Bittersweet Signatures: Bringing LWR to a Picnic for Hardware-Friendly MPC-in-the-Head – Extended Abstract –

Brieuc Balon[1], Gianluca Brian[2,3], Sebastian Faust[2], Carmit Hazay[4], Elena Micheli[2], and François-Xavier Standaert[1]

[1] UCLouvain, ICTEAM, Crypto Group, Louvain-la-Neuve, Belgium
[2] Technical University of Darmstadt, Germany
[3] ETH Zurich, Zurich, Switzerland
[4] Bar-Ilan University, Israel

Abstract. Post-quantum signature schemes are becoming increasingly important due to the threat posed by quantum computers to classical cryptographic schemes. Among the approaches considered in the literature, the MPC-in-the-head paradigm introduced by Ishai et al. (STOC'07) provides an elegant method for constructing zero-knowledge proofs by emulating Multi-Party Computation (MPC). This paradigm has become a versatile tool for designing cryptographic protocols, including post-quantum signature schemes. Building on this framework, we introduce Bittersweet signatures, a new class of schemes based on the Learning With Rounding (LWR) assumption that combine the strengths of two successful research directions: on the one hand, they retain the simplicity of the MPC-in-the-head paradigm underlying several post-quantum signatures; on the other hand, they establish a new lattice-based design relying on the LWR assumption, whose algebraic structure enables a particularly regular and parallelizable implementation and offers promising potential for leakage-resilience. Concretely, we exploit (almost) key-homomorphic pseudorandom functions (PRFs), that preserve pseudorandomness while supporting linear operations on keys, to obtain the highly regular computation pattern that naturally lends itself to parallel implementations. From a theoretical perspective, analyzing Bittersweet signatures requires addressing new challenges arising from the (carry) leakage introduced by almost key-homomorphic operations. We develop techniques to control this leakage while preserving the security guarantees of the MPC-in-the-head framework. The resulting constructions achieve competitive signature sizes and highlight a promising design space that combines conceptual simplicity, scalability, and suitability for hardware-oriented and leakage-resilient implementations. Beyond the concrete schemes, the abstractions introduced in this work also suggest several directions for further optimization and generalization.

1 Introduction

The rapid progress in quantum computing represents a major challenge to cybersecurity, as the exceptional processing power of future quantum machines could be used to break many traditional public-key cryptosystems [40].

In response, security agencies worldwide are urging the transition of critical infrastructure to post-quantum cryptography [18,38]. Designing post-quantum secure signature schemes is an important part of this standardization effort [35], which culminated with the selection of three first algorithms for standardization by the NIST in July 2022: Dilithium [22] and Falcon [27], whose security relies on the hardness of lattice problems, and SPHINCS$^+$ [3], whose security relies on cryptographic hash functions. Designing new post-quantum signature schemes remains of fundamental importance for several reasons. First, increasing the diversity of the hardness assumptions these algorithms rely on is needed to avoid "putting all eggs in the same basket". Second, the performances and cost of these algorithms remain a topic of concern, especially if security against implementation (*e.g.,* side-channel, fault) attacks is required [31,4,17,32]. Finally, the state of the art regarding signatures with advanced functionalities such as threshold- and multi-signatures [10], group signatures [16] and ring signatures [39], remains relatively sparse. This has prompted the NIST to open a call for new signature schemes, leading to the submission of 50 candidates in June 2023 [36].

Among the approaches considered for constructing signature schemes, the Multi-Party Computation-in-the-Head (MPCitH) paradigm introduced by Ishai *et al.* [30] at STOC'07 provides an elegant solution. This technique has become increasingly popular due to its high modularity and generality: 6 out of 14 submissions that have reached the second round of the latest NIST standardization process rely on this framework or extensions thereof [37].

At a high level, the MPCitH paradigm uses a Multi-Party Computation (MPC) protocol to build a zero-knowledge sigma protocol for a given hard relation. Typically, this relation consists of finding a secret key (the witness) that produces a specified input–output pair (the statement) under a pseudorandom function (PRF). In the sigma protocol, the prover emulates several parties locally by secret-sharing the witness among them and executes the MPC protocol "in its head". Each virtual party computes a share of the PRF evaluation, and the prover obtains the outputs of all parties, such that recombining these output shares yields the correct PRF value. The first message of the sigma protocol contains the output shares of all the parties, along with commitments to their input shares and internal computations usually called views. The verifier then challenges the prover to open a subset of these commitments, thereby checking their consistency and the correctness of the computed output. Once the sigma protocol is constructed, it can be converted into a signature scheme in the (quantum) random oracle model by using the Fiat-Shamir transform [26].

A central design choice in this framework is the selection of the PRF, as it largely determines the structure of the underlying MPC protocol. In particular, non-linear operations within the PRF typically result in more interaction among the simulated parties. This increased interaction translates into more complex and less parallelizable schemes, motivating the following question:

Can we find a PRF resulting in a minimal amount of interaction in the MPC protocol? What would the resulting signature scheme look like?

In this extended abstract, we address these questions by outlining Bittersweet signatures, which leverage the (almost) key homomorphism of a PRF based on the Learning With Rounding (LWR) assumption. We first recall the structure of MPCitH paradigm. We then detail the modifications required to instantiate it with our choice of PRF. We conclude by discussing instantiation issues and show that the conceptual simplicity of Bittersweet signatures translates into nicely parallel implementations which, despite not improving the most optimized schemes of the state of the art, have nice potential for leakage-resilience. Due to place constraints, we refer the reader to [5] for complete proofs and results.

2 Technical Overview

The starting point of Bittersweet signatures is the Picnic signature scheme [13], which we recall in Section 2.1. In a nutshell, we incorporate an (almost) key-homomorphic PRF in its design, resulting in a number of adjustments to the underlying sigma protocol.[5] We then illustrate our sigma protocol in the Section 2.2. Finally, we discuss the formal security analysis of Bittersweet signatures in 2.3. For this purpose, we mainly need to address the challenges raised by the (carry) leakage that almost key-homomorphic operations lead to.

2.1 MPCitH Signature Schemes, and the Picnic Case Study

In the following, we present the main ideas underlying MPCitH-based constructions, using the Picnic scheme [14] as a case study. At a high level, the design of a MPCitH signature scheme involves the following four main steps:

1. *Selecting the hard relation* between public and secret keys;
2. *Choosing the MPC scheme* used to compute the relation;
3. *Constructing the Sigma protocol* for the hard relation;
4. *Transforming the Sigma protocol into a signature scheme.*

A well-known example of an MPCitH signature scheme is the Picnic scheme [14]. We provide the high-level description of its non-optimized variant below.

1. *Selecting the hard relation:* The hard relation is given by the pairs $((x, y), k)$ satisfying the relation $y = F_k(x)$ where (x, y) are public values representing the input and output of the PRF F, and k is the PRF secret key. In the case of Picnic, this PRF is instantiated with the LowMC block cipher [2].
2. *Choosing the MPC scheme*: The scheme of Giacomelli *et al.*, on which Picnic relies, exploits any linear function decomposition [28]. This is a notion introduced to capture protocol virtualization as done in MPCitH. For their

[5] We only discuss modifications to the sigma protocol, as the final transformation into a signature scheme is obtained by applying the Fiat–Shamir transform with abort. However, these changes affect the final sigma protocol security guarantees. This requires leveraging Fiat Shamir security results relying on different assumptions compared to those used for Picnic. We discuss this better in Section 2.3.

particular instantiation, they consider a (2,3)-decomposition, which operates by performing secret sharing of the computations carried on the gates defining the arithmetic representation of F. First, the secret key k is additively shared into 3 shares k_1, k_2, k_3. Then, linear operations are conducted on each share locally, while multiplication gates require interaction. The information involved in the computation of each output share y_1, y_2, y_3 is finally stored in the corresponding views, denoted as $\mathsf{view}_1, \mathsf{view}_2, \mathsf{view}_3$.

3. *Constructing the Sigma protocol:* Picnic is based on the ZKBoo protocol of Giacomelli *et al.* [28], which works as follows. The prover first runs the MPC protocol described above on the key k, resulting in a triple of key shares k_1, k_2, k_3, output shares y_1, y_2, y_3, and views $\mathsf{view}_1, \mathsf{view}_2, \mathsf{view}_3$. Next, it commits to all the key shares and views and sends the commitments and output shares to the verifier. The verifier picks a couple of random indices i_1, i_2 and sends them to the prover. As a response, the prover reveals the key shares and views corresponding to the parties indexed by i_1, i_2. Finally, the verifier can check three properties: (1) the correctness of two output shares based on the opened key shares and views, (2) the validity of two commitments, and (3) that the output shares reconstruct to the PRF output y. This process can then be repeated t times with the same long-term key k and different random coins in order to reduce the soundness error.
4. *Transforming the sigma protocol into a signature scheme:* In the latest versions of Picnic [13], this is done via the Fiat-Shamir heuristic, both in the classic and quantum setting. The public key is the input-output PRF pair (x, y), the signing key is the PRF key k, and a signature consists of a sigma protocol transcript. To make it non-interactive, the challenge is computed by hashing the content of the public key, the first round of every iteration of the sigma protocol and the message to be signed. The security proof is in the quantum random oracle model and relies on several features of the sigma protocol: completeness, unpredictable commitments, special soundness, zero-knowledge, and quantum computationally unique responses.

2.2 Our Sigma Protocol

The following paragraphs progressively introduce the necessary modifications to the sigma protocol underlying Picnic, eventually leading to the description of the Bittersweet signature scheme. Figure 1 summarizes our scheme description, highlighting the changes with Picnic in different colors.

(i) The impact of key-homomorphic PRFs in MPCitH schemes. We begin by observing that the size and performance of Picnic signatures depend significantly on the number of multiplication gates in the circuit representation of the PRF F (since it affects the running time of the MPC protocol and the size of the views). This observation motivates the use of key-homomorphic PRFs [12] as a building block for MPCitH-based signatures. Key-homomorphic PRFs allow the computation on each key share independently, *i.e.*, satisfy $F_k(x) = F_{k_1}(x) + \ldots + F_{k_d}(x)$ for every additive sharing $k_1, \ldots, k_d$ of the key k. In other words, they

imply a "non-interactive MPC" scenario where each party contributes a single message to the MPC protocol. Consequently, a key homomorphism eliminates the need for large views or interaction between the virtual parties, allowing one to possibly parallelize this part of the computation.

While this intuition holds regardless of the details of the key-homomorphic PRF, its specific choice affects security and efficiency. This is because each signature contains many PRF evaluations, one for each party and each iteration of the sigma protocol. Therefore, we adopt the key-homomorphic PRF introduced by Boneh *et al.* [12], which is based on the LWR assumption [6]. Our primary motivation for choosing this scheme is its conceptual simplicity and, looking ahead to future work, its potential for leakage-resilient implementations via masking [23]. In this scheme, the PRF key is a random vector $\mathbf{k} \in \mathbb{Z}_{2^q}^n$, the input is a random matrix $\mathbf{X} \in \mathbb{Z}_{2^q}^{m \times n}$, and the output is computed as $\mathbf{y} = F_{\mathbf{k}}(\mathbf{X}) = \lfloor \mathbf{X}\mathbf{k} \rfloor_{2^p} \in \mathbb{Z}_{2^p}^m$, where $\lfloor \mathbf{x} \rfloor_{2^p} = \lfloor \mathbf{x} \cdot \frac{2^p}{2^q} \rfloor$.[6] Instantiating the sigma protocol underlying Picnic with this PRF results in the changes marked in blue in Figure 1.

Using this PRF in turn requires further modifications to the scheme. The reason is that this LWR-based PRF only satisfies a weaker variant of key homomorphism, known in the literature as *almost* key homomorphism. For a function F and key k, it ensures that $|F_k(x) - (F_{k_1}(x) + \ldots + F_{k_d}(x))| \leq \delta$ for every additive sharing $k_1, \ldots, k_d$ of the key, where δ is a small error. For our LWR-based PRF, this error scales with $\log(d)$ and intuitively describes the carry information that is preserved when *summing-before-rounding* and lost when *rounding-before-summing*. Introducing a small reconstruction error translates into several technical and conceptual challenges in the analysis and design of the scheme, forming the core of our technical contribution. In the following paragraphs, we outline these challenges and explain how we address them.

(ii) A modified verification check and its impact on the security of the hard relation. To ensure completeness, we include a small error tolerance in the verification check of the output shares, therefore modifying check (3) of Picnic (*cf.* Item 3). Instead of checking the consistency of the output shares as $\mathbf{y}_1 + \ldots + \mathbf{y}_d = \mathbf{y}$, we check that $\mathbf{y} - (\mathbf{y}_1 + \ldots + \mathbf{y}_d) \leq \delta$ component-wise. Beyond completeness, we also target special soundness, a form of extractability for sigma protocols. To achieve it, we introduce a tolerance in the hard relation as well, defining it as $\mathcal{R} = \{((\mathbf{X}, \mathbf{y}), \mathbf{k}) : |\mathbf{y} - F_{\mathbf{k}}(\mathbf{X})| \leq \delta\}$. Since the distance is measured component-wise and the vector $\mathbf{y} - F_{\mathbf{k}}(\mathbf{X})$ has m components, this relation is $(2\delta + 1)^m$ times easier to break compared to a strict equality check (see Lemma 1 in [5] for the security bound). Interestingly, we reduce this factor to 3^m when considering a setting where all-but-one shares are revealed in each protocol iteration, modifying both the relation $\mathcal{R}$ and check (3) accordingly. This is because having $d-1$ additive shares out of d can be seen as having 1 additive share out of 2. In practice, we consider the relation $\mathcal{R} = \{((\mathbf{X}, \mathbf{y}), \mathbf{k}) : |\mathbf{y} - F_{\mathbf{k}}(\mathbf{X})| \leq 1\}$ and modify check (3) as follows. The verifier first recombines the opened key shares

[6] Formally, our analysis only requires *weak* PRF, and we do not need to hash the input with a random oracle as Boneh *et al.* in order to get a PRF. We nevertheless use a hash function in our concrete instances, for performance reasons.

into their sum for each iteration j, which we denote with $\mathbf{k}_{rec}^{(j)} = \sum_{i \in \mathcal{I}^{(j)}} \mathbf{k}_i^{(j)}$, where $\mathcal{I}^{(j)}$ is a subset of $d-1$ distinct indices. Then, he verifies almost key homomorphism with the only unopened output share, denoted with $\mathbf{y}_{nop(j)}^{(j)}$. The verification check finally becomes $\mathbf{y} - (F_{\mathbf{k}_{rec}^{(j)}}(\mathbf{X}) + \mathbf{y}_{nop(j)}^{(j)}) \leq 1$. This change is illustrated and colored in green in Figure 1, where we use the symbol $\tilde{\mathbf{y}}^{(j)}$ in order to denote the approximated output $F_{\mathbf{k}_{rec}^{(j)}}(\mathbf{X}) + \mathbf{y}_{nop(j)}^{(j)}$.

(iii) Average-case completeness and zero-knowledge. The standard definition of zero-knowledge quantifies over all statement-witness pairs $((\mathbf{X}, \mathbf{y}), \mathbf{k})$ in the relation, which implies that privacy holds even if the distinguisher knows the witness. This creates an issue with our simulation due to having almost key homomorphism: while the quantity $\mathbf{y} - \mathbf{X}\mathbf{k}$ is deterministic for the distinguisher, it remains hidden from the simulator, which can only guess it. Hence, a distinguisher that knows $\mathbf{k}$ could use the opened key shares to reconstruct each unopened share as $\mathbf{k} - \mathbf{k}^{(j)}$ and detect inconsistencies in a simulated transcript. To address this, we restrict ourselves to pairs generated through an external distribution, denoted as $\mathcal{D}_{\mathcal{X} \times \mathcal{W}}$. This notion, referred to as average-case zero-knowledge [9], resolves the simulation issue by ensuring that the distinguisher no longer has access to the witness. An orthogonal problem that arises with completeness further requires the switch to the average case definition. The problem arises because a statement-witness pair is in the relation if the absolute value difference between $\mathbf{y}$ and $F_{\mathbf{k}}(\mathbf{X})$ is less than 1 component-wise. However, honestly-generated proofs for pairs satisfying $|\mathbf{y} - F_{\mathbf{k}}(\mathbf{X})| = \mathbf{1}$ might not be accepted. This is because the verifier compares $\mathbf{y}$ with $\tilde{\mathbf{y}}$, which it computes locally based on the decommitments and the unopened output shares. Since the reconstruction of $\tilde{\mathbf{y}}$ can also differ from $F_{\mathbf{k}}(\mathbf{X})$ by a factor $\mathbf{1}$, by the triangle inequality the difference between $\mathbf{y}$ and $\tilde{\mathbf{y}}$ can be 2 for some components, resulting in rejection. Therefore, we restrict $\mathcal{D}_{\mathcal{X} \times \mathcal{W}}$ to sample from the set of instances in which $\mathbf{y} = F_{\mathbf{k}}(\mathbf{X})$. These changes do not translate into modifications to the sigma protocol itself, but are relevant to precisely characterize its achieved guarantees. Looking ahead, this will not pose a problem for the security of our signature scheme because it is defined over honestly generated keys, and therefore, it suffices to generate the key pairs with $\mathcal{D}_{\mathcal{X} \times \mathcal{W}}$. Furthermore, the security of Fiat-Shamir signatures is often proved for sigma protocols with constraints on the statement-witness distribution (cast as identification schemes), and thus will adapt to our security definition.

(iv) The carry leakage information and the abort condition. The main technical challenge of this work is to control the amount of information leaked by the sigma protocol transcript. Intuitively, this is because every iteration j of the sigma protocol provides an eavesdropper with a noisy version $\mathbf{z}^{(j)}$ of $\mathbf{X}\mathbf{k}$, computable from the sum $\mathbf{k}^{(j)}$ of the opened key shares $(\mathbf{k}_i^{(j)})_{i \in \mathcal{I}^{(j)}}$ and the unopened output share $\mathbf{y}_{nop(j)}^{(j)}$ as

$$\mathbf{z}^{(j)} = \mathbf{X}\mathbf{k}^{(j)} + \frac{2^q}{2^p}\mathbf{y}_{nop(j)}^{(j)} \in \left[\mathbf{X}\mathbf{k} - \left(\frac{2^q}{2^p} + 1\right)\mathbf{1}, \mathbf{X}\mathbf{k}\right].$$

By averaging such samples over enough iterations, an eavesdropper could eventually isolate $\mathbf{Xk}$ and learn the secret, and thus break the zero-knowledge of the scheme. Interestingly, we show that an abort-based approach allows us to control the amount of information leaked.[7] Before introducing this approach, we look at this transcript leakage from a different perspective, taking the two-party setting ($d = 2$) with a single iteration ($t = 1$) and scalars ($m = n = 1$) as a use case. Assume that the first key share k_1 is open, and that k_2 stays secret. We consider the following decomposition for any value $v \in \mathbb{Z}_{2^q}$ and any $p' < q$: we denote with $\text{MSBs}^{(p')}(v) = \lfloor v \rfloor_{2^{p'}}$ the p' most significant bits output by the rounding function, and with $\text{LSBs}^{(q-p')}(v)$ the remaining $q - p'$ bits, so that

$$v = \text{MSBs}^{(p')}(v) \parallel \text{LSBs}^{(q-p')}(v).$$

We observe that we can rewrite the public value $y = \lfloor Xk \rfloor_{2^{p'}}$ as $\text{MSBs}^{(p')}(Xk_1 + Xk_2)$, which in turn can be expressed as a sum of the output shares y_1, y_2 plus a carry c propagating from the LSBs as follows:

$$\begin{array}{lcccc} Xk_1 & : & y_1 & \parallel & \text{LSBs}^{(q-p')}(Xk_1) \quad + \\ Xk_2 & : & y_2 & \parallel & \text{LSBs}^{(q-p')}(Xk_2) \quad = \\ Xk & : & y_1 + y_2 + c & \parallel & * \end{array}$$

The main observation is that $\text{LSBs}^{(q-p')}(Xk_2)$ needs to stay secret to ensure zero-knowledge. Yet, the carry c depends on it. Since $c = y - (y_1 + y_2)$ is publicly computable, it constitutes *public* and *secret-dependent* information. However, we observe that an abort-based approach allows restricting this information to the α most significant bits of $\text{LSBs}^{(q-p')}(Xk_2)$. To model this, we consider an alternative decomposition including an α–bit buffer between the LSBs and MSBs. That is, for a value $v \in \mathbb{Z}_{2^q}$ and for p', α such that $p' + \alpha < q$, we rewrite

$$v = \text{MSBs}^{(p')}(v) \parallel \text{buffer}^{(\alpha)}(v) \parallel \text{LSBs}^{(q-(p'+\alpha))}(v).$$

We assume that the LWR-based PRF still rounds up to p' bits, meaning that only the MSBs are public for both the secret value Xk and its shares (Xk_1, Xk_2). However, we make the prover's abort condition also depend on the buffer. To explain our intuition, we first assume just one bit of buffer, namely $\alpha = 1$. In this case, whenever $\text{buffer}^{(1)}(Xk_1) = \text{buffer}^{(1)}(Xk_2)$, the carry c into the MSBs depends only on the buffer bits and is independent of the LSBs. Aborting when this does not happen prevents leakage from the LSBs and makes the abort event only depend on the buffer. This condition is generalized to $\alpha > 1$ by checking that the two quantities $\text{buffer}^{(\alpha)}(Xk_1), \text{buffer}^{(\alpha)}(Xk_2)$ coincide in at least one bit position. The same reasoning extends to multiple parties, vector components, and iterations by replacing k_1, k_2 with $\mathbf{k}^{(j)}_{rec}, \mathbf{k}^{(j)}_{nop(j)}$, and performing this check component-wise for every iteration. Since the zero-knowledge simulator does not

[7] A similar technique is used in the analysis of the NIST candidate Dilithium [22].

have access to $\mathbf{Xk}^{(j)}_{nop(j)}$, we reformulate this check using buffer($\mathbf{Xk}$) instead. We do so by inverting the sum into a subtraction $\mathbf{Xk}^{(j)}_{nop(j)} = \mathbf{Xk} - \mathbf{Xk}^{(j)}_{rec}$.

The above requires studying a *borrow* instead of a carry, but the logic is the same: we need to ensure that the borrow affecting the MSBs only depends on the buffers, and not on the LSBs. This holds whenever $\text{buffer}^{(\alpha)}(\mathbf{Xk})$, $\text{buffer}^{(\alpha)}(\mathbf{Xk}^{(j)}_{rec})$ differ in every vector component for every iteration. Therefore, we abort if these α bits coincide in any component at any iteration of the protocol.

These considerations lead to the pink modifications in Figure 1, where $p' = p - \alpha$ corresponds to the public MSBs output by the rounding function $F^{(p-\alpha)}$ and the α LSBs of the p–rounding correspond to the buffers being checked. This buffer-dependent abort makes the security rely on p–LWR rather than just $(p-\alpha)$–LWR, even though the α bits of buffer of the secret values $\mathbf{Xk}$, $(\mathbf{Xk}^{(j)}_{nop(j)})_j$ are never explicitly revealed. This happens because the zero-knowledge simulator needs the additional α bits of buffer of $\mathbf{Xk}$ to decide when to abort.

Since the abort condition excludes a $\approx t \cdot m \cdot 2^{-\alpha}$ fraction of the transcripts, we can increase the parameter α to make the abort probability arbitrarily small. We discuss in Section 3 that combining such an increase with reasonable heuristics can significantly reduce the transcript size, and therefore also the size of Bittersweet signatures. We also propose a formal approach to the carry leakage in appendix A of [5]. There, we show that the carry leakage information $\mathbf{c}^{(j)}$ is as informative as the averaging the samples $\mathbf{z}^{(j)}$ as described above. We finally connect these quantities to two new variants of the LWR problem and prove that, under certain conditions, both variants reduce to standard LWR.

2.3 Security Analysis of Bittersweet Signatures

To obtain a signature scheme from our sigma protocol, we use the Fiat Shamir [26] with aborts [33] transform (recalled in Appendix B). This is a standard technique for MPCitH-based schemes. An important difference from typical MPCitH schemes, however, is that the underlying sigma protocols usually provide *statistical* zero-knowledge, whereas our construction achieves only *computational* zero-knowledge. This is a consequence of the carry leakage: a transcript reveals α additional bits of the LWR instance, which only computationally hides the underlying key. Therefore, we need to achieve signature unforgeability under weaker assumptions. Moreover, the carry leakage requires the use of aborts, which must also be accounted for in the security analysis. For these reason, we consider an alternative proof roadmap that we outline in this section.

Luckily, there is a vast literature on the post-quantum analysis of the Fiat-Shamir transform under different assumptions. All we need to do is to slightly adjust existing results to match the security notions achieved by our scheme. We first discuss our sigma protocol security guarantees, and then show how they imply signature unforgeability using simple variants of results from the literature. When stating the guarantees achieved by our sigma protocol, we always consider the relation

$$\mathcal{R} = \{((\mathbf{X}, \mathbf{y}), \mathbf{k}) : |\mathbf{y} - F^{(p-\alpha)}_{\mathbf{k}}(\mathbf{X})| \leq 1\}.$$

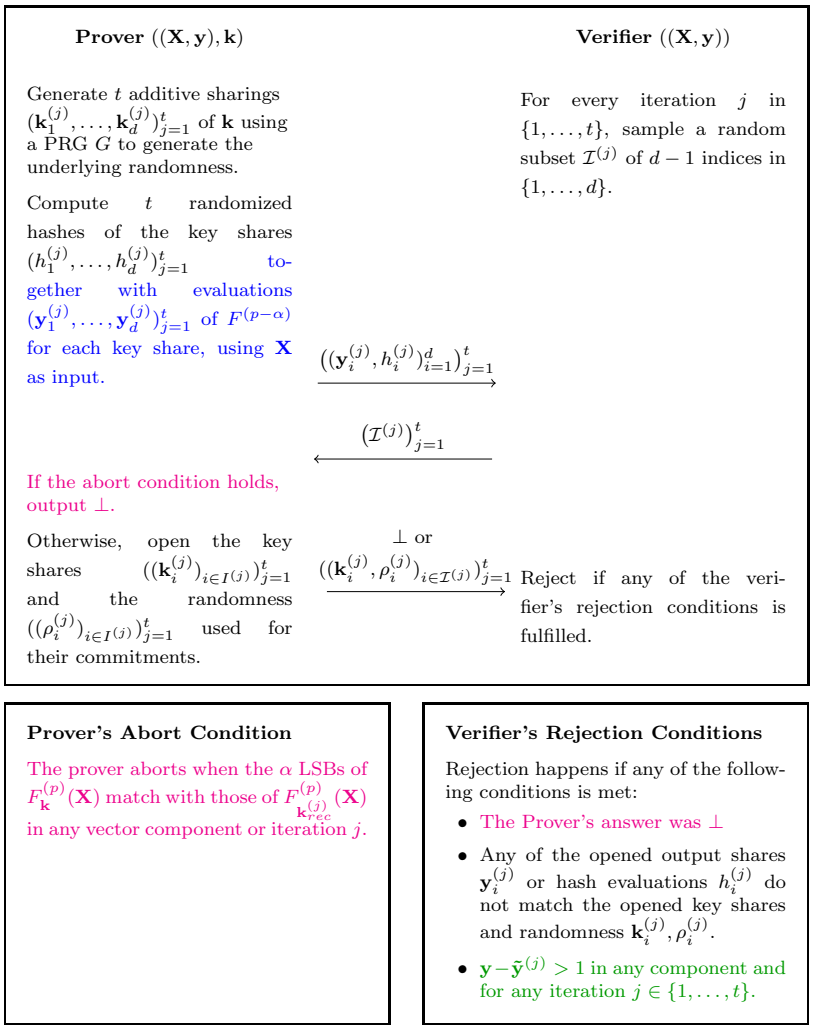

Fig. 1. This figure represents our sigma protocol. $F^{(p)}$ and $F^{(p-\alpha)}$ denote the LWR-based PRF that, upon input $\mathbf{X}$ and key $\mathbf{k}$, output respectively $\lfloor\mathbf{Xk}\rfloor_{2^p}$ and $\lfloor\mathbf{Xk}\rfloor_{2^{p-\alpha}}$. $\mathbf{k}_{rec}^{(j)}$ denotes the sum of the opened key shares for every iteration j, namely $\sum_{i\in\mathcal{I}^{(j)}}\mathbf{k}_i^{(j)}$. The symbol $\tilde{\mathbf{y}}^{(j)}$ represents the noisy output reconstructible from the transcript at each iteration, namely $F_{\mathbf{k}_{rec}^{(j)}}^{p-\alpha}(\mathbf{X})+\mathbf{y}_{nop(j)}^{(j)}$, where $nop(j)$ corresponds to the index of the unopened output share at iteration j. The main modifications compared to Picnic are marked with different colors. The modifications in blue are discussed in Paragraph (i), the ones in green in Paragraph (ii), and the pink ones in Paragraph (iv).

The definitions of the notions mentioned in this section are given in Appendix A.

Average-case completeness. As long as the abort condition is not triggered and the statement-witness pair is generated with $\mathcal{D}_{\mathcal{X}\times\mathcal{W}}$, an honestly-generated proof always passes the verification checks. This makes the completeness error only depend on the probability of aborts. In Theorem 4 of [5], we upper bound this probability with

$$tm \cdot 2^{-\alpha} + t \cdot \epsilon_{\mathsf{PRG}} + \epsilon_{\mathsf{LWR}}^{(p)},$$

assuming that G is an ϵ_{PRG}–PRG and that LWR holds with error $\epsilon_{\mathsf{LWR}}^{(p)}$ when rounding q bits to p bits. Recall that the abort condition occurs when, in any of the t iterations and for any of the m components, the α LSBs of $F_{\mathbf{k}}^{(p)}(\mathbf{X})$ match with those of $F_{\mathbf{k}_{rec}^{(j)}}^{(p)}(\mathbf{X})$. By reducing t times to the PRG and once to LWR, we can ensure that these vectors are pseudorandom and independent of each other. This reduces the problem to upper bounding the probability that $t \cdot m$ α-bit strings equal a random one, which is at most $t \cdot m \cdot 2^{-\alpha}$ by union bound.

Average-case accepting multi-honest-verifier zero-knowledge. This notion is satisfied when an adversary observing N non-aborting transcripts for a statement-witness pair hardly gets information about the secret key, so long as the pair is generated according to the distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{W}}$. In Theorem 5 of [5], we prove that this property holds with simulation error at most

$$Nt\epsilon_{\mathsf{Hide}} + \frac{2Nt\epsilon_{\mathsf{PRG}} + 4\epsilon_{\mathsf{LWR}}^{(p)}}{1 - Ntm \cdot 2^{-\alpha}} + 2\gamma^{\omega},$$

where ϵ_{Hide} upper bounds the distinguishing advantage for the input of the randomized hash function given its output,[8] ω is any value in $\mathbb{N}$ related to the (polynomial) runtime of the simulator, and $\gamma < 1$. As anticipated in Paragraph (iv) of Section 2.2, this is complicated by the only almost key-homomorphic PRF and the resulting carry leakage. Hence, we motivate it with a proof sketch.

Proof (sketch). This variant of zero-knowledge is formalized by the definition of an algorithm Sim that does not have access to the witness, but produces N non-aborting transcripts that are indistinguishable from their honestly generated, non-aborting, counterparts. To define Sim, we first present an algorithm Sim^* able to simulate a full transcript without knowledge of the secret key, yet with a rather high probability of aborting. Then, we construct Sim from Sim^* using a common technique from the literature: Sim simply runs Sim^* up to ω times and stops at the first non-aborting execution, drastically reducing the probability of aborts. Concretely, Sim^* takes as input a statement $\mathbf{X}, \mathbf{y}$ generated with $\mathcal{D}_{\mathcal{X}\times\mathcal{W}}$ and produces N transcripts for it in the following way:

– The Sim^* algorithm samples a random challenge $\mathcal{I}_\iota^{(j)}$ for each sigma protocol execution ι and iteration j of these protocol executions.

[8] In the literature, this relates to the *hiding* property of the *canonical commitment scheme*, which is proven in the random oracle model (*cf.* Lemma 17 of [41]).

- It sets the key shares as a (pseudorandom) sharing of a random key.
- It computes the output shares and the hashes for the opened parties by just evaluating the LWR-based PRF and the hash over these key shares.
- The hashes for the unopened shares are then computed by (cryptographically) hashing uniformly random and independent values.
- It samples a uniformly random α-bit string, which we call $\mathbf{r}$. The simulator will use this string to replace the α-bit buffer of $\mathbf{Xk}$.
- The output shares for the unopened parties are computed based on the buffers to ensure that the output shares sum up to $\mathbf{y}$. For every protocol execution ι and iteration j, the simulator first computes the borrow b as

$$b = \begin{cases} 1 & \text{if } \mathbf{r} < \text{buffer}^{(\alpha)}(\mathbf{Xk}_{rec}), \\ 0 & \text{otherwise,} \end{cases}$$

  using the simulated buffer $\mathbf{r}$ for $\mathbf{Xk}$ and the sum of the open key shares $\mathbf{k}_{rec}$ for that specific iteration and execution. Then, the output share for the unopened party is defined as $\mathbf{y}_{nop} = \mathbf{y} - \mathbf{y}_{rec} - b$. This follows the intuition that, when the prover's abort condition is not triggered, the borrow $\mathbf{b}$ only depends on the buffers. Moreover, the unopened output share satisfies $\mathbf{y}_{nop} = \mathbf{y} - \mathbf{y}_{rec} - \mathbf{b}$, as represented below:

$$\begin{array}{lcccccc} \mathbf{Xk} & : & \mathbf{y} - \mathbf{b} & \| & \text{buffer}^{(\alpha)}(\mathbf{Xk}) & \| & * \; - \\ \mathbf{Xk}_{rec} & : & \mathbf{y}_{rec} & \| & \text{buffer}^{(\alpha)}(\mathbf{Xk}_{rec}) & \| & * \; = \\ \hline \mathbf{Xk}_{nop} & : & \mathbf{y} - \mathbf{y}_{rec} - \mathbf{b} & \| & * & \| & * \end{array}$$

- The prover's abort condition is checked as in the original experiment for every execution, using the random vector $\mathbf{r}$ instead of the α-bit buffer of $\mathbf{Xk}$.

We now define a sequence of hybrids to prove that N executions of Sim^* are indistinguishable from N (possibly non-accepting) sigma protocol executions.

H_1: This first experiment is defined as the original experiment, but the crypotgraphic hash of each unopened share is computed as in Sim^*.

H_2: This experiment is defined as H_1, where the unopened output are computed as in Sim^*, but using $\text{buffer}^{(\alpha)}(\mathbf{Xk})$ instead of the random vector $\mathbf{r}$.

H_3: This experiment is defined as H_2, but key shares are computed as in Sim^*.

H_4: This experiment is defined as H_3, but, as in Sim^*, the prover's abort condition is computed using the random vector $\mathbf{r}$ instead of $\text{buffer}^{(\alpha)}(\mathbf{Xk})$. The output shares are also computed as in Sim^*, but there the LWR output $\mathbf{y}$ is further replaced with a uniformly random $(p-\alpha)$–bit vector $\mathbf{y}^*$.

The first hybrid H_1 and the original experiment are indistinguishable up to a factor $Nt\epsilon_{\mathsf{Hide}}$. Indeed, any adversary that distinguishes between the two would necessarily break the hiding property of at least one unopened hash value. Such a break would allow the adversary to recover the corresponding underlying key and combine it with the opened keys to test consistency with the LWR instance.

The hybrids H_2 and H_1 are identically distributed, but only for non-aborting transcripts. This happens because a non-aborting transcript ensures that each borrow $\mathbf{b}$ only depends on the buffers. Otherwise, the unopened output shares might be computed differently in the two experiments.

To prove indistinguishability for H_3 and H_2, we recall how a pseudorandom encoding is generated: one samples $d-1$ PRG seeds, sets the first $d-1$ key shares $\mathbf{k}_1, \ldots, \mathbf{k}_{d-1}$ as PRG evaluations of these seeds, and the last key share $\mathbf{k}_d$ as $\mathbf{k} - \sum_{i=1}^{d-1} \mathbf{k}_i$. When the opened set of parties coincides with $\{1, \ldots, d-1\}$, the two experiments compute the key shares identically. Whenever the d–th share is opened instead, H_2 computes the secret key-dependent value $\mathbf{k} - \sum_{i=1}^{d-1} \mathbf{k}_i$, while H_3 does the same but with a random unrelated key $\mathbf{k}^*$, namely sets $\mathbf{k}_d = \mathbf{k}^* - \sum_{i=1}^{d-1} \mathbf{k}_i$, making $\mathbf{k}_d$ uniformly random. However, a standard reduction to the PRG proves them indistinguishable. Since this reasoning must be repeated for every iteration and execution, this hybrid hop contributes a factor $Nt\epsilon_{\mathsf{PRG}}$.

The hybrids H_4 and H_3 are $\epsilon_{\mathsf{LWR}}^{(p)}$–indistinguishable assuming LWR with p–bit outputs. It holds because H_3 operates over a LWR-based PRF output $\mathbf{y} \parallel \mathrm{buffer}^{(\alpha)}(\mathbf{Xk})$, while H_4 replaces it with a uniformly random vector $\mathbf{y}^* \parallel \mathbf{r}$.

Finally, H_4 is indistinguishable from the ideal experiment using Sim^*, still because of the LWR assumption. This holds because H_4 uses the uniformly random p–bit vector $\mathbf{y}^* \parallel \mathbf{r}$, while Sim^* uses the real PRF output for the first $p-\alpha$ bits, *i.e.*, $\mathbf{y} \parallel \mathbf{r}$. This is also obtained thanks to a standard reduction to the LWR assumption, contributing an additional factor $\epsilon_{\mathsf{LWR}}^{(p-\alpha)} \leq \epsilon_{\mathsf{LWR}}^{(p)}$.

To prove *accepting* zero-knowledge, we need to recombine the factors provided by the hybrid hops, conditioning on the event **Success** where abort is not triggered. Since this event evolves across different experiments, we provide a technical lemma (*cf.* Lemma 18 in [5]) that relates the distinguishing advantage for two experiments conditioned on an event to both the probability of that event and the advantage for the two experiments without the condition. More in detail, the lemma guarantees that the advantage of a distinguisher with respect to two experiments is at most ϵ, then the advantage for the same experiments but conditioned on an event $\mathbf{E}$ can be upper bounded with $\frac{2\epsilon}{\Pr(\mathbf{E})}$. To use the lemma, we compute the probability of the event **Success** in the ideal experiment using Sim^*. Since this event is triggered by the equality of the random α bits $\mathbf{r}$ with an independent value in any vector component, execution or iteration, we get that $\Pr(\mathbf{Success}) \geq 1 - mNt2^{-\alpha}$. By applying the lemma, we deduce that the real world (an honest execution of the sigma protocol) and the ideal world (where we use Sim^* instead) are indistinguishable up to the factor $Nt\epsilon_{\mathsf{Hide}} + \frac{2Nt\epsilon_{\mathsf{PRG}} + 4\epsilon_{\mathsf{LWR}}^{(p)}}{1 - mNt2^{-\alpha}}$, conditioning on non-aborting transcripts. This does not yet conclude the proof, as *accepting* zero-knowledge requires a simulator generating only accepting transcripts, up to a small probability. This is the step where we move from Sim^* to Sim. Recall that Sim makes ω signature attempts and only aborts if all of them fail, which happens with probability γ^ω where $\gamma < 1$ is the probability of a single abort. Hence, the transcripts produced by Sim are indistinguishable from the accepting transcripts generated by Sim^* up to a factor $2\gamma^\omega$.

Special soundness. Our scheme satisfies 2–special soundness (*cf.* Theorem 6 in [5]), meaning that two accepting transcripts with the same first message and distinct challenges always enable the witness' reconstruction. This happens because having $s = 2$ accepting proofs for the same commitment and different challenges provides the extractor with a complete sharing of a candidate witness $\mathbf{k}$. Therefore, extraction can be finalized by reconstructing $\mathbf{k}$ from its sharing. The validity of each transcript with respect to the same statement and first message, together with the almost key homomorphism of the rounding function, imply that this extraction always provides a valid witness for the given statement.

Quantum computationally unique responses. A fundamental property of our sigma protocol is having quantum computationally unique responses, ensuring hardness in finding two accepting transcripts with the same first message, same challenge, but distinct responses. Looking ahead, this relates to a powerful attack to the unforgeability of the signature scheme: the attacker could observe an honestly generated signature (which is a proof), and forge a valid one by just changing the response. This attack is hard if the responses and the rest of the transcripts are related in a binding way. For this reason, as in many MPCitH–based schemes, we hash the response with an ϵ_{Coll}–collapsing hash function, and include the result in the first message. This guarantees that our sigma protocol has ϵ_{QCUR}–quantum computationally unique responses, where $\epsilon_{\mathsf{QCUR}} \leq t(d-1) \cdot \epsilon_{\mathsf{Coll}}$ (*cf.* Theorem 7 in [5]). Our argument supporting the bound is very similar to the one provided for Picnic1 in Corollary 26 of [21]. To prove quantum computationally unique responses, we must analyze the event where an adversary provides two different responses for the same first message and challenge, and the detection of a measurement in the computational basis. Notably, the response of our sigma protocol only contains preimages of the hash function. Therefore, the QCUR property reduces to the collapsingness of H on each of the $(d-1)t$ opened values. Using the bound on the concurrent composition of collapsing functions (Lemma 4, [25]), the theorem follows.

Unpredictable commitments. Finally, we prove that our scheme has unpredictable commitments. Looking ahead to the signature scheme, this property ensures that the input to the Random Oracle (RO) is sufficiently random, thereby guaranteeing that the RO-generated challenge is also random. This allows to relate the correctness of the signature to the completeness of the sigma protocol and to prevent the forger from crafting "comfortable" challenges. Similarly to other MPCith-based scheme, ours has highly unpredictable commitments. This is because each iteration involves the sampling of a new sharing, and each first message contains the evaluation of such shares with a collapsing hash function. In Theorem 8 of [5], we prove our scheme to have $\epsilon_{\mathsf{ComGuess}}$–unpredictable commitments, with

$$\epsilon_{\mathsf{ComGuess}} \leq \left(2\epsilon_{\mathsf{Coll}}\left(1 - \frac{1}{|\mathcal{S}|}\right) + \frac{1}{|\mathcal{S}|}\right)^{(d-1)t},$$

with $\mathcal{S}$ the size of the space from which the PRG seeds are sampled. To motivate this bound, we consider the probability that an honest prover outputs a

fixed first message *firstmes**. This happens only if all hash values match those in *firstmes**. In turn, this requires that either all the PRG seeds equal those underlying *firstmes**, or that some of the PRG seeds differ but still yield identical hash outputs. Producing the same hash from two different seeds corresponds to finding a collision, which happens with probability at most $2\epsilon_{\mathsf{Coll}}$ for ϵ_{Coll}– collapsing hash functions (cf. Lemma 24 of [41]). Combining the independent contributions of all seeds and applying the binomial theorem, we obtain the theorem bound.

Moving to signature unforgeability. To prove security of our signature scheme, we finally adjust existing results to match the security notions achieved by our sigma protocol. For *correctness*, we directly apply Theorem 6 of Devevey *et al.* [19] (recalled as Theorem 1 in [5]). In this theorem, the correctness error is computed based on the sigma protocol's probability of aborts and the min entropy of its first message. We then use Theorem 21 and 25 of Don *et al.* [21] to prove *unforgeability against no message attacks*. This is guaranteed under the following assumptions: the relation $\mathcal{R}$ is hard, the challenge space is sufficiently large, and the sigma protocol satisfies special soundness and has quantum computationally unique responses. One subtlety is that the results of [21] concern the abort-free setting, while our signature relies on Fiat-Shamir with aborts. However, aborts can only reduce the adversary's success probability in a no-message attack, so the proof naturally extends to the setting with aborts. We make all parameters explicit in Theorem 2 of [5]. Finally, we use a variant of Theorem 2 of Barbosa *et al.* [7] to lift unforgeability against no-message attacks to standard unforgeability, presented as Theorem 3 in [5]. It requires the underlying sigma protocol to satisfy average-case accepting multi honest-verifier zero-knowledge and have unpredictable commitments. The main difference is that Theorem 2 of [7] assumes statistical ZK, while we rely on the computational notion of multi HVZK. The proof of our Theorem 3 is similar to the one in [7], the only difference being the necessity to replace the real proofs with simulated ones all at once, which is possible in the computational setting thanks to the definition of *multi* HVZK.

3 Instantiation and Performance

We complete this extended abstract by examining the instantiation of Bittersweet signatures. We start with a more operational description, highlighting the parallel nature of their design, and conclude with performance evaluations.

3.1 Operational Specifications

Let us first define a function that outputs m LWR samples:

$$\mathsf{LWR}_{\mathbf{k}}(x) : \lambda \times \mathbb{Z}_{2^q}^n \to \mathbb{Z}_{2^{p-\alpha}}^m : x \mapsto \mathbf{y} = \lfloor \mathsf{G}(x) \cdot \mathbf{k} \rceil_{2^{p-\alpha}} ,$$

where $\mathsf{G}(x)$ is a function that expands the λ-bit input x into a $m \times n$ matrix $\mathbf{X} \in \mathbb{Z}_{2^q}^{m \times n}$, typically instantiable with an eXtendable Output Function (XOF) [34].

The Bittersweet signatures' blueprint is given in Figure 2, which illustrates the parallel nature of their design. It is based on the following steps, which essentially match the descriptions in the previous sections, with a few additional details (separation bits, salt, ...) mimicking what can be found in Picnic [15,13].

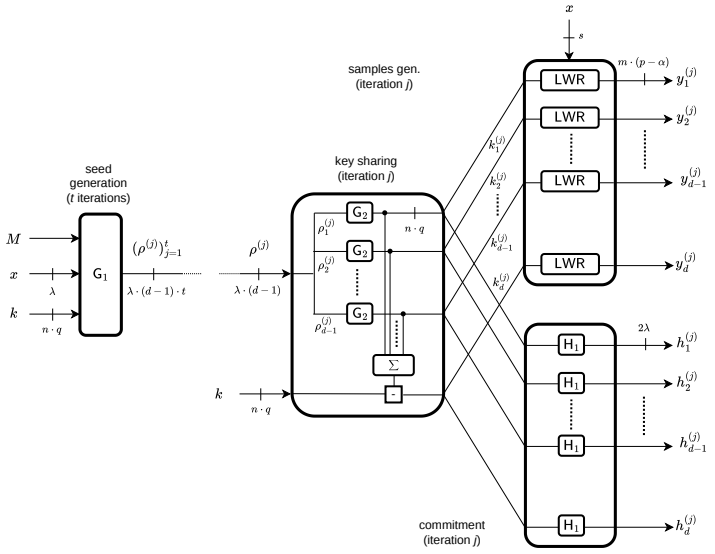

Fig. 2. Bittersweet signatures' blueprint.

Key generation. Pick up a plaintext $x \xleftarrow{\$} \{0,1\}^\lambda$ and a secret key $\mathbf{k} \xleftarrow{\$} \mathbb{Z}_{2^q}^n$ uniformly at random. Compute the public key $pk := (x, \mathbf{y})$ with $\mathbf{y} = \mathsf{LWR}_{\mathbf{k}}(x)$.

Seeds generation. Compute seeds $(\rho^{(j)})_{j=1}^t = \mathsf{G}_1(M, x, \mathbf{k}, salt)$, where M is the message to sign, x the plaintext part of the public key, $\mathbf{k}$ the secret (signature) key, the λ-bit *salt* is used to prevent multi-target attacks [20] and t is the number of iterations of the signature process. The resulting seeds have bitsize $\lambda \cdot (d-1) \cdot t$, with d being the number of parties. For each iteration j, the seed $\rho^{(j)}$ is split into $d-1$ equal parts denoted as $\rho_i^{(j)}$, $1 \leq i \leq d-1$, each of them having bitsize λ.

Key sharing. For each iteration j, $1 \leq j \leq t$ and for each party i, $1 \leq i \leq d-1$, compute expanded key shares $\mathbf{k}_i^{(j)} = \mathsf{G}_2(\rho_i^{(j)}, i, j, salt)$ and $\mathbf{k}_d^{(j)} = \mathbf{k} - \sum_{i=1}^{d-1} \mathbf{k}_i^{(j)} \bmod 2^q$, so that every key share $\mathbf{k}_i^{(j)}$, $1 \leq i \leq d$, $1 \leq j \leq t$, has bitsize $n \cdot q$.

Commitment. For each iteration j, $1 \leq j \leq t$, commit on the key share by computing $h_i^{(j)} = \mathsf{H}_1(\mathbf{k}_i^{(j)}, i, j, salt)$, $1 \leq i \leq d$, so that each $h_i^{(j)}$ has bitsize 2λ.

LWR samples generation. For each iteration j, $1 \leq j \leq t$, compute the output shares $\mathbf{y}_i^{(j)} = \mathsf{LWR}_{\mathbf{k}_i^{(j)}}(x)$, $1 \leq i \leq d$, so that each $\mathbf{y}_i^{(j)}$ has bitsize $m \cdot (p - \alpha)$.

Challenge. Compute the hash of the message to sign M, the commitments $h_{1:d}^{1:t}$, the output shares $\mathbf{y}_{1:d}^{1:t}$, the signer's public key $(x, \mathbf{y})$ and the salt as:

$$c = \mathsf{H}_2(\mathbf{y}_{1:d}^{1:t}, h_{1:d}^{1:t}, M, x, \mathbf{y}, salt).$$

Signature assembly. Assume $d = 2^\delta$ and $|h| = \delta t$ for simplicity. For each iteration j, denote the δ-bit index of the key share that is not opened as $nop(j)$, and the indexes of the $d-1$ key shares that are opened as $op(j)$. The signature σ is obtained by concatenating the *salt* with all the output shares $\mathbf{y}_{1:d}^{1:t}$ and commitments $h_{1:d}^{1:t}$ as well as the opened key shares $(\mathbf{k}_{op(j)}^{(j)})_{j=1}^{t}$.

$$\sigma = (salt \;||\; \mathbf{y}_{1:d}^{1:t} \;||\; h_{1:d}^{1:t} \;||\; (\mathbf{k}_{op(j)}^{(j)})_{j=1}^{t})$$

Aborts. For each iteration j, $1 \leq j \leq t$, compute $\mathbf{Xk}_{rec}^{(j)}$ and $\mathbf{Xk}$, this time rounded to p (rather than $p-\alpha$) bits. As explained in Section 2.3, abort if the α LSBs of these values match in any of the m vector components.

Verification. Derive the challenge c from the signature. For each iteration j, $1 \leq j \leq t$, recompute the commitments and output shares of the opened parties using $(\mathbf{k}_{op(j)}^{(j)})_{j=1}^{t}$. Then check that they correspond to the ones present in the signature using the indexing associated to the challenge. If this holds, compute the approximated output for each iteration by $\tilde{\mathbf{y}}^{(j)} = \mathsf{LWR}_{\mathbf{k}_{rec}^{(j)}}(x) + \mathbf{y}_{nop(j)}^{(j)}$ and if the inequality $\mathbf{y}^{(j)} - \tilde{\mathbf{y}}^{(j)} \leq 1$ holds component-wise, accept the signature.

Optimizations. We use the next optimizations to reduce the signature size. First, the output share of the opened parties $(\mathbf{y}_{op(j)}^{(j)})_{j=1}^{t}$ can be removed from the signature since they can be recomputed using the opened key shares $(\mathbf{k}_{op(j)}^{(j)})_{j=1}^{t}$ and the public key x. Second, the commitments of the opened parties $(h_{op(j)}^{(j)})_{j=1}^{t}$ can also be removed from the signature as they can be recomputed from the opened key shares $(\mathbf{k}_{op(j)}^{(j)})_{j=1}^{t}$. To guarantee that these opened key shares were correctly committed, we include the challenge c in the signature. This challenge acts as a "commitment to the commitments" [15], since the verifier checks that the recomputed commitments are consistent by verifying that the recalculated challenge matches the one present in the signature. Third, the size of the first $d-1$ key shares can be reduced by transmitting the corresponding seeds. Using a GGM-tree, only the root seed of the tree must be sent if the last party remains unopened. Otherwise, the last key share is sent with the sibling-path of the GGM tree associated to the unopened party, which consists of $\log_2(d)$ seeds. Fourth, the unopened output share $\mathbf{y}_{nop(j)}^{(j)}$ can be replaced by the error vector $\mathbf{e}^{(j)} = \mathbf{y}^{(j)} - \tilde{\mathbf{y}}^{(j)}$ in order to reduce its size from $m \cdot (p-\alpha)$ to $m \cdot 1$ bits.

Signature size. Using the different aforementioned optimizations, the averaged size (in bits) of Bittersweet signatures is:

$$\begin{aligned}
|\sigma| = t \cdot \left(\frac{d-1}{d} \cdot \left(n \cdot q + \lceil \log_2(d) \rceil \cdot \lambda \right) + \frac{1}{d} \cdot \lambda \right) \quad &\rightarrow \text{ seeds and key shares} \\
+ t \cdot m \quad &\rightarrow \text{ error vectors} \\
+ t \cdot 2\lambda \quad &\rightarrow \text{ commitments} \\
+ \lambda + t \cdot \lceil \log_2(d) \rceil \quad &\rightarrow \text{ salt and challenge}
\end{aligned}$$

3.2 Parameters' Choices and Performance

As explained in Section 2.2, the security of Bittersweet requires a secure LWR instance rounding to p bits, despite the specifications in this section round to $p' = p - \alpha$ bits. Informally, this is because each time a signature is produced, one value of the buffer can be rejected. As discussed in [5], it is possible to obtain more efficient instances (rounding to p' bits rather than p bits) by tolerating a small leakage on this buffer. Namely, we posit that the security of such LWR instances is not significantly affected if the number of rejected buffer values (which is equal to the number of signatures N_{Sign}) is small in front of 2^{α} (*e.g.,* $N_{\text{Sign}} = 2^{32}$ or 2^{64} in front of 2^{90}). Using the estimator of Albrecht et al. [1], this allows us to obtain the Bittersweet parameters summarized in Table 1.

Table 1. Bittersweet parameters set for NIST security levels L1, L3 and L5.

NIST Level	**Security** (bits) λ	**Parameters**								**Sizes** (Bytes)		
		LWR					MPCitH					
		q	p'	α	n	m	d	t	P_{abort}	$\lvert sk\rvert$	$\lvert pk\rvert$	$\lvert\sigma\rvert$
L1	128	96	3	90	11	747	32	26	$2^{-75.7}$	132	297	8 645
							256	16	$2^{-76.4}$	132	297	6 183
L3	192	96	3	90	18	1222	32	39	$2^{-74.4}$	216	483	20 602
							256	24	$2^{-75.2}$	216	483	14 622
L5	256	96	3	90	25	1697	32	52	$2^{-73.6}$	300	669	37 648
							256	32	$2^{-74.3}$	300	669	26 627

In order to highlight the interest of Bittersweet signatures' simple design for hardware implementations, we report FPGA performances and resources' utilization (using SHA3 and SHAKE as hash function and XOF) in Table 2.

Table 2. Hardware performance and resources utilization of Bittersweet on Xililinx Artix-7 (XC7A200T-3) platform. The maximal frequency of the designs is 125 MHz

NIST Level	d	**KeyGen**	**Sign**	**Verify**	**FPGA Utilisation**			
		Mcycles/ms	Mcycles/ms	Mcycles/ms	LUT	FF	DSP	BRAM
L1	32	0.015/0.12	1.06/8.48	0.91/ 7.28	15 891	9 242	180	60
L3		0.049/0.39	2.43/19.44	2.01/16.08	19 372	11 541	294	110
L5		0.095/0.76	4.54/36.32	3.78/30.24	21 170	12 254	410	144
L1	256	0.015/0.12	5.19/41.25	4.39/ 35.12	18 425	9 617	180	286
L3		0.049/0.39	11.79/94.32	10.03/ 80.24	22 352	12 171	294	384
L5		0.095/0.76	22.02/176.16	18.50/ 148.00	24 512	13 514	410	512

These figures confirm that Bittersweet natively leads to competitive signature sizes, especially for a scheme relying on non-structured lattice assumptions. The comparisons given in [5] further show that it leads to hardware performance gains for moderate software performance overheads when compared with state-of-the-art MPC-in-the-heads schemes, while admittedly lagging a bit behind recent proposals based on Threshold-Computation- or VOLE-in-the-head. We believe the potential of Bittersweet for leakage-resilient implementations should reduce this gap, which we leave as an interesting direction for further research. Not only because the key-homomorphic nature of its computations should make masking significantly simpler and more efficient than for MPC-in-the-head signatures based on standard symmetric primitives [23]. Also because the large modulus $Q = 2^{96}$ should make the recombination of the information leaked by different shares significantly more difficult, especially with inner product masking [24].

Acknowledgments. We are grateful to Martin Albrecht, Cecilia Boschini, Yu-Hsuan Huang, and Andreas Hülsing for our very insightful discussions. Carmit Hazay is supported by ISF grant No. 607/25. François-Xavier Standaert is a research director of the F.R.S.-FNRS. This work has been funded by the German Research Foundation (DFG) CRC 1119 CROSSING (project S7), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and by the European Research Concil (ERC) Consolidator Grant 101044770 (CRYPTOLAYER) and Advanced Grant 101096871 (BRIDGE). Views and opinions expressed in the paper are those of the authors and do not necessarily reflect those of the European Union nor the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

1. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.
2. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Berlin, Heidelberg, Apr. 2015.
3. J.-P. Aumasson, D. J. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, and B. Westerbaan. SPHINCS, 2017.
4. M. Azouaoui, O. Bronchain, G. Cassiers, C. Hoffmann, Y. Kuzovkova, J. Renes, T. Schneider, M. Schönauer, F.-X. Standaert, and C. van Vredendaal. Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR TCHES*, 2023(4):58–79, 2023.
5. B. Balon, G. Brian, S. Faust, C. Hazay, E. Micheli, and F.-X. Standaert. Bittersweet signatures: Bringing lwr to a picnic for hardware-friendly mpc-in-the-head –full version–. Cryptology ePrint Archive, Paper 2026/397, 2026. https://eprint.iacr.org/2026/397.

6. A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Berlin, Heidelberg, Apr. 2012.
7. M. Barbosa, G. Barthe, C. Doczkal, J. Don, S. Fehr, B. Grégoire, Y.-H. Huang, A. Hülsing, Y. Lee, and X. Wu. Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium. In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 358–389. Springer, Cham, Aug. 2023.
8. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
9. I. Berman, A. Degwekar, R. D. Rothblum, and P. N. Vasudevan. From laconic zero-knowledge to public-key cryptography - extended abstract. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 674–697. Springer, Cham, Aug. 2018.
10. G. Bleumer. Threshold Signatures. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1294–1296. Springer, 2011.
11. D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random oracles in a quantum world. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Berlin, Heidelberg, Dec. 2011.
12. D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. Key homomorphic PRFs and their applications. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Berlin, Heidelberg, Aug. 2013.
13. M. Chase, D. Derler, S. Goldfeder, J. Katz, V. Kolesnikov, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, X. Wang, et al. The Picnic Signature Scheme.
14. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1825–1842. ACM Press, Oct. / Nov. 2017.
15. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS*, pages 1825–1842. ACM, 2017.
16. D. Chaum and E. van Heyst. Group Signatures. In *EUROCRYPT*, pages 257–265. Springer, 1991.
17. K.-Y. Chen and J.-P. Chen. Masking floating-point number multiplication and addition of falcon first- and higher-order implementations and evaluations. *IACR TCHES*, 2024(2):276–303, 2024.
18. T. E. Commission. Commission Recommendation on a Coordinated Implementation Roadmap for the transition to Post-Quantum Cryptography, 2024.
19. J. Devevey, P. Fallahpour, A. Passelègue, and D. Stehlé. A detailed analysis of Fiat-Shamir with aborts. In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 327–357. Springer, Cham, Aug. 2023.
20. I. Dinur and N. Nadler. Multi-target attacks on the picnic signature scheme and related protocols. In *EUROCRYPT*, pages 699–727. Springer, 2019.
21. J. Don, S. Fehr, C. Majenz, and C. Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 356–383. Springer, Cham, Aug. 2019.

22. L. Ducas, T. L. Eike Kiltz, V. Lyubashevsky, G. S. Peter Schwabe, and D. Stehlé. CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation, 2017.
23. S. Dziembowski, S. Faust, G. Herold, A. Journault, D. Masny, and F.-X. Standaert. Towards sound fresh re-keying with hard (physical) learning problems. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 272–301. Springer, Berlin, Heidelberg, Aug. 2016.
24. S. Faust, L. Masure, E. Michel, H. H. Nguyen, M. Orlt, and F. Standaert. IP masking with generic security guarantees under minimum assumptions, and applications. In *ASIACRYPT (2)*, volume 16246 of *Lecture Notes in Computer Science*, pages 544–575. Springer, 2025.
25. S. Fehr. Classical proofs for the quantum collapsing property of classical hash functions. In A. Beimel and S. Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 315–338. Springer, Cham, Nov. 2018.
26. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, Aug. 1987.
27. P.-A. Fouque, J. Hoffstein, V. L. Paul Kirchner, T. P. Thomas Pornin, G. S. Thomas Ricosset, W. Whyte, and Z. Zhang. Falcon, 2017.
28. I. Giacomelli, J. Madsen, and C. Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In T. Holz and S. Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, Aug. 2016.
29. A. B. Grilo, K. Hövelmanns, A. Hülsing, and C. Majenz. Tight adaptive reprogramming in the QROM. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 637–667. Springer, Cham, Dec. 2021.
30. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In D. S. Johnson and U. Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.
31. M. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen. PQM4: Post-Quantum Crypto Library for the ARM Cortex-M4, 2019.
32. M. J. Kannwischer, A. Genêt, D. Butin, J. Krämer, and J. Buchmann. Differential power analysis of XMSS and SPHINCS. In J. Fan and B. Gierlichs, editors, *COSADE 2018*, volume 10815 of *LNCS*, pages 168–188. Springer, Cham, Apr. 2018.
33. V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, Dec. 2009.
34. A. Mittelbach and M. Fischlin. *The Theory of Hash Functions and Random Oracles - An Approach to Modern Cryptography.* Information Security and Cryptography. Springer, 2021.
35. NIST. NIST Asks Public to Help Future-Proof Electronic Information, 2016.
36. NIST. Post-Quantum Cryptography: Additional Digital Signature Schemes, 2023.
37. NIST. Status Report on the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process, 2024.
38. NSA. Commercial National Security Algorithm Suite 2.0, 2024.
39. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Berlin, Heidelberg, Dec. 2001.
40. P. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct. 1997.

41. D. Unruh. Computationally binding quantum commitments. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 497–527. Springer, Berlin, Heidelberg, May 2016.

A Standard Cryptographic Primitives

A.1 Digital signature schemes

We call *digital signature scheme* a triple of algorithms $(\mathsf{S.Gen}, \mathsf{Sign}, \mathsf{S.Verify})$ satisfying the following properties. The *key generation algorithm* $\mathsf{S.Gen}$ outputs a key pair (sk, vk), where sk is called *signing key* and is private, and vk is called *verification key* and is public. The *signing algorithm* Sign takes as input the signing key and a message msg and outputs a signature σ. The *verification algorithm* takes as input the verification key, a message and a signature and outputs a decision bit indicating acceptance or rejection. The signature and verification algorithm may have additional access to a random oracle.

We require *correctness*, meaning that for every key pair (sk, vk) and every message, a signature produced using sk is always accepted by $\mathsf{S.Verify}$ under vk. *Unforgeability* holds when no efficient attacker can forge a valid message-signature pair (msg^*, σ^*) for a fresh message msg^* without knowledge of the signing key. We call a signature unforgeable *against no message attacks* if the forgery is produced without observing any signature, otherwise we assume the attacker can observe signatures for a polynomial amount of chosen messages.

A.2 Sigma protocols

A *Sigma protocol* for $\mathcal{R}$ is an interactive protocol between two parties, called the *prover* and the *verifier*. The prover is modeled as a couple of probabilistic polynomial-time algorithms $\Sigma.\mathsf{Prove} = (\Sigma.\mathsf{Prove}_1, \Sigma.\mathsf{Prove}_2)$, which are assumed to share state. The verifier $\Sigma.\mathsf{Verify}$ is a deterministic polynomial-time algorithm that outputs a decision bit. Sigma protocols always consist of three rounds. In the first round, the prover generates a first message $firstmes \xleftarrow{\$} \Sigma.\mathsf{Prove}_1(x, w)$. Next, in the second round, the verifier answers with a uniformly random generated challenge ch. In the third round, the prover publishes a response $resp \xleftarrow{\$} \Sigma.\mathsf{Prove}_2(ch)$. To conclude, the verifier runs $\Sigma.\mathsf{Verify}$ on inputs$(x, firstmes, ch, resp)$ and outputs the resulting decision bit. We call *transcript* of the protocol the triple $(firstmes, ch, resp)$. We next introduce the properties that are presented formally in Section 3.4 of [5].

Completeness ensures that honestly generated proofs for statements in the language are accepted by the verifier, unless with a small probability named *completeness error*. Similarly to [9], we consider *average-case* variant of this notion, where completeness is required to hold only for statement–witness pairs sampled according to a given distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{W}}$.

Honest-verifier zero-knowledge (HVZK) guarantees that the transcript of an execution where the verifier issues a random challenge (and thus behaves honestly) reveals no more information on the witness that what could be derived

from the statement alone. Formally, this is captured by the existence of an efficient simulator that, given only the statement (and not the witness), produces transcripts indistinguishable from honestly generated ones. If the simulated and real transcripts are statistically indistinguishable, we obtain *statistical* HVZK; if indistinguishability holds only against efficient distinguishers, we obtain *computational* HVZK. We call *simulation error* the corresponding distinguishing advantage. Our analysis in Section 2.3 requires considering an unusual flavor of this notion, named *average-case accepting multi* HVZK. We unpack this notion word by word in the following. *Average-case* refers to the same context as considered for completeness, where the statement-witness pair used to produce the (real or ideal) transcript comes from a given distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{W}}$. This weakens the adversaries considered in the experiment for ZK: their choices cannot depend on the internal coins used for sampling the statement, but only on the statement itself. The *accepting* property comes from [7] in the context of sigma protocols with aborts, and requires the existence of a simulator whose output is indistinguishable from honestly-generated transcripts that are always accepting. Finally, we incorporate the *multi*-HVZK variant from [29], where HVZK is required to hold across multiple transcripts for the same statement-witness pair.[9]

The property of *s-special soundness* ensures the existence of an algorithm, called an *extractor*, that can recover a witness for a given statement from s transcripts, provided that each transcript is valid, the transcripts share the same first message, and their challenges are pairwise distinct.

Commitment unpredictability concerns the probability of guessing the sigma protocol's first message. It holds if, when averaging over statement–witness pairs sampled according to $\mathcal{D}_{\mathcal{X}\times\mathcal{W}}$, the probability of correctly guessing the most likely first message *firstmes* is upper bounded by a small value $\epsilon_{\mathsf{ComGuess}}$.

To describe *quantum computationally unique responses*, we begin with its classical counterpart. The classical notion informally captures the idea that, once the first message and the challenge of a sigma protocol are fixed, there is essentially only one valid response that can be produced efficiently. This is extended to the post-quantum setting by requiring that if an attacker produces accepting superpositions $(\mathit{firstmes}, \mathit{ch})$ and resp, then it cannot determine whether resp was measured unless with a small probability ϵ_{QCUR}.

A.3 Pseudorandom Generators

A pseudorandom generator (PRG) is a deterministic polynomial-time algorithm G that takes as input a uniformly random seed and expands it into a longer string that appears random to any efficient observer. We denote by ϵ_{PRG} the maximum advantage of a probabilistic polynomial-time (PPT) distinguisher in distinguishing the output of G from a truly uniform string of the same length.

[9] As noted in [29], this stronger variant follows from statistical HVZK, but not from computational HVZK in the context of average-case schemes.

A.4 Pseudorandom Functions and LWR

Let $F = (F_k)_k$ be a family of efficiently computable functions with the same input and output space. We call F a *q–pseudorandom function* (PRF) if the evaluations of F over a random key and q random inputs look random as long as the key stays secret.[10] The *Learning with Rounding* (LWR) assumption [6] is a cryptographic assumption stating that a certain efficiently computable function forms a PRF. To introduce the PRF, let us first define the *rounding function*. Given two integers Q, P such that $Q > P$, the rounding function $\lfloor \cdot \rceil_P \colon \mathbb{Z}_Q \to \mathbb{Z}_P$ is defined as $\lfloor x \rceil_P \coloneqq \lfloor x \cdot \frac{P}{Q} \rceil$, where $\lfloor . \rceil$ denotes the floor function. The same definition applies to vectors, assuming that the rounding function is evaluated component-wise. For integers m, n, P, Q, the PRF is defined as the family indexed by keys $\mathbf{k} \in \mathbb{Z}_Q^n$, with inputs $\mathbf{X} \in \mathbb{Z}_Q^{m \times n}$ and outputs $\mathbf{y} \in \mathbb{Z}_P^m$ such that $F_{\mathbf{k}}(\mathbf{X}) = \lfloor \mathbf{X}\mathbf{k} \rceil_P$. The learning with rounding assumption states that F is a 1–PRF.

A.5 Hash Functions

We call *hash function* a non-injective function H that maps arbitrary-length inputs into fixed-length outputs. The notion of *collapsing* hash function was introduced by Unruh [41] to extend the notion of collision-resistance to the quantum setting. Intuitively, collision resistance requires that it is hard to find two distinct inputs yielding the same output through a hash function. An hash function is collapsing when, in case a quantum-polynomial-time attacker can find a superposition of inputs yielding the same hash value, it cannot determine whether this superposition was measured. Many cryptographic security proofs idealize the hash function as a truly random function, referred to as a *random oracle* [8]. In the random oracle model (ROM), the hash function is replaced by an oracle that samples its outputs uniformly at random, and the adversary can evaluate it only by issuing explicit queries on chosen inputs. Boneh *et al.* [11] extend this idea to the quantum setting, introducing the *quantum random oracle model* (QROM). In the QROM, the adversary is allowed to query the oracle in superposition, meaning that it can submit quantum states and receive the corresponding superposition of hash evaluations. This captures the capabilities of quantum adversaries interacting with classically specified hash functions. The security of our scheme in Section 2.3 relies on this model, since it applies the Fiat-Shamir transform (see Section B) in the post-quantum setting, where quantum access to the random oracle must be taken into account.

[10] In fact, this is a relaxed notion of pseudorandomness usually referred to as "weak" in the literature. Since this is the only notion of pseudorandom function used in this work, we omit the term weak for simplicity.

B Signatures from Sigma Protocols and Fiat-Shamir with Aborts

Section 2.2 presents the construction of a post-quantum signature scheme derived from a Sigma protocol. To this end, we employ the Fiat–Shamir transform [26] in the variant with aborts introduced by Lyubashevsky [33].

In the abort-free setting, the construction proceeds as follows: the public key is a Sigma protocol statement x, the signing key is the corresponding witness w, and a signature consists of a transcript of the Sigma protocol. Non-interactivity is achieved by deriving the challenge as the hash of the public key, the first-round messages of each iteration of the Sigma protocol, and the message to be signed. If the underlying Sigma protocol may produce aborting transcripts, the signing algorithm repeats the protocol until an accepting transcript is obtained. The overall construction is summarized in Figure 3. This paradigm has been analyzed under a wide range of models and assumptions on the underlying primitives, which we use in Section 2.3 to prove the security of our scheme.

$\mathsf{S.Gen}(1^\lambda)$

$(x, w) \xleftarrow{\$} \mathcal{D}_{\mathcal{X}\times\mathcal{W}}(1^\lambda)$
$(vk, sk) \leftarrow (x, (x, w))$
return (vk, sk)

$\mathsf{S.Verify}(vk, msg, \sigma)$

$(\mathit{firstmes}, resp) \leftarrow \sigma$
$ch \leftarrow H(\mathit{firstmes} \| msg)$
return $\Sigma.\mathsf{Verify}(vk, \mathit{firstmes}, ch, resp)$

$\mathsf{Sign}_{sk}(msg)$

while $resp = \perp$ **do**
  $\mathit{firstmes} \leftarrow \Sigma.\mathsf{Prove}_1(sk)$
  $ch \xleftarrow{\$} H(\mathit{firstmes} \| msg)$
  $resp \leftarrow \Sigma.\mathsf{Prove}_2(ch)$
$\sigma \leftarrow (\mathit{firstmes}, resp)$
return σ

Fig. 3. The signature scheme constructed via Fiat-Shamir with Abort from any Sigma protocol $(\Sigma.\mathsf{Prove}_1, \Sigma.\mathsf{Prove}_2, \Sigma.\mathsf{Verify})$. $\mathcal{D}_{\mathcal{X}\times\mathcal{W}}$ is any distribution generating a pair of statement and witness satisfying the Sigma protocol relation $\mathcal{R}$.