cd21cd7f9b4f611a8695b57cb9b1c  PQCsignKAT_153_76298.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_54_12472.req
f943e6ec6fa2eb1ea841b7514eb0bbdf924c5be04ee68ce1c96d7bb450e444a16817794239b04e716bb1de621ceb40c043d61d3a04df82716adc981d802c0b28  PQCsignKAT_54_12472.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_54_7956.req
ea9a36027fbd080c5cde6f73de0546151bcef38947915b9f760b8e93db48f3c635814fab43996c547f1db411f52b581aca2e945d8ab921db892830aff8060f08  PQCsignKAT_54_7956.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_54_9236.req
cba39f8eb2d2da26f14baaf4054a19595ce0d84eee25653c5e51d1c89d514f9d348ba090991d1a36a8fa98110948508a111a8695e40570ed1b2efdb068979a7f  PQCsignKAT_54_9236.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_77_10080.req
397ef30ad39647fdf352767f009ec89151d54a7ed51546fc1b75ffd9fa4576a7fddfadca6144fb06a427fd6511c7c98beb0c22dfe243c12d6b610ea2e8208c92  PQCsignKAT_77_10080.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_77_12912.req
52d73ae699f58351b5347f016f0387a0a42500063534d716f335e1a49e7afb099b42afb5c4df95fb4f83e821950fd715c7e547326b857e69277343091c3ca219  PQCsignKAT_77_12912.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_77_19152.req
4333db07df5a6d767e72be60848cc1a4c096ba1c1a6744e6c9d307274c26952bed2fb2bc3dac7d2756b7a7872e86f8ea7a16e131ff67e572f5dd8fefebfecf48  PQCsignKAT_77_19152.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_83_18188.req
bdfde14b53e23938ae471b66844d32977d5ea799d5b351f4d36679386775f2aa1eb9a32b32715ed22ce898133a0e42a0f3317f63c4f6d6105208133d4cc7ae83  PQCsignKAT_83_18188.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_83_23380.req
6e928536d6f138c77d31471a5c0cf735288e8ce1d42986115219bb2de477aad3cd907a12eb85bb62de4fd6b960cc921e8fdc7a1d237491cf5feacc556b550b34  PQCsignKAT_83_23380.rsp
a87eccf3d19fd50883d3a2c21435ac031e998c7d20f9ba81da57a70b9709f99b77fef37cae8856740002e15c46d2873348a9b37ad07a59659076b5e8a46a8458  PQCsignKAT_83_27404.req
a9a2bbd89b7d7afebd3a55c9756957df7a2070e1812f8f383d8d93251abba8fe3b183dbd636af89cf0eb54946e211dc8a23b7766f5474e87a2ea67ce9c5f1ecb  PQCsignKAT_83_27404.rsp
```

12 Advantages and Limitations

Advantages

- Due to the usage of restricted errors and the resulting full Hamming weight, generic decoders in this setting have an increased cost compared to generic decoders in the Hamming metric. As our thorough security analysis shows, this allows us to choose smaller parameters to achieve the same security level. We have adapted the best-known techniques from classical ISDs and subset-sum solvers.

- By leveraging a ZK protocol, we do not require any code with algebraic structure and thus do not rely on any indistinguishability assumption. The used code is chosen uniformly at random and is made public. Since the secret is given by the randomly chosen restricted error vector, an adversary faces an NP-hard problem: either R-SDP or R-SDP(G).

- The ZK protocol CROSS-ID follows the well-established structure of CVE [26], which is a well-known and studied protocol. It can also be classified as a q2-Identification protocol, which immediately implies EUF-CMA security [41].

- The choice of a ZK protocol allows for a flexible choice of parameters, trading performance for signature size and vice versa.

- We considered the attack to signatures derived from 5-pass ID protocols reported in [43] and performed the corresponding analysis when fixed-weight challenges are employed. We considered the computational improvements of this work and designed the system parameters conservatively.

- Restricted error vectors and their transformations can be compactly represented, which significantly reduces the signature sizes compared to other settings, such as when using fixed Hamming weight error vectors.

- The fully random parity-check matrix can be derived on the fly from a small seed using a CSPRNG. This allows us to compress the public key to < 0.1 kB, which means the signature scheme is suitable for highly memory-constrained devices such as smartcards. Furthermore, the small public key size and sub-10 kB signature sizes endorse its use in X.509 certificates.

- The transformations of restricted vectors do not require permutations, which ensures a simplified constant-time implementation.

- Since roughly half of the operations are performed in a smaller field, $\mathbb{F}_z$, the computations there will be less expensive than in other schemes which use the full ambient space.

- Due to the order of the ambient spaces $\mathbb{F}_p$ and $\mathbb{F}_z$ being either a Mersenne prime or close to one, CROSS enjoys fast arithmetic and achieves fast signature generation and verification.
- Since CROSS only chooses two different ambient spaces, namely $(p = 127, z = 7)$ and $(p = 509, z = 127)$, the code size and area of its realization are more compact concerning schemes that require tailored arithmetic for each NIST security category.
- For the R-SDP variant of CROSS, the choice of z is small enough to allow expensive operations to be performed via a constant-time table lookup, as the entire table fits into a (64-bit) register.
- CROSS only requires simple operations, such as symmetric primitives (CSPRNGs and cryptographic hashes) and vector/matrix operations among small elements. This also allows for a straightforward constant-time implementation of the scheme.
- The nature of the arithmetic operation in CROSS allows efficient vectorization with ISA extensions such as Intel's AVX2: the computation of the arithmetic operations, when vectorized, reduces the amount of time spent in them to a minority in the overall signature time
- Only a single standardized primitive (SHAKE, as per FIPS-202) is required in each CROSS implementation, reducing both hardware and software implementation complexity.

Limitations

- The achieved signature sizes are still in the range of 8 kB for NIST level I, which is larger than the standardized signatures Falcon and Dilithium but matches that of SPHINCS+. This range of signature sizes is to be expected from a signature scheme derived through a ZK protocol.
- The restricted syndrome decoding problem is relatively new [7], but closely related to the classical syndrome decoding problem and the subset sum problem, both of which are well studied in literature [12, 13]. Due to this relation, the best-known solvers for R-SDP [8, 23] are modifications of the best-known solvers for SDP and the subset sum problem.

13 Acknowledgments

Violetta Weger is supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 899987.

Felice Manganiello and Freeman Slaughter are partially supported by the American National Science Foundation (NSF) under grant DMS-1547399.

Sebastian Bitzer and Antonia Wachter-Zeh were supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under Grant No. WA3907/4-1, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 801434), and acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK002.

Patrick Karl also acknowledges the financial support by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK002.

14 Bibliography

[1] Footguns as an axis for security analysis. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/l2iYk-8sGnI/m/sHWyfvfNDAAJ.

[2] Carlos Aguilar, Philippe Gaborit, and Julien Schrek. A new zero-knowledge code based identification scheme with reduced communication. In *2011 IEEE Information Theory Workshop*, pages 648–652. IEEE, 2011.

[3] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the sdith. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 564–596. Springer, 2023.

[4] Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed σ-protocol theory for lattices. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II*, pages 549–579. Springer, 2021.

[5] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-Shamir transformation of multi-round interactive proofs. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part I*, pages 113–142. Springer, 2022.

[6] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. A finite regime analysis of information set decoding algorithms. *Algorithms*, 12(10):209, 2019.

[7] Marco Baldi, Massimo Battaglioni, Franco Chiaraluce, Anna-Lena Horlemann-Trautmann, Edoardo Persichetti, Paolo Santini, and Violetta Weger. A new path to code-based signatures via identification schemes with restricted errors. *arXiv preprint arXiv:2008.06403*, 2020.

[8] Marco Baldi, Sebastian Bitzer, Alessio Pavoni, Paolo Santini, Antonia Wachter-Zeh, and Violetta Weger. Zero knowledge protocols and signatures from the restricted syndrome decoding problem. *PKC 2024*, 2024.

[9] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the F5 Gröbner basis algorithm. *Journal of Symbolic Computation*, 70:49–70, 2015.

[10] Alexander Barg. Some new NP-complete coding problems. *Problemy Peredachi Informatsii*, 30(3):23–28, 1994.

[11] Elaine Barker and John Kelsey. NIST SP 800-90A Rev. 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators. https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final, 2015.

[12] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 364–385. Springer, 2011.

[13] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How 1+ 1= 0 improves information set decoding. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 520–536. Springer, 2012.

[14] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[15] Daniel J. Bernstein. eBACS: ECRYPT Benchmarking of Cryptographic Systems. https://bench.cr.yp.to/supercop.html.

[16] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: ball-collision decoding. In *Advances in Cryptology–CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings 31*, pages 743–760. Springer, 2011.

[17] Slim Bettaieb, Loïc Bidoux, Olivier Blazy, and Philippe Gaborit. Zero-knowledge reparation of the Véron and AGS code-based identification schemes. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 55–60. IEEE, 2021.

[18] Ward Beullens. Sigma protocols for MQ, PKP and SIS, and fishy signature schemes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 183–211. Springer, 2020.

[19] Loïc Bidoux and Philippe Gaborit. Shorter signatures from proofs of knowledge for the SD, MQ, PKP and RSD problems. *arXiv preprint arXiv:2204.02915*, 2022.

[20] Loïc Bidoux, Philippe Gaborit, Mukul Kulkarni, and Victor Mateu. Code-based signatures from new proofs of knowledge for the syndrome decoding problem. *Designs, Codes and Cryptography*, pages 1–48, 2022.

[21] Loïc Bidoux, Philippe Gaborit, and Nicolas Sendrier. Quasi-cyclic Stern proof of knowledge. *arXiv preprint arXiv:2110.05005*, 2021.

[22] Mina Bigdeli, Emanuela De Negri, Manuela Muzika Dizdarevic, Elisa Gorla, Romy Minko, and Sulamithe Tsakou. *Semi-Regular Sequences and Other Random Systems of Equations*. Springer International Publishing, Cham, 2021.

[23] Sebastian Bitzer, Alessio Pavoni, Violetta Weger, Paolo Santini, Marco Baldi, and Antonia Wachter-Zeh. Generic decoding of restricted errors. In *2023 IEEE International Symposium on Information Theory (ISIT)*, pages 246–251. IEEE, 2023.

[24] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[25] Alessio Caminata and Elisa Gorla. Solving multivariate polynomial systems and an invariant from commutative algebra. In Jean Claude Bajard and Alev Topuzoğlu, editors, *Arithmetic of Finite Fields*, pages 3–36, Cham, 2021. Springer International Publishing.

[26] Pierre-Louis Cayrel, Pascal Véron, and Sidi Mohamed El Yousfi Alaoui. A zero-knowledge identification scheme based on the q-ary syndrome decoding problem. In *International Workshop on Selected Areas in Cryptography*, pages 171–186. Springer, 2010.

[27] André Chailloux. On the (In) security of optimized Stern-like signature schemes. In *Proceedings of WCC 2022: The Twelfth International Workshop on Coding and Cryptography, March 7 - 11, 2022, Rostock (Germany). URL: https://www.wcc2022.uni-rostock.de/storages/uni-rostock/Tagungen/WCC2022/Papers/WCC_2022_paper_54.pdf*, 2022.

[28] Il'ya Isaakovich Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25(1):24–32, 1989.

[29] Andre Esser, Alexander May, and Floyd Zweydinger. McEliece needs a break–solving McEliece-1284 and quasi-cyclic-2918 with modern isd. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part III*, pages 433–457. Springer, 2022.

[30] Thibauld Feneuil. Building MPCitH-based signatures from MQ, MinRank, Rank SD and PKP. *Cryptology ePrint Archive*, 2022.

[31] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Shared permutation for syndrome decoding: New zero-knowledge protocol and code-based signature. *Cryptology ePrint Archive*, 2021.

[32] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Shared permutation for syndrome decoding: New zero-knowledge protocol and code-based signature. *Designs, Codes and Cryptography*, pages 1–46, 2022.

[33] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: shorter signatures from zero-knowledge proofs. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, pages 541–572. Springer, 2022.

[34] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto*, volume 86, pages 186–194. Springer, 1986.

[35] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In *Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15*, pages 88–105. Springer, 2009.

[36] Giulia Gaggero and Elisa Gorla. The complexity of solving a random polynomial system.

[37] Shay Gueron, Edoardo Persichetti, and Paolo Santini. Designing a practical code-based signature scheme from zero-knowledge proofs with trusted setup. *Cryptography*, 6(1):5, 2022.

[38] Cheikh Thiécoumba Gueye, Jean Belo Klamti, and Shoichi Hirose. Generalization of BJMM-ISD using May-Ozerov nearest neighbor algorithm over an arbitrary finite field $\mathbb{F}_q$. In *International Conference on Codes, Cryptology, and Information Security*, pages 96–109. Springer, 2017.

[39] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

[40] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Advances in Cryptology – EUROCRYPT 2010*, pages 235–256. Springer, 2010.

[41] Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass MQ-based identification to MQ-based signatures. *IACR Cryptol. ePrint Arch.*, 2016:708, 2016.

[42] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

[43] Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In *Cryptology and Network Security: 19th International Conference, CANS 2020, Vienna, Austria, December 14–16, 2020, Proceedings*, pages 3–22. Springer, 2020.

[44] Felice Manganiello and Freeman Slaughter. Generic error SDP and generic error CVE. In Andre Esser and Paolo Santini, editors, *Code-Based Cryptography*, pages 125–143, Cham, 2023. Springer Nature Switzerland.

[45] National Institute of Standards and Technology. FIPS 180-4 - Secure Hash Standard (SHS). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf, 2015.

[46] National Institute of Standards and Technology. FIPS 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf, 2015.

[47] Paolo Santini, Marco Baldi, and Franco Chiaraluce. Computational hardness of the permuted kernel and subcode equivalence problems. *IEEE Transactions on Information Theory*, 2023.

[48] Jacques Stern. A method for finding codewords of small weight. In *International Colloquium on Coding Theory and Applications*, pages 106–113. Springer, 1988.

[49] Jacques Stern. A new identification scheme based on syndrome decoding. In *Annual International Cryptology Conference*, pages 13–21. Springer, 1993.

[50] The Classic McEliece team. Listing of Information Set Decoding related papers. https://classic.mceliece.org/papers.html.

[51] Pascal Véron. Improved identification schemes based on error-correcting codes. *Applicable Algebra in Engineering, Communication and Computing*, 8(1):57–69, 1997.

[52] David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

[53] Violetta Weger, Karan Khathuria, Anna-Lena Horlemann, Massimo Battaglioni, Paolo Santini, and Edoardo Persichetti. On the hardness of the Lee syndrome decoding problem. *Advances in Mathematics of Communications*, 2022.