On the Efficiency and Flexibility of Signature Verification

Cecilia Boschini[1], Dario Fiore[2], and Elena Pagnin[3]

[1] Università della Svizzera Italiana, Lugano, Switzerland
cecilia.boschini@usi.ch
[2] IMDEA Software Institute, Madrid, Spain
dario.fiore@imdea.org
[3] Lund University, Lund, Sweden
elena.pagnin@eit.lth.se

Abstract. Digital signatures are a well-established mean to securely certify data integrity and authenticate sources. One core component of digital signature schemes is signature verification. Traditionally, verification is monolithic and returns a decision (accept/reject) only at the very end of the process. In this work, we pose two questions that dismantle this monolithic view on signature verification: *(1) is it possible to extract meaningful information from a partial verification?* (flexibility); and *(2) is it possible to split the verification process into a computational heavy, one-time set-up, and a lightweight, reusable part, without undermining unforgeability?* (efficiency). We answer both questions in a positive way for specific classes of schemes that include post-quantum secure signatures from lattices and from multivariate polynomials.
We develop formal frameworks for signatures with efficient verification, flexible verification, and combinations of the two. Crucially, we regard these as features that may enhance existing constructions, rather than requiring a re-design. For each framework, we exhibit generic transformations to realize efficient (and/or) flexible verification for signature schemes that involve a matrix-vector multiplication among the checks. Our transformations apply to the NIST finalist Rainbow; MP12 (EUROCRYPT); GVW15 (STOC); and Lyub12 (EUROCRYPT) when implemented with non-cryptographic hash functions as suggested by Chen et al. (CRYPTO21), among other schemes.

Keywords: Digital signatures, amortized efficiency, flexible verification, post quantum signatures.

1 Introduction

Digital technologies gained a fundamental role in our society, affecting the security of crucial systems such as autonomous vehicles, healthcare, payments, e-voting, and access control. Digital signatures are among the cryptographic primitives employed to safeguard such systems from misuse [11,13,27,34]. Concretely, digital signatures allow one party, the signer, to use her secret key to authenticate a message in such a way that anyone holding the corresponding public key, the verifiers, can check its validity at any later point in time. Among many properties, digital signatures serve the purpose of securely establishing the source and integrity of the information. In many applications, the outcome of a signature verification determines what decisions to take, e.g., whether to accept a financial transaction (Bitcoin protocol), install software updates (Android OS), or deliver e-services (e-Health, electronic tax systems).

While the above examples are not time-critical, digital signatures may be employed in real-time cyber-physical systems as well, where the speed at which verification is performed plays a crucial role. This is the case, e.g., for automatic safety measures in connected vehicles or nuclear power plants. In real-time systems, verification speed (due to resource constraints) is not the only problem though. For a variety of reasons, a computation can get arbitrarily interrupted, leaving the verifier with an unsolved answer about the validity of a given signature. This has to do with the fact that standard verification procedures provide a binary outcome (0 or 1, reject or accept),

which is established only at the very end of the execution. One may wonder: *is it possible to extract meaningful information from a partial signature verification?* Le et al. [25] proposed to address unexpected interruptions using signatures with flexible verification. In a nutshell, such schemes admit a verification algorithm that increasingly builds confidence on the validity of the signature while it performs more steps. In this way, at the moment of an interrupt, the verifier is left with a value $\alpha \in [0,1] \cup \perp$ that probabilistically quantifies the validity of the signature. In particular, flexible signatures identify tradeoffs between the amount of computation performed and the integrity of a signature. A different way to approach this problem would be to limit the verification to few, quick steps, as to reduce the chance of interruptions in media res. One may wonder: *is it possible to speed up the verification process without impacting unforgeability?* This is the aim of efficient verification: to leverage special tradeoffs to quickly reach accurate conclusions about the validity of a signature.

1.1 Our contribution

In this work, we address two main challenges: (1) speeding up the verification of digital signatures; and (2) extracting meaningful information from a partial signature verification. We do so by designing *new* verification methods for *existing* signature schemes. Namely, our goal is to keep the signatures (i.e., key generation, sign and verify algorithms) as they are, and to devise techniques for alternative verification methods that are significantly more efficient and/or that retain usefulness even when interrupted *in media res*.

We introduce formal security models for the notions of efficient verification and flexible verification. For both settings, we provide information-theoretic transformations (that we call compilers) that apply to a wide range of existing signature schemes and turn their verification into efficient / flexible. We prove the resulting schemes secure in our security model. Interestingly, large part of the proof is devoted to a detailed analysis of the leakage due to verification queries (that now involves secret randomness). We consider this leakage analysis a result of independent interest that can be used to estimate leakage in similar information-theoretic approaches to provably secure algorithmic speed-ups or flexiblity. Moreover, we show our compilers can be combined to simultaneously achieve flexible and efficient verification for which we present a formal model, generic realizations, security arguments and implications. Finally, we remark that our models can easily be extended to include signatures with advanced properties including: ring, threshold, homomorphic, attribute-based and constrained.

Realizations. We focus our realizations on two families of digital signatures: lattice-based and multivariate-polynomials-based. Both are interesting examples thanks to their plausible post-quantum security. Concretely, our compilers apply to signature schemes with '**Mv**'-style verification, i.e., for which the core of the verification is a matrix-vector product. The multivariate-polynomials-based schemes we consider are Rainbow [14,15], one of the NIST candidates for standardization, and LUOV [5]. For lattice-based constructions, our compiler supports a wide family of schemes, such as GPV [20] (hash & sign), MP [29] (Boyen/ BonsaiTree), Lyubashevsky [26] (Fiat-Shamir with abort) when instantiated with a gadget-matrix-based hash function as recently shown in [12], and GVW [21] (homomorphic).

Efficient Verification. We build on the well-known offline/online paradigm to define an alternative verification that consists of two phases. An offline *probabilistic* preprocessing offVer that, given only

the public key and a security parameter, outputs a concise (short) verification key svk. An online algorithm onVer that uses such verification key to establish the validity of a signature on a given message. In our paradigm, (i) the preprocessing is one-time, namely its output svk can be reused to verify an unbounded number of signatures, (ii) the running time of onVer is required to be asymptotically smaller than that of the standard verification algorithm, and (iii) the preprocessed verification key svk must be kept secret by the verifier. While the first two properties (i)-(ii) are clearly the ones that provide the desired efficiency boost, one may wonder if property (iii) is really needed. To this end, we observe that devising algorithms $\mathsf{offVer}, \mathsf{onVer}$ with efficient verification (i)-(ii) and a *public* svk for an existing signature scheme would immediately lead to finding a new version of the signature scheme with faster verification. We therefore consider keeping svk secret and trade some usability for more efficiency. We formalize this secrecy requirement in a rigorous security model in which the standard unforgeability experiment is extended by asking the adversary to produce a forgery $(\mu, \boldsymbol{\sigma})$ that verifies under $\mathsf{onVer}(\mathsf{svk}, \cdot, \cdot)$ and by giving it oracle access to the online verification with the same preprocessed svk. Intuitively, this experiment shows the learning outcome of verification queries does not leak enough information about svk and does not help the adversary in producing a forgery. We stress that *in our model signatures are still publicly verifiable*; svk is only a secret that the verifier generates for itself, in order to speedup its computation.

In terms of realizations, we propose a compiler that yields efficient verification algorithms for a broad class of digital signatures in which the verification includes a matrix-vector multiplication. This can model the $\mathbf{A} \cdot \boldsymbol{\sigma} = \mathbf{u}$ check of several lattice-based signatures (LBS) where matrix $\mathbf{A}$ may be a public-key constant and $\mathbf{u}$ be message-dependent, as in [16,20,21], or $\mathbf{A}$ may be message-dependent and $\mathbf{u}$ be constant [8,9,29]. Or it can model the check of a system of multivariate quadratic polynomials $\{f_i(\mathbf{s}) = h_i\}_i$, where $\mathbf{s}$ depends on the signature, h_i is message-dependent and $f_i(\cdot)$ is a public-key-dependent polynomial [5,14,15]. For the LBS instantiations we consider, assuming a lattice of dimension $n = 128$, our method offers an online verification that is between $97\times$ to $99\times$ faster (see Section 2.2 for details).

Flexible Verification. We consider flexible verification as an *add-on* property to existing signature schemes. Concretely, we devise flexible verification via an algorithm $\mathsf{flexVer}$ and a confidence function α_{flex} with the following properties. First, $\mathsf{flexVer}$ is stateful, uses private-coins, and consists of at most $N + 1$ inner steps, so that at the end of step i one is either sure about rejection or expects the signature to be valid with confidence $\alpha_{\mathsf{flex}}(i)$. Second, $\alpha_{\mathsf{flex}} : \{0, \ldots, N\} \to [0, 1]$ is a non-decreasing function (that depends both on the signature scheme and on $\mathsf{flexVer}$) satisfying $\alpha_{\mathsf{flex}}(0) = 0$ and $\alpha_{\mathsf{flex}}(N) = (1 - \varepsilon(\lambda))$. In particular, reaching the last flexible verification step provides the same security guarantees as standard verification (except for a negligible chance of error). Third, we model secure flexible verification by letting the adversary decide at which step to interrupt each execution of $\mathsf{flexVer}$, even in the forgery check. Since in such a case a wrong signature may be marked as 'maybe valid' with non negligible probability (as the adversary may choose to verify it with not enough steps to reach overwhelming confidence), we require the adversary to find a forgery that is not rejected with probability non-negligibly *higher* than the expected confidence level at the chosen interruption step i, i.e., $\alpha_{\mathsf{flex}}(i)$.

In terms of realizations, we again focus on the aforementioned class of signatures with a matrix-vector multiplication and we propose flexible verification algorithms for them. In a nutshell, for signatures that work over $\mathbb{Z}_q$ or $\mathbb{F}_q$, our flexible verification interrupted at step i can achieve confidence $(1 - q^{-i})$. In the case of signature schemes where $q = 2^{poly(\lambda)}$, this confidence level is always overwhelming, even with minimal computation, i.e., $i = 1$. Interestingly, in this case the verification

is both flexible and efficient: there exists a minimum number N so that N executions of $\mathsf{flexVer}$ are faster than N executions of Ver. For signature schemes where $q = \mathsf{poly}(\lambda)$, we show a variant of our compiler that simultaneously realizes efficient and flexible verification. This though holds in a weaker security model, in which the confidence function α_{flex} depends, and degrades, with the number of verifications.

1.2 An Overview of Our Technique

Our Compiler for Efficient Verification. We consider the class of digital signature schemes for which the bulk of computation in the verification procedure consists of a matrix-vector product (what we call '$\mathbf{Mv}$'-style check). Several lattice-based [6,16,20,21,26,29] and multivariate-polynomials-based signatures [5,15] fall in this category. For simplicity, here we give a technical overview for LBS constructions only, for the case $\mathbf{Mv} = \mathbf{u}$, where $\mathbf{M} \in \mathbb{Z}_q^{n \times m}$ is fixed (e.g., the signer's public key), $\mathbf{v}$ is a vector (e.g., a signature) and $\mathbf{u}$ depends on the message [16,20,21]. The key observation to efficient verification is that if a signature $\mathbf{v}$ verifies for a given $\mathbf{M}$ (and $\mathbf{u}$), then it also verifies for any linear combination of the rows of $\mathbf{M}$ (against the same combination to the entries of $\mathbf{u}$). Let $\mathbf{c} \xleftarrow{\$} \mathbb{Z}_q^n$ be a random vector; denote $\mathbf{z} \leftarrow \mathbf{c} \cdot \mathbf{M} \in \mathbb{Z}_q^m$, and $w \leftarrow \mathbf{c} \cdot \mathbf{u} = \sum_{i=1}^{n} \mathbf{c}[i]\mathbf{u}[i] \in \mathbb{Z}_q$. Then

$$\mathbf{Mv} = \mathbf{u} \mod q \Longrightarrow \mathbf{c} \cdot (\mathbf{Mv}) = \mathbf{c} \cdot \mathbf{u} \mod q \Leftrightarrow \mathbf{z} \cdot \mathbf{v} = w \mod q \tag{1}$$

Our compiler is based on the fact that the first implication in (1) holds left-to-rightwards (always) *and also* right-to-leftwards with *good enough* probability over the random choice of $\mathbf{c}$. Moreover, one can further increase this probability by increasing the number of vectors $\mathbf{c}$ and $\mathbf{z}$ to use in the signature verification. To achieve efficient 'online' verification, we can precompute a set of $k \geq 1$ 'random combinations of rows of $\mathbf{M}$' during an 'offline verification' phase; and re-use the output pairs $(\mathbf{c}_j, \mathbf{z}_j)$, $j \in \{1, \ldots, k\}$ to efficiently verify an unbounded number of signatures (or in other words, the verifier stores matrices $\mathbf{C} \in \mathbb{Z}_q^{k \times n}, \mathbf{Z} \in \mathbb{Z}_q^{k \times m}$). Concretely, the efficiency gain in the online verification comes by replacing the check $\mathbf{Mv}$ (that requires nm multiplications modulus q) with k checks of the form $\mathbf{z}_j \cdot \mathbf{v} = w_j$ (that require at most km multiplications).

To argue that this verification method is secure we rely on the fact that the verifier chooses and keeps secret the matrix $\mathbf{C}$, and that an adversary, without knowledge of $\mathbf{C}$, cannot generate signatures for which the right-to-leftward implication does not hold, namely signatures that would be rejected by standard verification but accepted by the efficient one using $\mathbf{C}$. However, the leftwards implication in (1) holds, information-theoretically, with probability $1-q^k$, which is overwhelming for very small values of k, and even with $k = 1$ for schemes where $q = 2^{\mathsf{poly}(\lambda)}$, e.g., [16,21]. Although this is the basic intuition behind the security of our compiler, the precise security analysis of our compiler turns much more complex due to the fact that, before producing a forgery, the adversary can interact with an oracle that verifies signatures using the secret $\mathbf{C}$ and might thus leak information about this matrix. So, a main technical contribution of our paper is an analysis of this event. In particular, we show how to reduce such adversaries to a core lemma (Lemma 2) that bounds the probability of finding a nonzero vector $\mathbf{w}^*$ in the right kernel of $\mathbf{C}$ (i.e., $\mathbf{C}\mathbf{w}^* = 0$) under the knowledge of i vectors $\mathbf{w}_j$ that are not in the kernel, i.e., such that $\mathbf{C} \cdot \mathbf{w}_i \neq 0$. We believe this result can be of independent interest.

Our Compiler for Flexible Verification. The idea of the compiler presented above can be used to achieve flexible verification in the following way. For a set of freshly sampled vectors $\{\mathbf{c}_j\}_{j=1}^{J}$, a verification that uses a subset of $k < J$ such vectors is correct with probability $(1 - q^{-k})$; hence this value can be the confidence, i.e., $\alpha_{\mathsf{flex}}(k) = (1 - q^{-k})$.

1.3 Related Work

The problem of trading security for less computation during a signature verification has been considered by Le et al. [25], who introduced the notion of flexible signatures and a construction based on the Lamport-Diffie one-time signature [24] with Merkle trees [28]. A similar idea in the context of message authentication codes (MACs) has been considered by Fischlin [17] who put forth progressive verification for MACs and presented two concrete constructions. One main difference between our model and those of [17,25] is that we aim to capture flexible verification as an independent feature that can enhance existing schemes, rather than a standalone primitive. This is in a way more challenging as it leaves less design freedom when crafting these algorithms. Therefore we decided to define flexible verification as a *stateful* algorithm in contrast to stateless [17,25]: although this makes our model slightly more involved, it is comparably more general and can capture more (existing) schemes.

Our model for efficient verification is close the offline-online paradigm used in homomorphic authentication [2,10] and verifiable computation [19]; where a preprocessing is done with respect to a function f, and its result can be used to verify computation results involving the same f. An early instantiation of this technique for speeding up the verification of Rabin-Williams signatures appears in [4].

Sipasseuth et al. [32] investigate how to speed up lattice-based signature verification while reducing the memory (storage) requirements. The overall idea in [32] is similar to ours (and inspired to Freivalds' Algorithm): to replace the inefficient matrix multiplication in the verification with a probabilistic check via an inner product computation. However, [32] focuses on the DRS signature [31], and investigates the trade-off between pre-computation time for verification and memory storage for this scheme only. Moreover, the work lacks a formal, abstract analysis of the security impact of such a shift in the verification procedure. In contrast, we devise a general framework to model 'more efficient' and 'partial' signature verification. Albeit we developed our approach independently of [32], our techniques can be seen as a generalization of what presented in [32]. Taleb and Vergnaud [33] study how to speed up the verification of the RSA, ECDSA and GVP [20] signatures. While the first two are complementary to our work, the latter construction is a different take on achieving efficient verification for this class of LBS (they do not have a comparable approach for the flexible case). Their approach exploits a particular type of error correcting codes that do not have words with small Hamming weight. Given a generator matrix $\mathbf{G}$ for the code, their verification algorithm checks that the codeword $\mathbf{G}(\mathbf{A}\boldsymbol{\sigma} - \mathbf{u}) \mod q$ has null components. The sparsity property of the code ensures that if $\mathbf{A}\boldsymbol{\sigma} - \mathbf{u} \neq 0 \mod q$ (i.e., $\boldsymbol{\sigma}$ is not a valid signature), the corresponding codeword has enough nonzero components that a random choice of them would contain a nonzero one with overwhelming probability. A drawback with this technique is that the resulting construction is not a compiler, as ours, but requires to modify both the key generation and the verification algorithm, and requires the public key to be longer, thus slowing down the full verification. Moreover, their progressive verification still requires a number of vector multiplications linear in the security parameter, while ours only requires a logarithmic number of linear products.

Remark on terms. We use the term 'flexible signature' to be consistent with previous work [25], however, this wording may be misleading in that there is no flexibility in the signature scheme, but rather the verification procedure is carried out in a progressive way (á la Fischlin [17]): the more computation one performs the more confident one becomes of accepting or rejecting a signature (gradually approaching 100% certainty). We believe that an algorithm should be able to read (and understand) its input, and should not have an accessory input that cannot use (the $[[k]]$ of [25]).

Differently from ours, their technique does not yield a compiler and requires to modify both key generation and the public key of [20], which becomes longer: this avoids the need of a secret-key preprocessing, as in our case, but results in a less efficient verification than our approach.

2 Efficient Verification for Digital Signatures

In this section, we introduce the concept of *efficient verification* for digital signatures and a suitable formal security model that allows to estimate the impact efficient verification has on the unforgeability of the scheme. Then, we describe a generic compiler to obtain efficient verification for a wide class of signatures, prove its security, show realizations from lattices and multivariate polynomials and finally discuss its concrete efficiency against full-fledged verification. In addition we discuss concrete instantiations and their corresponding concrete efficiency gains. This section ends with a generalization of our results to signatures with properties.

Notation A function $\varepsilon : \mathbb{Z}_{\geq 0} \to [0,1]$ is negligible if $\varepsilon(\lambda) < 1/\mathsf{poly}(\lambda)$ for every univariate polynomial $\mathsf{poly} \in \mathbb{R}[\mathsf{X}]$ and a large enough integer $\lambda \in \mathbb{Z}_{\geq 0}$. Throughout the paper, $\lambda \in \mathbb{Z}_{\geq 0}$ denotes the security parameter of a cryptographic scheme. The abbreviation PPT refers to algorithms that are probabilistic and run in polynomial time.

Digital Signatures. A signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ with message space $\mathcal{M}$ includes a key generation algorithm $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, a sign algorithm $\boldsymbol{\sigma} \leftarrow \mathsf{Sign}(\mathsf{sk}, \mu)$ that outputs a signature $\boldsymbol{\sigma}$ on a message μ, and a verification algorithm $\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma})$ that outputs 1 (accept) or 0 (reject).

Efficient Verification. In a nutshell, the core idea of efficient signature verification is to split the verification process into two steps. The first step is a one-time and signature-independent setup called 'offline verification'. Its purpose is to produce randomness to derive a (short, secret) verification key svk from the signer's public key pk. Note that the offline verification does not change the signature, which remains publicly verifiable; instead it 'randomizes' pk to obtain a concise verification key svk that essentially enables one to verify signatures with (almost) the same precision as the standard verification, but in a more efficient way. The second verification step consists of an 'online verification' procedure. It takes as input svk and can verify an unbounded number of message-signature pairs performing significantly less computation than the standard verification algorithm.

Definition 1 (Efficient Verification). *A signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ admits efficient verification if there exist two PPT algorithms $(\mathsf{offVer}, \mathsf{onVer})$ with the following syntax:*

$\mathsf{offVer}(\mathsf{pk}, k)$**:** *on input a verification key pk, a positive integer $k \in \{1, \ldots, \lambda\}$ (referred to as confidence level), the offline verification algorithm returns a secret verification key svk.*

$\mathsf{onVer}(\mathsf{svk}, \mu, \boldsymbol{\sigma})$**:** *on input a secret verification key svk, a message μ, and a signature $\boldsymbol{\sigma}$, the efficient online verification algorithm outputs 0 (reject) or 1 (accept).*

For convenience we may refer to the signature scheme augmented with the efficient verification algorithms as $\Sigma^E = (\Sigma, \mathsf{offVer}, \mathsf{onVer})$.

The standard properties of an efficient verification scheme are described below.

Correctness. A scheme $\Sigma^E = (\Sigma, \mathsf{offVer}, \mathsf{onVer})$ realizes efficient verification correctly if, for a given security parameter λ, for any honestly generated key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\lambda)$, for any message $\mu \in \mathcal{M}$, for any signature $\boldsymbol{\sigma}$ such that $\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma}) = 1$, and for any confidence level $k \in \{1, \ldots, \lambda\}$; correctness requires that

$$\Pr[\mathsf{onVer}(\mathsf{svk}, \mu, \boldsymbol{\sigma}) = 1 \mid \mathsf{svk} \leftarrow \mathsf{offVer}(\mathsf{pk}, k)] = 1.$$

This guarantees that a valid signature $\boldsymbol{\sigma}$ is always accepted by online verification. We remark that any signature rejected by onVer, would be rejected by Ver as well.

Concrete Amortized Efficiency. Intuitively, amortized efficiency relies on the fact that the global computational cost of running offVer once, and onVer a given number times, is considerably smaller than the cost of running the standard signature verification Ver the same number of times. More formally, let $\mathsf{cost}(\cdot)$ be a function that, given in input an algorithm returns its computational cost (in some desired computational model). A scheme $\Sigma^E = (\Sigma, \mathsf{offVer}, \mathsf{onVer})$ realizes concrete amortized efficient verification if verifying r signatures with the efficient verification approach requires less computational effort than running r standard verifications.

Definition 2 (Concrete Amortized Efficiency). *A scheme $\Sigma^E = (\Sigma, \mathsf{offVer}, \mathsf{onVer})$ realizes $(\mathsf{r}_0, \mathsf{e}_0)$-concrete amortized efficient verification if given a security parameter λ and a confidence level k; for any key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\lambda)$, for any pair $(\mu, \boldsymbol{\sigma})$ with $\mu \in \mathcal{M}$ and $\boldsymbol{\sigma}$ such that $\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma}) = 1$; there exist a minimum non-negative integer r_0, and a small, real constant $0 < \mathsf{e}_0 < 1$ such that:*

$$\forall\ \mathsf{r} \geq \mathsf{r}_0\ , \quad \frac{\mathsf{cost}\big(\mathsf{offVer}(\mathsf{pk}, k)\big) + \mathsf{r} \cdot \mathsf{cost}\big(\mathsf{onVer}(\mathsf{svk}, \mu, \boldsymbol{\sigma})\big)}{\mathsf{r} \cdot \mathsf{cost}\big(\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma})\big)} < \mathsf{e}_0 \tag{2}$$

Security. Our definition of efficient verification lets the verifier set the confidence level $k \in \{1, .., \lambda\}$ at which she wishes to carry out the signature verification. Notably k also determines the amount of computation to be performed by offVer and onVer and thus, the efficiency gain. Intuitively, we say that a scheme Σ^E realizes efficient verification in a secure way if the probability onVer accepts a signature that would be rejected by Ver is negligible. We cannot expect a more efficient verification to detect more forgeries than the full verification. The security game gives the adversary access to the signing oracle $O\mathsf{Sign}$ and to the efficient verification oracle $O\mathsf{onVer}$, since this employes the secret verification key ouptput by offVer. As one would expect, the adversary can query the oracles in an adaptive and parallel way, a number times which is polynomial in the security parameter λ. This is formalized in the following definition.

Definition 3 (Security of Efficient Verification). *Let Σ be an existentially unforgeable signature scheme that admits an efficient verification $\Sigma^E = (\Sigma, \mathsf{offVer}, \mathsf{onVer})$. For a given security parameter λ and for any confidence level $k \in \{1, \ldots, \lambda\}$, the pair of algorithms $(\mathsf{offVer}, \mathsf{onVer})$ realizes a secure efficient verification for Σ if it is existentially unforgeable under adaptive chosen message and verification attack. In other words, if for all PPT adversaries $\mathcal{A}$ the success probability in the cmvEUF experiment in Figure 1 is negligible, i.e.:*

$$Adv^{\mathsf{cmvEUF}}_{\mathcal{A},\Sigma}(\lambda, k) = \Pr\Big[\mathsf{Exp}^{\mathsf{cmvEUF}}_{\mathcal{A},\Sigma}(\lambda, k) = 1\Big] \leq \varepsilon = \varepsilon(\lambda, k).$$

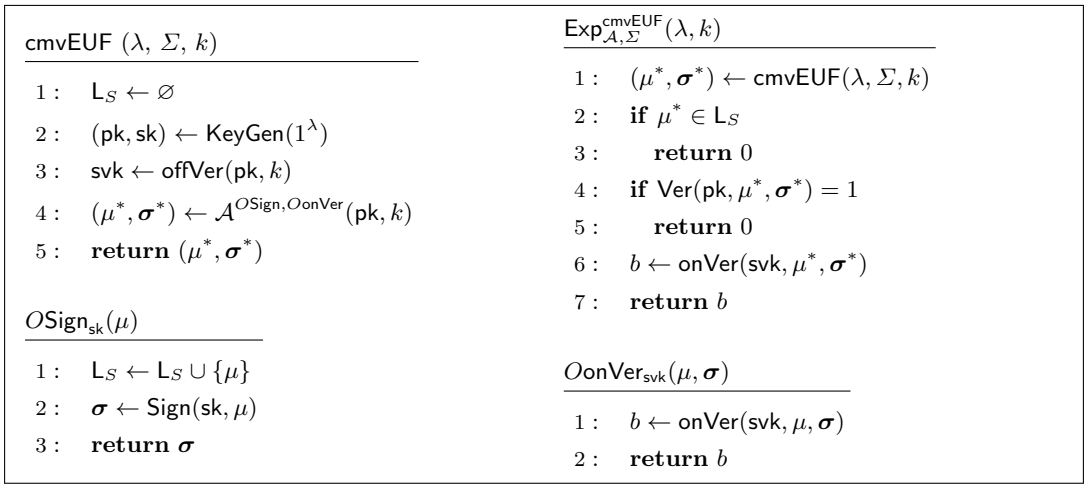


Fig. 1: Security model for efficient verification of signatures: existential unforgeability under adaptive chosen message and verification attack (security game, experiment and oracles).

Line 5 of the cmvEUF experiment excludes forgeries against the original signature scheme. This is justified by the correctness of efficient verification and by the fact that Σ is existentially unforgeable. In other words, the above security experiment checks whether the adversary outputs invalid signatures that are however accepted by the efficient verification mechanism.

Notably, both the security game and the advantage depend on the confidence level k and assume all algorithms are entirely executed. If we relax this last constraint and let $\mathcal{A}$ *prematurely interrupt* the execution of onVer, the advantage may become *non-negligible* for some values of k. A meaningful definition of security in such scenarios requires a new model as we show in Section 3.

2.1 A Compiler for Efficient Mv-style Verifications

So far we presented a framework for efficient verification of digital signatures. In what follows we show how to apply the efficient verification paradigm to a wide class of existing schemes. To this end, we identify a class of signature schemes Σ, that we call *with* $\mathbf{Mv}$*-style verification*, for which Ver can be seen as the combination of two types of verification checks: a matrix-vector multiplication (referred to as $\mathbf{Mv} = 0$, for appropriate matrix $\mathbf{M}$ and vector $\mathbf{v}$) and other generic checks (collected in the Check subroutine), see Figure 2 for details and an explanatory example. Next, we present a generic way to realize efficient verification whenever the computational complexity of Ver is dominated by the matrix-vector multiplication, i.e., $\mathsf{cost}(\mathsf{Check}) << \mathsf{cost}(\mathbf{Mv}) \sim mn$ field multiplications (for $\mathbf{M} \in \mathbb{Z}_q^{n \times m}$). We refer to this transformation as *our compiler* to transform any Σ with $\mathbf{Mv}$-style verification into a scheme Σ^E that realizes efficient verification.

Among the schemes with $\mathbf{Mv}$-style verification we highlight some of the seminal lattice-based signatures [8,9,20,29], signatures obtained using the Fiat-Shamir with abort technique [26], homomorphic signatures [6,16,21], and multivariate signatures [5,15].

We start by a quick recap of the notation we use when describing our compilers.

Notation. We denote the set of real values by $\mathbb{R}$, integers by $\mathbb{Z}$, natural numbers by $\mathbb{N}$, and finite fields of integers by $\mathbb{Z}_q$, where q is a (power of a) prime number. We denote vectors by bold, lower-case letters, and matrices by bold, upper-case letters. We use $\mathbf{v}[i]$ to identify the i-th entry of a vector $\mathbf{v}$, and $\mathbf{A}[i, j]$ to identify the entry in the i-th row and j-th column of a matrix $\mathbf{A}$. The

$\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma})$

// INITIALIZE ACCEPTANCE BITS
1 : $b_1 \leftarrow 0, b_2 \leftarrow 0$
// SPLIT pk INTO MARTIX - AUX. DATA
2 : **parse** $\mathsf{pk} = (PK, PK.\mathsf{aux})$
// ADDITIONAL VERIFICATION CHECKS
3 : $b_1 \leftarrow \mathsf{Check}(PK.\mathsf{aux}, \mu, \boldsymbol{\sigma})$
// FORMATTING Mv-STYLE CHECK
4 : $(\mathbf{M}, \mathbf{v}) \leftarrow \mathsf{GetMv}(\mathsf{pk}, \mu, \boldsymbol{\sigma})$
// MATRIX-VECTOR MULT. CHECK
5 : **if** $(\mathbf{M} \cdot \mathbf{v} = 0)$
6 : $\quad b_2 \leftarrow 1$
7 : **return** $(b_1 \ \wedge \ b_2)$

Example: $\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma})$ for GPV08 [20]

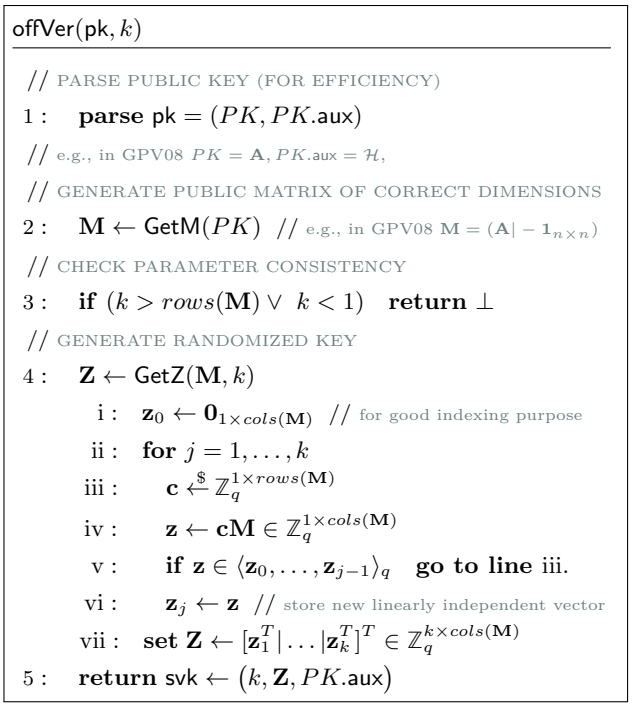
1 : $b_1 \leftarrow 0, b_2 \leftarrow 0$
2 : **parse** $\mathsf{pk} = (PK, PK.\mathsf{aux})$
set $PK \leftarrow \mathbf{A}$
set $PK.\mathsf{aux} \leftarrow (\mathcal{H}, \beta)$
3 : $\mathsf{Check}(PK.\mathsf{aux}, \mu, \boldsymbol{\sigma})$:
if $\|\boldsymbol{\sigma}\| < \beta$ **set** $b_1 \leftarrow 1$
4 : $\mathsf{GetMv}(\mathsf{pk}, \mu, \boldsymbol{\sigma})$:
set $\mathbf{M} \leftarrow [\mathbf{A}| - \mathbf{I}_{rows(\mathbf{A})}]$
set $\mathbf{u} \leftarrow \mathcal{H}(\mu) \in \mathbb{Z}_q^{rows(\mathbf{A}) \times 1}$
set $\mathbf{v} \leftarrow [\boldsymbol{\sigma}^T | \mathbf{u}^T]^T$
5 : **if** $(\mathbf{M} \cdot \mathbf{v} = \mathbf{0}_{rows(\mathbf{A}) \times 1} \mod q)$
6 : $\quad$ **set** $b_2 \leftarrow 1$
7 : **return** $(b_1 \ \wedge \ b_2)$

Fig. 2: General structure of a signature with **Mv**-style verification (on the left); an instructive example: the GPV08 [20] signature verification (on the right).

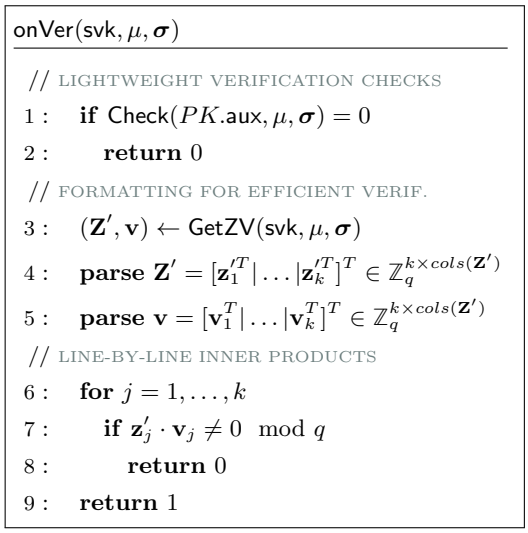
norm of a vector is denoted as $\|\mathbf{v}\|$ and unless otherwise specified, it is assumed to be the infinity norm, i.e., $\|\mathbf{v}\| = \max_i\{\mathbf{v}[i]\}$. $\mathbf{A}^T$ denotes the transposed of a matrix. We use $rows(\mathbf{A})$, $cols(\mathbf{A})$, and $rk(\mathbf{A})$ to respectively refer to the number of rows, the number of columns, and the rank of a matrix $\mathbf{A}$; $\mathbf{1}_{1\times n}$ (resp. $\mathbf{0}_{1\times n}$) denotes the row vector of length n that has all entries equal to 1 (resp. 0); while $\mathbf{I}_n$ denotes the n by n identity matrix of dimension n. We omit the explicit dimensions when they are clear from the context. We denote the span (linear space) generated by a set of vectors $\mathbf{z}_1, \ldots, \mathbf{z}_i$ in the discrete vector space $\mathbb{Z}_q^m$ as $\langle \mathbf{z}_1, \ldots, \mathbf{z}_i \rangle_q = \{\mathbf{z} \in \mathbb{Z}_q^m \ : \ \mathbf{z} = \sum_{j=1}^{i} a_j \mathbf{z}_j \mod q, \exists a_1, \ldots, a_i \in \mathbb{Z}_q\}$. We denote by $L_1|L_2$ the result of appending a list of elements L_2 to L_1. Given two values $a < b$, we denote a continuous interval as $[a, b] \subseteq \mathbb{R}$, and a discrete interval as $\{a, \ldots, b\} \subseteq \mathbb{Z}$.

Our Compiler. Our compiler for efficient verification is detailed in Figure 3 with a sketch of instantiation for the lattice-based scheme GPV08 [20] as a running example. Further details and instantiations from other schemes can be found in Section 2.2.

Our transformation takes as input Σ, a signature scheme with **Mv**-style verification; and it returns $\Sigma^E = (\Sigma, \mathsf{offVer}, \mathsf{onVer})$ that securely instantiates efficient verification for Σ. The heart of our compiler leverages the fact that for any pair of vectors $\boldsymbol{\sigma}$ and $\mathbf{u}$, and for any matrix $\mathbf{A}$ (of opportune dimensions) if $\mathbf{A} \cdot \boldsymbol{\sigma} = \mathbf{u}$ then for any random vector $\mathbf{c}$ (of opportune dimension) it holds that $\mathbf{c} \cdot (\mathbf{A} \cdot \boldsymbol{\sigma}) = \mathbf{c} \cdot \mathbf{u}$. Collecting variables on the left hand yields $\mathbf{c} \cdot [\mathbf{A}| - \mathbf{1}] \cdot (\boldsymbol{\sigma}|\mathbf{u}) = 0$. Thus one can precompute the vector $\mathbf{z} \leftarrow \mathbf{c} \cdot [\mathbf{A}| - \mathbf{1}]$ and run the efficient online verification check $\mathbf{z} \cdot \mathbf{v} \stackrel{?}{=} 0$, where $\mathbf{v} \leftarrow (\boldsymbol{\sigma}, \mathbf{u})$. In a nutshell the idea is to replace the matrix-vector multiplication with a vector-vector multiplication in a sound way. Correctness and efficiency are immediate. Soundness essenitally comes from the fact that if $\mathbf{z} \cdot \mathbf{v} = 0$, then with all but negligible probability also the original system of linear equations $\mathbf{A} \cdot \boldsymbol{\sigma} = \mathbf{u}$ is satisfied, as proven in Theorem 1. Despite this result being intuitive, analysing it while considering the leakage of svk that occurs through verification queries is not trivial and is one main technical contribution of this result.



(a) The offline verification algorithm.



(b) The online verification algorithm.

Fig. 3: Our compiler for efficient verification of signatures with **Mv**-style verification. The four scheme-dependent subroutines are: **parse** pk and GetZ (in offVer); Check and GetZV (in onVer). The complexity of the onVer is linear in k, the chosen confidence level.

Security Analysis of Our Compiler. In what follows, we state and prove the security of our compiler for efficient verification. We discuss the tradeoffs between the module size $q(\lambda)$ and the confidence level k in the next section.

Theorem 1. *Let Σ be an existentially unforgeable signature scheme with* **Mv***-style verification (as in Figure 2). The scheme $\Sigma^E = (\Sigma, \mathsf{offVer}, \mathsf{onVer})$ obtained via our compiler depicted in Figure 3 is existentially unforgeable under adaptive chosen message and verification attacks. Concretely, the advantage is $Adv^{CMVA}_{\mathcal{A},\Sigma}(\lambda, k) \leq \frac{q_V+1}{q^k - q_V}$ where $k \in \{1, \ldots, rk(\mathbf{M})\}$ denotes the chosen confidence level, $q_V = \mathsf{poly}(\lambda) << q^k$ is a bound on the total number of verification queries and q is the modulus.*

Remark. The above theorem considers only existential unforgeability. This is a bare simplification as the statement and the proof actually adapt with ease to other security models such as strong and selective unforgeability. We defer a detailed discussion the security properties carried on by our compiler to Section 2.3.

Proof. In the cmvEUF security experiment (Figure 1), the winning conditions require $\mathcal{A}$ to produce a message-signature pair $(\mu^*, \boldsymbol{\sigma}^*)$ such that μ^* has not been queried to the signing oracle during the game (existential unforgeability); $\boldsymbol{\sigma}^*$ is invalid under the standard verification, i.e., $\mathsf{Ver}(\mathsf{pk}, \mu^*, \boldsymbol{\sigma}^*) = 0$; and the pair is accepted by the online verification, i.e., $\mathsf{onVer}(\mathsf{svk}, \mu^*, \boldsymbol{\sigma}^*) = 1$. The goal of the proof is to bound the probability this event occurs.

Let Win be the event $\{\mathsf{Exp}^{\mathsf{cmvEUF}}_{\mathcal{A},\Sigma}(\lambda,k)=1\}$. Let $i=1$ to q_V be the index of the queries $(\mu_i,\boldsymbol{\sigma}_i)$ submitted by $\mathcal{A}$ to the $O\mathsf{onVer}$ oracle. For notation sake, define the family of events bad_i as:

$$\mathsf{bad}_i := \{\mathsf{Ver}(\mathsf{pk},\mu_i,\boldsymbol{\sigma}_i)=0 \wedge \mathsf{onVer}(\mathsf{svk},\mu_i,\boldsymbol{\sigma}_i)=1\},\ \text{for } i=1,\ldots,q_V+1$$

where bad_{q_V+1} corresponds to $\mathcal{A}$ returning a valid forgery $(\mu^*,\boldsymbol{\sigma}^*) =: (\mu_{q_V+1},\boldsymbol{\sigma}_{q_V+1})$ at the end of the experiment. We can rewrite the winning condition of the security experiment as $\mathsf{Win} = \{\mathsf{bad}_{q_V+1} \ \wedge\ \mu^* \notin \mathsf{L}_S\}$ (recall that L_S is the list of messages queried to the signing oracle in the game execution). Consider the event Bad defined as "there exists at least one query index i in the game execution for which bad_i occurs", formally

$$\mathsf{Bad} := \{\exists\ i \in \{1,\ldots,q_V\} : \mathsf{Ver}(\mathsf{pk},\mu_i,\boldsymbol{\sigma}_i)=0\ \wedge\ \mathsf{onVer}(\mathsf{svk},\mu_i,\boldsymbol{\sigma}_i)=1\}\ .$$

Clearly

$$\begin{aligned} Adv^{CMVA}_{\mathcal{A},\Sigma}(\lambda,k) &= \Pr[\mathsf{Win}\wedge\mathsf{Bad}] + \Pr[\mathsf{Win}\wedge\neg\mathsf{Bad}] \\ &\leq \Pr[\mathsf{Bad}] + \Pr[\mathsf{Win}\mid\neg\mathsf{Bad}] \end{aligned} \tag{3}$$

where the inequality comes from applying the definition of conditional probability and upperbounding $\Pr[\mathsf{Win}\mid\mathsf{Bad}]$ and $\Pr[\neg\mathsf{Bad}]$ by 1.

We notice that $\Pr[\mathsf{Win}\mid\neg\mathsf{Bad}]$ is essentially the probability that the event bad_i occurs only for $i=q_V+1$ and never before, i.e.,

$$\Pr[\mathsf{Win}\mid\neg\mathsf{Bad}] \leq \Pr\left[\mathsf{bad}_{q_V+1}\mid \bigwedge_{i=1}^{q_V}\neg\mathsf{bad}_i\right] \tag{4}$$

(In case Σ is strongly unforgeable, the inequality in (4) turns into an equality). In order to bound $\Pr[\mathsf{Bad}]$, let us define, for $i=1$ to q_V, Bad^*_i as the event "bad_i occurs *for the first time* at query i", namely $\mathsf{Bad}^*_i = \mathsf{bad}_i \wedge \left(\bigwedge_{j=1}^{i-1}\neg\mathsf{bad}_j\right)$. Then we have

$$\Pr[\mathsf{Bad}] = \Pr\left[\bigvee_{i=1}^{q_V}\mathsf{Bad}^*_i\right] = \sum_{i=1}^{q_V}\Pr[\mathsf{Bad}^*_i] \leq \sum_{i=1}^{q_V}\Pr\left[\mathsf{bad}_i\mid\bigwedge_{j=1}^{i-1}\neg\mathsf{bad}_j\right] \tag{5}$$

where the second equality holds because the events Bad^*_i are all disjoint, and the inequality follows from applying the definition of conditional probability and upperbounding $\Pr\left[\bigwedge_{j=1}^{i-1}\neg\mathsf{bad}_j\right]$ by 1, for all i. Therefore, putting together equations (4), (5) and (3) we have

$$Adv^{CMVA}_{\mathcal{A},\Sigma}(\lambda,k) \leq \sum_{i=1}^{q_V+1}\Pr\left[\mathsf{bad}_i\mid\bigwedge_{j=1}^{i-1}\neg\mathsf{bad}_j\right] \tag{6}$$

Thanks to Lemma 1 we can bound the above advantage by

$$Adv^{CMVA}_{\mathcal{A},\Sigma}(\lambda,k) \leq \sum_{i=1}^{q_V+1}\frac{1}{q^k-(i-1)} \leq \frac{q_V+1}{q^k-q_V}$$

without additional assumptions on q_V, and, assuming $\frac{1}{2} < q_V < \frac{2q^k+1}{2}$ (which is almost always true) $Adv^{CMVA}_{\mathcal{A},\Sigma}(\lambda,k) \leq \ln\left(\frac{q^k}{q^k-q_V}\right) + \frac{1}{q^k-q_V} \leq \frac{q_V+1}{q^k-q_V}$.

Lemma 1. *For all tuples $q_V, q, k \in \mathbb{N}$ satisfying $q_V < q^k$, it holds that*

$$\sum_{i=1}^{q_V+1} \Pr\left[\mathsf{bad}_i \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\right] \leq \frac{q_V+1}{q^k - q_V}.$$

Moreover, whenever the number of verification queries q_V is such that $\frac{1}{2} < q_V < \frac{2q^k+1}{2}$ the following tighter bound holds: $\sum_{i=1}^{q_V+1} \Pr\left[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\right] \leq \ln\left(\frac{q^k}{q^k-q_V}\right) + \frac{1}{q^k-q_V}$.

Proof of Lemma 1 We use a result proven in Lemma 2, namely that $\Pr\left[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\right] \leq \frac{1}{q^k-(i-1)}$ for all $i = 1, \ldots q_V + 1$. This implies that

$$\sum_{i=1}^{q_V+1} \Pr\left[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\right] \leq \sum_{i=1}^{q_V+1} \frac{1}{q^k-(i-1)} \ .$$

Observing that $\frac{1}{q^k-(i-1)} \leq \frac{1}{q^k-q_V}$ for all i yields $\sum_{i=1}^{q_V+1} \frac{1}{q^k-(i-1)} \leq \frac{q_V+1}{q^k-q_V}$, which proves the first, general bound.

Towards proving the tighter bound, we begin by observing that

$$\sum_{i=1}^{q_V+1} \frac{1}{q^k-(i-1)} = \sum_{i=q^k-q_V}^{q^k} \frac{1}{i} = \sum_{i=1}^{q^k} \frac{1}{i} - \sum_{i=1}^{q^k-q_V} \frac{1}{i} + \frac{1}{q^k-q_V} = H_{q^k} - H_{q^k-q_V} + \frac{1}{q^k-q_V} \ , \qquad (7)$$

where $H_n = \sum_{k=1}^{n} \frac{1}{k}$ denotes the n-th harmonic number. It is known that harmonic numbers satisfy the following folklore lower and upper bounds[4]:

$$\frac{1}{2n+1} + \ln(n) + \gamma \leq H_n \leq \frac{1}{2n} + \ln(n) + \gamma \qquad (8)$$

where γ is the Euler-Mascheroni constant, $0.5 < \gamma < 0.58$. Using the upper bound in (8) to over estimate H_{q^k} and the lower bound to under estimate $H_{q^k-q_V}$ in Equation (7) we get

$$\begin{aligned}
\sum_{i=1}^{q_V+1} \frac{1}{q^k-(i-1)} &\leq \left(\frac{1}{2q^k} + \ln(q^k) + \gamma\right) - \left(\frac{1}{2(q^k-q_V)+1} + \ln(q^k-q_V) + \gamma\right) + \frac{1}{q^k-q_V} \\
&= \ln\left(\frac{q^k}{q^k-q_V}\right) + \frac{1}{q^k-q_V} - \frac{2q_V-1}{2q^k(2q^k-2q_V+1)} \\
&\leq \ln\left(\frac{q^k}{q^k-q_V}\right) + \frac{1}{q^k-q_V} \ ,
\end{aligned}$$

where the second inequality follows from the the fact that for $q \neq 0$ and $q_V \in \mathbb{N}$ if $\frac{1}{2} < q_V < \frac{2q^k+1}{2}$ then $\frac{2q_V-1}{2q^k(2q^k-2q_V+1)} > 0$. It is easy to see that the second bound is tighter than the generic one; this claim follows directly from the fact that for all $x > 0$, $\ln(x) \leq x-1$, thus $\ln\left(\frac{q^k}{q^k-q_V}\right) + \frac{1}{q^k-q_V} \leq (\frac{q^k}{q^k-q_V} - 1) + \frac{1}{q^k-q_V} = \frac{q_V+1}{q^k-q_V}$. □

[4] The upper bound is immediate. The lower bound can be obtained from the following expression of the Euler-Mascheroni constant γ in terms of the harmonic numbers and the Hurwitz zeta function $\zeta(a,b)$: $\gamma = H_n - \ln(n) - \sum_{i=2}^{\infty} \frac{\zeta(i,n+1)}{i}$. One can prove that this implies that $\gamma = H_n - \ln(n) - \frac{1}{2n} + \frac{1}{12n^2} - \frac{1}{120n^4} + \epsilon$ for $0 < \epsilon < \frac{1}{252n^6}$. Isolating H_n on the right hand side, and observing that $\frac{1}{2n+1} \leq \frac{1}{2n} - \frac{1}{12n^2}$ for all $n \in \mathbb{N}$ yields the lower bound.

Lemma 2. *For every $i = 1$ to $q_V + 1$, it holds* $\Pr\left[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\right] \leq \frac{1}{q^k-(i-1)}$.

Proof of Lemma 2 To evaluate the probability given in Equation (6) we need to analyze the leakage $\mathcal{A}$ can get from verification queries. First of all, by correctness $\mathsf{onVer}(\mathsf{svk}, \mu_i, \boldsymbol{\sigma}_i) = 0 \Rightarrow \mathsf{Ver}(\mathsf{pk}, \mu_i, \boldsymbol{\sigma}_i) = 0$ and $\mathsf{Ver}(\mathsf{pk}, \mu_i, \boldsymbol{\sigma}_i) = 1 \Rightarrow \mathsf{onVer}(\mathsf{svk}, \mu_i, \boldsymbol{\sigma}_i) = 1$ for every possible svk generated from pk. Leakage about svk happens when: (1) an event bad_i occurs ($O\mathsf{onVer}$ accepts where the standard verification would reject); and (2) $O\mathsf{onVer}$ rejects a query (here $\mathcal{A}$ may learn that some combination of rows of pk must appear in svk). Equation 6 gives us a way to bound the adversary's advantage (and thus, the magnitude of this leakage) in terms of the events bad_i and $\neg\mathsf{bad}_i$. The goal of this lemma is to give a generic structure for estimating the probabilities in Equation 6; the concrete values are calculated in the Lemma 3.

Consider the i-th query $(\mu_i, \boldsymbol{\sigma}_i)$ to $O\mathsf{onVer}$. If the oracle returns 0, the adversary knows that $\mathbf{C} \cdot (\mathbf{M}_i \cdot \mathbf{v}_i) \neq 0 \mod q$. In other words, *not all* rows of $\mathbf{C} \in \mathcal{C} := \{\mathbf{C} \in \mathbb{Z}_q^{k\times n} \ : \ rk(\mathbf{C}) = k\}$ are vectors in the hyperplane orthogonal to $\mathbf{w}_i := \mathbf{M}_i \cdot \mathbf{v}_i$ (which is known to $\mathcal{A}$, as $(\mathbf{M}_i, \mathbf{v}_i)$ can be computed from the pk, μ_i and $\boldsymbol{\sigma}_i$). Actually, *some* rows of $\mathbf{C}$ may be orthogonal to $\mathbf{w}_i$, however since $O\mathsf{onVer}$ returned 0, there exists at least one row $\mathbf{c}_j$ for which $\mathbf{c}_j \cdot \mathbf{w}_i \neq 0 \mod q$. For convenience we introduce the sets $\mathcal{H}_i \subseteq \mathcal{C}$ of full-rank matrices $\mathbf{C} \in \mathcal{C}$ whose rows are all orthogonal to $\mathbf{w}_i$, formally:

$$\mathcal{H}_i := \left\{ \mathbf{C} \in \mathcal{C} \ : \ \mathbf{C} = \begin{bmatrix} \mathbf{c}_1 \\ \dots \\ \mathbf{c}_k \end{bmatrix} \wedge \ \mathbf{c}_j \cdot \mathbf{w}_i = 0 \mod q \ \forall \ j = 1, \dots, k \right\} .$$

For simplicity, assume $\mathcal{A}$ be able to pick the vectors $\mathbf{w}_i \in \mathbb{Z}_q^n \setminus \{0\}$ of her choosing (e.g., by generating suitable pairs $(\mu_i, \boldsymbol{\sigma}_i)$).[5]

At the first verification query $(\mu_1, \boldsymbol{\sigma}_1)$, $\mathcal{A}$ has no information about $\mathbf{C}$ beyond the fact that it was uniformly sampled from the set $\mathcal{C} := \{\mathbf{C} \in \mathbb{Z}_q^{k\times n} \ : \ rk(\mathbf{C}) = k\}$. Therefore, for any choice of $\mathbf{w}_1 \neq 0$, if the event bad_1 occurs, then $\mathsf{bad}_1 = \{\mathbf{C} \cdot \mathbf{w}_1 = 0 \mod q \ \wedge \ \mathbf{C} \xleftarrow{\$} \mathcal{C}\}$, thus $\Pr[\mathsf{bad}_1] = \Pr\left[\mathbf{C} \cdot \mathbf{w}_1 = 0 \mod q \ \wedge \ \mathbf{C} \xleftarrow{\$} \mathcal{C}\right] = \frac{|\mathcal{H}_1|}{|\mathcal{C}|}$. The first (rejected) verification query leaks the fact that $\mathbf{C} \in \mathcal{C} \setminus \mathcal{H}_1$.

For the second verification query, we assume, without loss of generality, that $\mathcal{A}$ chooses $\mathbf{w}_2$ linearly independent from $\mathbf{w}_1$, i.e., $\mathbf{w}_2 \notin \langle \mathbf{w}_1 \rangle_q$. In this case, we have

$$\begin{aligned}
\Pr[\mathsf{bad}_2 \mid \neg\mathsf{bad}_1] &= \Pr[\mathbf{C} \cdot \mathbf{w}_2 = 0 \mod q \mid \mathbf{C} \xleftarrow{\$} \mathcal{C} \ \wedge \ \mathbf{C} \in (\mathcal{C} \setminus \mathcal{H}_1)] \\
&= \frac{\Pr\left[\mathbf{C} \cdot \mathbf{w}_2 = 0 \mod q \ \wedge \ \mathbf{C} \xleftarrow{\$} \mathcal{C} \ \wedge \ \mathbf{C} \in (\mathcal{C} \setminus \mathcal{H}_1)\right]}{\Pr\left[\mathbf{C} \xleftarrow{\$} \mathcal{C} \ \wedge \ \mathbf{C} \in (\mathcal{C} \setminus \mathcal{H}_1)\right]} \\
&\leq \frac{\Pr\left[\mathbf{C} \cdot \mathbf{w}_2 = 0 \mod q \ \wedge \ \mathbf{C} \xleftarrow{\$} \mathcal{C}\right]}{\Pr\left[\mathbf{C} \xleftarrow{\$} \mathcal{C} \ \wedge \ \mathbf{C} \in (\mathcal{C} \setminus \mathcal{H}_1)\right]} \\
&= \frac{\frac{|\mathcal{H}_2|}{|\mathcal{C}|}}{\frac{|\mathcal{C}\setminus\mathcal{H}_1|}{|\mathcal{C}|}} = \frac{|\mathcal{H}_1|}{|\mathcal{C} \setminus \mathcal{H}_1|}
\end{aligned}$$

[5] This assumption is generous as it gives the adversary a large amount of power and freedom in the game. Any weaker adversary will have even less probability of winning the security experiment. Notice that the constraint $\mathbf{w}_1 \neq 0$ implies $\mathsf{Ver}(\mathsf{pk}, \mu_1, \boldsymbol{\sigma}_1) = 0$, which is a necessary condition for $O\mathsf{onVer}$ leaking information about svk.

where the inequality follows from the fact that, given three events E_1, E_2, E_3, it always holds that $\Pr[E_1 \wedge E_2 \wedge E_3] \leq \min\{\Pr[E_1 \wedge E_2], \Pr[E_1 \wedge E_3], \Pr[E_2 \wedge E_3]\}$; and the last equality is due to the fact that the hyperplane $\mathcal{H}_1$ and $\mathcal{H}_2$ have the same dimension.

The same reasoning applies to the generic i-th verification query, where, w.l.o.g., $\mathcal{A}$ chooses $\mathbf{w}_i$ outside of the space generated by the previous $\mathbf{w}_j$'s, i.e., $\mathbf{w}_i \notin \langle \mathbf{w}_1, \ldots, \mathbf{w}_{i-1} \rangle_q$. At such query, $\mathcal{A}$ knows that $\mathbf{C} \in \mathcal{C} \setminus \left(\bigcup_{j=1}^{i-1} \mathcal{H}_j \right)$. Analogously as before we get that

$$\begin{aligned} \Pr\left[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg \mathsf{bad}_j \right] &\leq \frac{\Pr\left[\mathbf{C} \cdot \mathbf{w}_i = 0 \mod q \ \wedge \ \mathbf{C} \xleftarrow{\$} \mathcal{C} \right]}{\Pr\left[\mathbf{C} \in \mathcal{C} \setminus \left(\bigcup_{j=1}^{i-1} \mathcal{H}_j \right) \ \wedge \ \mathbf{C} \xleftarrow{\$} \mathcal{C} \right]} \\ &= \frac{|\mathcal{H}_1|}{\left| \mathcal{C} \setminus \left(\bigcup_{j=1}^{i-1} \mathcal{H}_j \right) \right|} \ . \end{aligned} \tag{9}$$

Lemma 3 concludes the proof of Theorem 1 by showing that for $i > 1$: $\left| \mathcal{C} \setminus \left(\bigcup_{j=1}^{i-1} \mathcal{H}_j \right) \right| \geq |\mathcal{H}_1| \cdot \left(\frac{q^n - q}{q^{n-k} - 1} - (i-1) \right)$. Substituting this value into Equation (9) returns:

$$\begin{aligned} \Pr[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg \mathsf{bad}_j] &\leq \frac{|\mathcal{H}_1|}{|\mathcal{H}_1| \left(\frac{q^n - q}{q^{n-k} - 1} - (i-1) \right)} = \frac{1}{q^k \cdot \frac{1 - q^{1-n}}{1 - q^{k-n}} - (i-1)} \\ &\leq \frac{1}{q^k - (i-1)} \ , \end{aligned}$$

where the last bound follows from the chain:

$$q^{n-1} > q^{n-k} \ \Leftrightarrow \ \frac{1}{q^{n-1}} < \frac{1}{q^{n-k}} \ \Leftrightarrow \ 1 - \frac{1}{q^{n-1}} > 1 - \frac{1}{q^{n-k}} \ \Leftrightarrow \ \frac{1 - \frac{1}{q^{n-1}}}{1 - \frac{1}{q^{n-k}}} > 1 \ ,$$

as $1 < k < n$ and $q > 1$.

Lemma 3. *Let $\mathcal{C} := \{\mathbf{C} \in \mathbb{Z}_q^{k \times n} \ : \ rk(\mathbf{C}) = k\}$ be the set of $k \times n$ matrices that are full rank and have entries in $\mathbb{Z}_q$ (as introduced in the proof of Lemma 2). For any given set of vectors $\{\mathbf{w}_1, \ldots, \mathbf{w}_{q_V+1}\} \subseteq \mathbb{Z}_q^n$, such that for every $i \neq j, \mathbf{w}_i \notin \langle \mathbf{w}_j \rangle_q$ define the collection of sets $\{\mathcal{H}_i := \{\mathbf{C} \in \mathcal{C} \ : \ \mathbf{C} \cdot \mathbf{w}_i = 0 \mod q\}\}_{i=0}^{q_V+1} \subseteq \mathcal{C}$ containing the matrices having the corresponding $\mathbf{w}_i$ in their right kernel and $\mathcal{H}_0 = \{\varnothing\}$. It holds that*

$$\left| \mathcal{C} \setminus \left(\bigcup_{j=0}^{i-1} \mathcal{H}_j \right) \right| \geq |\mathcal{H}_1| \cdot \left(\frac{q^n - 1}{q^{n-k} - 1} - (i-1) \right) \quad \forall \ i = 1, \ldots, q_V. \tag{10}$$

Proof of Lemma 3 It is easy to see that the cardinality of $\mathcal{C}$ is:

$$|\mathcal{C}| = (q^n - 1) \cdot (q^n - q) \cdot (q^n - q^2) \cdot \ldots \cdot (q^n - q^{k-1}) = q^{\frac{k(k-1)}{2}} \cdot \prod_{j=0}^{k-1} (q^{n-j} - 1).$$

as we have full freedom for how to pick the first row of $\mathbf{C}$ (except for $\mathbf{c}_1 \neq 0$); for the second row, we can pick any vector $\mathbf{c}_2$ that is in $\mathbb{Z}_q^n$ but not in the span of the previous row of $\mathbf{C}$ (to keep the

matrix full rank), and so on. The formula after the final equality follows from decomposing each $(q^n - q^j)$ factor as $q^j(q^{n-j} - 1)$ and observing that $\prod_{j=1}^{k-1} q^j = q^{\frac{k(k-1)}{2}}$. The same reasoning applies to computing the cardinality of a generic $\mathcal{H}_i$ (for $i > 0$), after noticing that the rows of the matrices $\mathbf{C} \in \mathcal{H}_i$ must be picked (as linearly independent vectors) from the $(n-1)$-dimensional hyperplane orthogonal to $\mathbf{w}_i$; thus

$$|\mathcal{H}_i| = \prod_{j=0}^{k-1}(q^{n-1} - q^j) = q^{\frac{k(k-1)}{2}} \cdot \prod_{j=0}^{k-1}(q^{(n-1)-j} - 1) = q^{\frac{k(k-1)}{2}} \cdot \prod_{j=1}^{k}(q^{n-j} - 1)$$

This proves the base case ($i = 1$) of the bound in (10): $|\mathcal{C} \setminus \varnothing| = |\mathcal{H}_1| \cdot \frac{q^n-1}{q^{n-k}-1}$, since, compared to $|\mathcal{H}|$, $|\mathcal{C}|$ has the additional factor $j = 0$ and the missing factor $j = k$. Concretely this means that: $\Pr[\mathsf{bad}_1] \leq \frac{q^{n-k}-1}{q^n-1}$. For $i = 2$, again $|\mathcal{C} \setminus \mathcal{H}_1| = |\mathcal{C}| - |\mathcal{H}_1| = |\mathcal{H}_1|\left(\frac{q^n-1}{q^{n-k}-1} - 1\right)$. For $i = 3$, we remove from the pool of eligible $\mathbf{C}$ all those matrices in $\mathcal{H}_1 \cup \mathcal{H}_2$, i.e., that have either $\mathbf{w}_1$ or $\mathbf{w}_2$ in their right kernel.[6] In other words, matrices composed by only rows orthogonal to $\mathbf{w}_1$ or to $\mathbf{w}_2$. The hyperplanes $\mathbf{w}_1^\perp$ and $\mathbf{w}_2^\perp$ both have dimension $n-1$, and since we are in a space of dimension n, they must intersect in a subspace of dimension $n-2$. For a tighter bound we use: $|\mathcal{H}_2 \setminus \mathcal{H}_1| = |\mathcal{H}_2| - |\mathcal{H}_1 \cap \mathcal{H}_2|$ and recall that $0 \leq |\mathcal{H}_2| - |\mathcal{H}_1 \cap \mathcal{H}_2| \leq |\mathcal{H}_1|$. Hence after the second rejected query the number of possible $\mathbf{C}$ becomes:

$$\begin{aligned}
&|\mathcal{C} \setminus (\mathcal{H}_1 \cup \mathcal{H}_2)| = |\mathcal{C}| - |\mathcal{H}_1| - |\mathcal{H}_2| + |\mathcal{H}_1 \cap \mathcal{H}_2| \\
&= |\mathcal{H}_1| \cdot (\frac{q^n - 1}{q^{n-k} - 1} - 2) + |\mathcal{H}^{(n-2)}| \\
&= |\mathcal{H}_1| \cdot (\frac{q^n - 1}{q^{n-k} - 1} - 2) + q^{k(k-1)/2} \prod_{j=2}^{k-1}(q^{n-j} - 1) \\
&= |\mathcal{H}_1| \cdot (\frac{q^n - 1}{q^{n-k} - 1} - 2 + \frac{1}{(q^{n-1} - 1)(q^{n-k} - 1)}) \geq |\mathcal{H}_1| \cdot (\frac{q^n - 1}{q^{n-k} - 1} - 2).
\end{aligned}$$

Remark that $\mathbf{C}$ could be still composed by some elements of $\mathcal{H}_1$ *and* some of $\mathcal{H}_2 \setminus \mathcal{H}_1$; this would be consistent with $\mathcal{A}$'s view at this point.

We can now proceed by induction, assuming (10) holds for the query index i, prove it for $i+1$.

$$\begin{aligned}
\left|\mathcal{C} \setminus \bigcup_{j=0}^{i} \mathcal{H}_j\right| &= \left|\mathcal{C} \setminus \left((\bigcup_{j=0}^{i-1} \mathcal{H}_j) \cup \mathcal{H}_i\right)\right| \\
&= |\mathcal{C}| - \left|\bigcup_{j=0}^{i-1} \mathcal{H}_j\right| - |\mathcal{H}_i| + \left|\left(\bigcup_{j=0}^{i-1} \mathcal{H}_j\right) \cap \mathcal{H}_j\right| \\
&= \left|\mathcal{C} \setminus \bigcup_{j=0}^{i-1} \mathcal{H}_j\right| - |\mathcal{H}_1| + \left|\left(\bigcup_{j=0}^{i-1} \mathcal{H}_j\right) \cap \mathcal{H}_j\right| \geq \left|\mathcal{C} \setminus \bigcup_{j=0}^{i-1} \mathcal{H}_j\right| - |\mathcal{H}_1| \\
&\geq |\mathcal{H}_1| \cdot \left(\frac{q^n - 1}{q^{n-k} - 1} - i + 1\right) - |\mathcal{H}_1| = |\mathcal{H}_1| \cdot \left(\frac{q^n - 1}{q^{n-k} - 1} - i\right).
\end{aligned}$$

□

[6] By assumption the vectors $\mathbf{w}_i$ are not multiples of one another. Otherwise, $\mathcal{A}$ does not extract any new information from the rejected query, i.e., there is no additional leakage.

2.2 Concrete Instantiations of Our Compiler

Because any instantiation of our compiler is completely determined by the four subroutines **parse** pk, GetM, Check, and GetZV, in what follows we explain only how these four algorithms work. The complete descriptions of offVer and onVer are derived using the general structure detailed in Figure 3.

From Lattices We present concrete instantiations of our compiler for three categories of LBS: 'hash & sign' with representative the GPV08 signature [20], 'Boyen/BonsaiTree' style with representative MP12 [29], and 'Fiat-Shamir with Abort' style with representative Lyu12 [26] (with parameters from [12]).

Efficient Verification for GPV08 [20]. The **parse** pk procedure splits the public key into $PK = \mathbf{A} \in \mathbb{Z}_q^{n\times m}$ (the matrix identifying the signer's public key), and the auxiliary public information $PK.\mathsf{aux} = (\mathcal{H}, \beta)$, i.e., a description of a full-domain hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q^n$ and the norm bound $\beta \in \mathbb{R}$. The Check procedure is exactly as in the original verification (enforcing the norm bound β on the signature). The GetM algorithm takes in input the public matrix $PK = \mathbf{A}$, and tails to it the identity matrix: $\mathbf{M} = [\mathbf{A}|-\mathbf{I}_n]$. The GetZV routine returns the matrix $\mathbf{Z}'$ (explained momentarily) and the vector $\mathbf{v} = [\boldsymbol{\sigma}|\mathcal{H}(\mu)\cdot\mathbf{1}_{1\times n}]$. The matrix $\mathbf{Z}'$ is made up of the same 'randomized key' vectors produced by GetZ during the offline verification, i.e., $\mathbf{z}'_j = \mathbf{z}_j \leftarrow \mathbf{c}_j\mathbf{M} = [\mathbf{c}_j\mathbf{A}|-\mathbf{c}_j]$. Thus the core verification check (line 7 in onVer) is actually ensuring that $\mathbf{z}'_j\mathbf{v}_j = 0$, i.e., $\mathbf{c}_j\cdot\mathbf{A}\cdot\boldsymbol{\sigma} = \mathbf{c}_j\mathcal{H}(\mu)$ which is the probabilistic check of the original verification equality.

Efficient Verification for MP12 [29]. The **parse** pk procedure assigns $PK \leftarrow \mathbf{A} = [\tilde{\mathbf{A}}|\mathbf{A}_0 \mid \ldots \mid \mathbf{A}_\ell] \in \mathbb{Z}_q^{n\times(\bar{m}+n\lceil\log q\rceil\ell)}$ (the matrix identifying the signer's public key), where $\bar{m} = O(n\lceil\log q\rceil)$, and ℓ denotes the number of bits in the message, i.e., $\mu \in \{0,1\}^\ell$. The auxiliary public information is $PK.\mathsf{aux} = (\mathbf{u}, \beta)$. The Check procedure is exactly as in the original verification (enforcing the norm bound β on the signature). The GetM algorithm takes in input the public matrix $PK = \mathbf{A}$, and appends to it the identity matrix to obtain $\mathbf{M} = [\mathbf{A}|-\mathbf{I}_{n\times n}]$. The GetZV routine returns the matrix $\mathbf{Z}'$ and the vector $\mathbf{v}$. The matrix $\mathbf{Z}'$ is made up of vectors of the form $\mathbf{z}'_j = [\tilde{\mathbf{z}}_j \mid \mathbf{z}_j^0 + \sum_{i=1}^{\ell}\mu[i]\mathbf{z}_j^i|\mathbf{c}_j]$ that identify a message-dependent lattice (called $\mathbf{A}_\mu$ in [29]). The vector $\mathbf{v}$ is the concatenation of the signature with the auxiliary vector, i.e., $\mathbf{v} = [\boldsymbol{\sigma}|\mathbf{u}]$. Note that $\mathbf{u}$ is the same for all messages; thus, one could further optimize the online verification by computing (once and for all) the k inner products $\mathbf{z}_j[\bar{m}+n\lceil\log q\rceil+1] = \mathbf{c}_j\cdot\mathbf{u}$ during the offline phase. To conclude we notice that the online verification ensures that $\mathbf{z}'_j\mathbf{v}_j = 0$, i.e., $\mathbf{c}_j\cdot\mathbf{A}_\mu\cdot\boldsymbol{\sigma} = \mathbf{c}_j\cdot\mathbf{u}$ which is the probabilistic check of the original verification equality.

Efficient Verification for Lyu12 [26]. Until recently, our compiler seemed not applicable to the most efficient schemes among lattice-based signatures: the ones relying on the combination of an identification scheme with the Fiat-Shamir with aborts technique, introduced by Lyubashevsky [26]. Indeed, the (efficient version of the) verification for these signatures required to evaluate a hash function besides a norm check[7]. So far, the scheme seemed to require such a hash to be a cryptographic hash, thus its computation would not be of type $\mathbf{Mv}$. However, Chen, Lombardi, Ma

[7] This is because short signatures require the prover/signer to send only the challenge c and response $\mathbf{z}$, not the commit, thus the verifier has to check the norm of the response, and to verify that the hash of the message and some lienar combination of the public parameters and the response yields the challenge.

and Quach [12] have presented a work at CRYPTO 2021 that proves this limitation unnecessary: indeed, they proved that such signatures are secure when instantiated with a hash function $\mathcal{H}$ such that $\mathcal{H}(\mathbf{t}, \mu) = \mathbf{y}$ where $\mathbf{y}$ is such that $(\mathbf{t} - \mu) = \mathbf{G} * \mathbf{y}$, $\mu \in \mathbb{Z}_q^{\bar{m}}$ is the message to be signed, and $\mathbf{G} \in \mathbb{Z}_q^{n \times (n\lceil \log q \rceil)}$ is the standard gadget matrix; i.e., $\mathbf{y}$ is the bit decomposition of $\mathbf{t} - \mu$. Using this hash changes the verification algorithm to be of type "norm check + verification of a linear equation": upon receiving a message μ and a signature $(\mathsf{c}, \mathbf{z})$, verifying that $\mathsf{c} = \mathcal{H}(\mathbf{Az} - \mathbf{uc}, \mu)$ now requires to compute $\mathbf{Az} - \mathbf{G} * \mathsf{c} - \mu - \mathbf{u} * \mathsf{c} = 0$. Our compiler can now be applied to such a verifier as follows. Our compiler for efficient verification applies to [26] when the hash function employed is non-cryptographic, for instance the bit decomposition function, as proposed in [12]. In this case, the **parse** pk procedure sets $PK \leftarrow [\mathbf{A}|\mathbf{u}] \in \mathbb{Z}_q^{n \times (\bar{m}+1)}$, where $\bar{m} = n\lceil \log q \rceil$. Messages are vectors, i.e., $\mu \in \mathbb{Z}_q^m$. The auxiliary public information is $PK.\mathsf{aux} = (\mathbf{G}, \beta)$. The algorithm GetM returns $\mathbf{M} = [\mathbf{A}|\mathbf{G} + \mathbf{u}|\mathbf{I}_n] \in \mathbb{Z}_q^{n \times (\bar{m}+n\lceil \log q \rceil + n)}$. The Check procedure is exactly as in the original verification (enforcing the norm bound β on the signature). The GetZV routine returns a matrix $\mathbf{Z}'$ composed by the same 'randomized key' vectors produced by GetZ during the offline verification, i.e., $\mathbf{z}'_j = \mathbf{z}_j \leftarrow \mathbf{c}_j\mathbf{M}$, while the vector $\mathbf{v}$ is constructed as $\mathbf{v} = [\mathbf{z}| - \mathsf{c}| - \mu]$. Then, the online verification algorithm checks that $\mathbf{Z}'\mathbf{v} = 0$, i.e., $\mathbf{c}_j[\mathbf{A}|\mathbf{G} + \mathbf{u}|\mathbf{I}_n] \cdot [\mathbf{z}| - \mathsf{c}| - \mu] = 0$, which is a probabilistic variant of the original verification equality.

From Multivariate Equations For signatures based on multivariate equations we take Rainbow [14,15] as representative example as this is one of the NIST candidates for standardization. For completeness, we also show how to apply our compiler to the LUOV scheme [5].

Efficient Verification for Rainbow [14]. In the description below we consider the standard Rainbow verification. A similar approach can be used to speed up the verification also in the "cyclic" and the "compressed" Rainbow variants as in those cases the verification includes an additional initial phase to reconstruct the full public key. We recall that in this scheme the public key contains a system of m multivariate quadratic polynomials in n variables. For convenience, let $N = n(n+1)/2$ and $\mathbb{F} = \mathbb{F}_{2^r}$. Using a Macaulay matrix representation we can visualize this system as a wide matrix composed of a quadratic term $\mathbf{Q}$ (actually a $m \times N$ submatrix), a linear term $\mathbf{L}$ ($m \times n$ submatrix) and a constant term $\mathbf{C}$ (a $m \times 1$ vector). The **parse** pk procedure extracts from the public key PK this matrix $\mathsf{pk} = [\mathbf{Q}|\mathbf{L}|\mathbf{C}] \in \mathbb{F}^{m \times (N+n+1)}$ and a description of a full-domain hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{F}^m$ as the auxiliary public information $PK.\mathsf{aux} = \mathcal{H}$. The Check procedure is trivial and always returns 1. This is because the whole verification can be written as a matrix-vector multiplication. The GetM algorithm extracts from PK the matrix representing the system of quadratic multivariate equations $[\mathbf{Q}|\mathbf{L}|\mathbf{C}]$. Finally, it appends to this the identity matrix, so $\mathbf{M} \leftarrow [\mathbf{Q}|\mathbf{L}|\mathbf{C}| - \mathbf{I}_m]$. We remark that $\mathbf{M}$ can be seen as a matrix of blocks, where any block has the same height (m = number of rows), but different length (number of columns). The GetZV routine reads the matrix $\mathbf{Z}' = \mathbf{Z}$ made up of the rows $\mathbf{z}'_j = \mathbf{z}_j \leftarrow \mathbf{c}_j\mathbf{M} = [\mathbf{c}_j \cdot \mathbf{Q}|\mathbf{c}_j \cdot \mathbf{L}|\mathbf{c}_j \cdot \mathbf{C}| - \mathbf{c}_j] \in \mathbb{F}^{1 \times N+n+1}$. In addition, this algorithm parses the signature as $\boldsymbol{\sigma} = (\mathbf{s}, \mathsf{salt})$, computes the (salted) hash of the message $\mathbf{d}$ as $\mathbf{h} \leftarrow \mathcal{H}(\mathcal{H}(\mathbf{d})|\mathsf{salt})$ and outputs the vector $\mathbf{v} = [\tilde{\mathbf{s}}|\mathbf{s}|1|\mathbf{h}]$, where $\mathbf{s}$ is part of the signature and $\tilde{\mathbf{s}}$ is the 'quadratic vector' obtained by computing all products of pairs of elements in $\mathbf{s}$ (with monomials ordered lexicographically), i.e., $\tilde{\mathbf{s}} \leftarrow [\mathbf{s}[1]^2, \mathbf{s}[1]\mathbf{s}[2], \ldots, \mathbf{s}[n-1]\mathbf{s}[n], \mathbf{s}[n]^2]$. Clearly $\mathbf{z}'_j \cdot \mathbf{v} = 0$ if and only if $\mathbf{c}_j \cdot (\mathbf{Q}\tilde{\mathbf{s}} + \mathbf{Ls} + \mathbf{C}) = \mathbf{c}_j \cdot \mathbf{h}$, which is a probabilistic check of the original system of verification equations in Rainbow.

Efficient Verification for LUOV [5]. The **parse** pk procedure splits the public key into $PK = (\mathsf{public.seed}, \mathbf{Q}_2)$ (the concise information needed to retrieve the full signer's public key), and the auxiliary public information $PK.\mathsf{aux} = \mathcal{H}$, i.e., a description of a full-domain hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{F}^m$, where $m = rows(\mathbf{Q}_2)$ and $\mathbb{F} = \mathbb{F}_{2^r}$. The Check procedure is trivial and always returns 1. This is because the whole LUOV verification can be written as a matrix-vector multiplication The GetM algorithm takes in input $PK = (\mathsf{public.seed}, \mathbf{Q}_2)$ and derives the full public key as done in the original verification: it runs $[\mathbf{C}||\mathbf{L}||\mathbf{Q}_1] \leftarrow \mathcal{G}(\mathsf{public.seed})$ to get the constant constant (vector), the linear (matrix) and the first quadratic (matrix) parts of the verification equation; and then it reconstructs the full quadratic term as $\mathbf{Q} \leftarrow [\mathbf{Q}_1||\mathbf{Q}_2]$. Finally it appends to the public key the identity matrix $\mathbf{M} \leftarrow (\mathbf{C}, \mathbf{L}, \mathbf{Q}, -\mathbf{I}_{rows(\mathbf{Q})})$, we remark that $\mathbf{M}$ can be seen as a matrix of blocks, where any block has the same height (number of rows), but different lenghth (number of columns). The GetZV routine reads the matrix $\mathbf{Z}' = \mathbf{Z}$ made up of the rows $\mathbf{z}'_j = \mathbf{z}_j \leftarrow \mathbf{c}_j\mathbf{M} = (\mathbf{c}_j \cdot \mathbf{C}, \mathbf{c}_j \cdot \mathbf{L}, \mathbf{c}_j \cdot \mathbf{Q}, -\mathbf{c}_j)$. It also outputs the vector $\mathbf{v} = [1|\mathbf{s}|\tilde{\mathbf{s}}|\mathbf{h}]$, where $\mathbf{s}$ is part of the signature $\boldsymbol{\sigma} = (\mathbf{s}, \mathsf{salt})$, $\tilde{\mathbf{s}}$ is the 'quadratic vector' obtained by computing all products of pairs of elements in $\mathbf{s}$, i.e., $\tilde{\mathbf{s}} \leftarrow [\mathbf{s}[1]^2, \mathbf{s}[1]\mathbf{s}[2], \ldots, \mathbf{s}[n-1]\mathbf{s}[n], \mathbf{s}[n]^2]$, finally $\mathbf{h}$ is the hash of the message and the salt, i.e., $\mathbf{h} \leftarrow \mathcal{H}(\mu||0\mathsf{x}0||\mathsf{salt})$. Clearly $\mathbf{z}'_j \cdot \mathbf{v} = 0$ if and only if $\mathbf{c}_j \cdot (\mathbf{C} + \mathbf{L}\mathbf{s} + \mathbf{Q}\tilde{\mathbf{s}}) = \mathbf{c}_j \cdot \mathbf{h}$, which is a probabilistic check of the original verification equation in LUOV.

In what follows, we evaluate the efficiency gains provided by our compiler using the $(\mathsf{r}_0, \mathsf{e}_0)$-concrete efficiency notion of Equation (2). In brief, a Σ^E achieves $(\mathsf{r}_0, \mathsf{e}_0)$-concrete amortized efficiency if r_0 is the smallest, non-negative integer for which it holds that $\mathsf{e}_0 < 1$, where e_0 is an upperbound on the ratio between the cost of running the offline verification once and using its outcome in r_0 online verifications, over the cost of running r_0 standard signature verifications. For convenience, we estimate only the cost of the most expensive 'steps' in the verification, namely the ones involving several field element *multiplications* (e.g., matrix-vector products), and disregard the cost of adding elements, generating random values, reading algorithm inputs or evaluating hash functions. Moreover, we do not consider ad-hoc optimizations of matrix multiplication due to probabilistic checks using, e.g., Freivalds' Algorithm or its variant [32]. We remark that in this work, we develop a more general and full-fledged approach that is not limited to the specific technique in [32]. Table 1 collects the common notation, while Table 2 displays a summary of our findings, that we motivate below.

Table 1: Parameters involved in the performance analysis of our compiler for efficient verification.

q	Modulus of the lattice or size of the field
n	Number of rows in the public key
$m \in \Omega(n \log q)$	Number of columns in the public key
β	Bound on the noise / size of signatures
$\boldsymbol{\sigma}$ or $\mathbf{U}$	Vector or matrix signatures
k	Number of steps in the online verification (confidence level)
r	Number of signatures verified (repetitions of onVer)
$\mathsf{cost}(alg)$	Number of field multiplications needed to compute alg

The computational complexity of Ver for signature with $\mathbf{Mv}$-style verification, e.g., [5,14,15,20,29,21] and [26] instantiated with non-cryptographic hash functions á la [12], is dominated by a matrix-vector multiplication. Let $n = rows(\mathbf{M})$ and $m = cols(\mathbf{M})$, with $m \geq n$. The cost of computing

Table 2: A summary of the concrete efficiency achieved by various instatiations of our compiler for efficient verification. In the table, k_0 denotes the minimum accuracy level that ralizes efficient verification with 128 bits of security, i.e., for which $\Pr[\mathsf{Bad}] \leq 2^{-128}$ is negligible (cf. proof of Theorem 1, with $q_V = 2^{30}$); r_0 is the smallest positive integer for which $\frac{\mathsf{cost}(\mathsf{offVer}(\mathsf{pk},k_0)+\mathsf{r}\cdot\mathsf{cost}(\mathsf{onVer})}{\mathsf{r}\cdot\mathsf{cost}(\mathsf{Ver})} < 1$, and e_0 is a (tight) upperbound on this ratio.

Ring or Field Size (representative schemes)	**Min. Accuracy Level for 128-bit security**	**Concrete Amortized Efficiency (see Definition 2)**	**Online Efficiency** $\frac{\mathsf{cost}(\mathsf{onVer})}{\mathsf{cost}(\mathsf{Ver})} = \frac{k_0}{n}$
exponential: $q = 2^{128}$ (FMNP [16]; GVW [21])	$k_0 = 1$	$(\mathsf{r}_0 = 2, \mathsf{e}_0 = 0.51)$	$\frac{1}{256} < 0.4\%$
large poly.: $q = 2^{30}$ (Boyen [8]; GPV [20]; MP [29])	$k_0 = 5$	$(\mathsf{r}_0 = 6, \mathsf{e}_0 = 0.86)$	$\frac{5}{256} < 2\%$
mid-size poly.: $q = 2^{26}$ (Lyub [26])	$k_0 = 7$	$(\mathsf{r}_0 = 8, \mathsf{e}_0 = 0.89)$	$\frac{7}{512} < 1.4\%$
small poly.: $q = 16$ (Rainbow [15] $\mathbb{F}_{2^4}$-$(32, 32, 32)$)	$k_0 = 32$	$(\mathsf{r}_0 = 65, \mathsf{e}_0 = 0.99)$	$\frac{32}{64} = 50\%$

$\mathbf{M} \cdot \mathbf{v}$ is, in the worst case, nm filed multiplications. Our offline verification algorithm executes k vector-matrix multiplications (one for each $\mathbf{z}'_j$ in $\mathbf{Z}'$), resulting in knm multiplications in the worst case. The computational complexity of our online verification is dominated by the k vector-vector (inner) products $\mathbf{z}_i \cdot \mathbf{v}$, resulting in km multiplications in the worst case. Thus, the compiler presented in Section 2.1 outputs an efficient verification for signature with $\mathbf{Mv}$-style verification that has the following concrete amortized efficiency:

$$\frac{\mathsf{cost}(\mathsf{offVer}) + \mathsf{r} \cdot \mathsf{cost}(\mathsf{onVer})}{\mathsf{r} \cdot \mathsf{cost}(\mathsf{Ver})} = \frac{knm + \mathsf{r}km}{\mathsf{r}nm} = \frac{k}{\mathsf{r}} + \frac{k}{n}. \tag{11}$$

Clearly the first addend in Equation (11) comes from amortizing the cost of offVer (over verifying r signatures), while the second term is the fix trade-off between the computational costs of onVer and Ver (at each and every verification). Table 2 collects the figures for four representative classes of signature schemes, if we apply our compiler for efficient verification at 128 bit of security. The values are extrapolated as explained in the reminder of the section. In detail, k_0 depends on the signature scheme Σ as it is the minimal value of the confidence level k for which Σ^E is existentially unforgeable; k_0 determines the length of the svk. The value r_0 is the minimum number of verifications to run in order to achieve a concrete efficiency gain of e_0. Thus, lower values of e_0 and r_0 correspond to better efficiency gains. The last column in Table 2 displays the ratio k_0/n that essentially tells how much *cheaper* onVer is compared to the original verification Ver (ignoring the one-time cost of running offVer). Again, lower values in this column correspond to better efficiency; for instance, a ratio of 0.4% means that the computational cost of Ver is 99.6× higher than the one of onVer (in other words, onVer is expected to be about 99× faster).

For convenience, in our analysis we categorize signatures according to the size of their underlying algebraic structure (q exponential or polynomial in the security parameter).

The modulus q is exponential in λ: To the best of our knowledge, the only LBS constructions that fall in this category are the homomorphic signatures by Gorbunov et al. [21] and by Fiore et

al. [16]. In this case, using our compiler (with some caveat, as we show in the next section) yields that the advantage in the cmvEUF experiment (as per Definition 3) is negligible in the security parameter λ for any confidence level $k \geq k_0 = 1$. However, in [16,21] the complexity of Ver is dominated by the matrix-matrix multiplication $\mathbf{AU}$ where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is the fixed public key, and $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$ is the signature[8].We computed parameters for this family of schemes according to Albrecht et al.'s methodology [1]. Setting $\lambda = 128$, $q = 2^\lambda$ and $n = 256$ yields that reduction algorithms (in particular, the optimized BKZ algorithm) would have runtime 2^{128} and would solve at most $\mathsf{SIS}_{256,2^{128},65536,2^{80}}$, while the security of the scheme relies on a SIS instance with norm bound $\beta = 2^{49d}$, where d is the depth of the circuit. We can now use this set of parameters to determine the concrete amortized efficiency reached by our compiler for [16,21]. Recall formula given in Definition 2 for concrete amortized efficiency:

$$\frac{\mathsf{cost}\big(\mathsf{offVer}(\mathsf{pk}, k_0)\big) + \mathsf{r} \cdot \mathsf{cost}\big(\mathsf{onVer}(\mathsf{svk}, \mu, \boldsymbol{\sigma})\big)}{\mathsf{r} \cdot \mathsf{cost}\big(\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma})\big)} < 1$$

where k_0 is the chosen accuracy level and we are interested in finding the pair of values $(\mathsf{r}_0, \mathsf{e}_0)$ where r_0 is the minimum non-negative integer r for which this inequality hods, and $\mathsf{e}_0 < 1$ is a tight upperbound on the cost in the left hand side of the inequality when $\mathsf{r} = \mathsf{r}_0$. Notably, the better the efficiency of the offline/online verification, the smaller the value of e_0. In the case of signatures with **Mv**-style verification it is easy to see that the left hand side of the inequality can be approximated with $\frac{k_0 nm + \mathsf{r} k_0 m}{\mathsf{r} nm} = \frac{k_0}{\mathsf{r}} + \frac{k_0}{n}$. Setting $k = k_0 = 1$ and $n = 256$ we want to extract the minimum r_0 for which $1/\mathsf{r}_0 + 1/256$ is smaller than 1, formally $\mathsf{r}_0 = \min\{\mathsf{r} \in \mathbb{Z}_{>0} \mid 1/\mathsf{r} + 1/256 < 1\}$. It is easy to see that $\mathsf{r}_0 = 2$ suffices and we get $1 > \mathsf{e}_0 = 0.504 > 1/2 + 1/256$. In other words, the cost of setting up the online verification (running offVer) *plus* performing $\mathsf{r} = 2$ online verifications is about half of the cost of running 2 standard verifications, while preserving the security level. Moreover, for this set of parameters $\frac{\mathsf{cost(onVer)}}{\mathsf{cost(Ver)}} = \frac{k_0}{n} = \frac{1}{256} < 0.004$, i.e., our online verification requires about 0.4% of the computational cost of running the standard verification algorithm; alternatively, we can read this results as our onVer is $99\times$ faster than Ver.

The modulus q is a large polynomial in λ: This is the most common setting given the 'small' size of q. In this category fall the schemes by Gentry et al. [20], Boyen [8], and its improved version by Micciancio and Peikert [29], Lyubashevsky [26] with bit-decomposition as hash function [12], Boneh Franklin linearly homomorphic signature [6], Rainbow [15] and LUOV [5]. For the lattice-based constructions, in order to guarantee a negligible advantage in the cmvEUF experiment (see Definition 3) we need to set an appropriate value of $k \geq k_0 > 1$. We argue that 'appropriate' values of k are still 'small' in comparison to n and lead to a 'good' amortized efficiency even for 'few' verifications. We recall that for these constructions Ver computes a product $\mathbf{A}\boldsymbol{\sigma}$ where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and the signature is just a vector $\boldsymbol{\sigma} \in \mathbb{Z}_q^{m \times 1}$. To guarantee the security of our efficient verification, the value k should be set so that q^{-k} be negligible. In other words, for the cmvEUF advantage to be negligible it must hold that $q^{-k} \geq 2^{-\lambda}$. Hence, to estimate k, one need to first fix the value of λ, compute the corresponding q that can guarantee such level of security, and then extract the minimum value k_0 for which the above relation holds.

Computing parameters for lattice-based schemes is not straightforward, as so far there is no unique way to derive the parameters from a given λ. However, a good measure of the security

[8] In [16] the dimension m additionally depends on the number $t \geq 1$ of distinct identities (users) involved in labeled program. For simplicity, in what follows we consider $t = 1$.

of a set of parameters can be extracted computing a value δ that was introduced by Gama and Nguyen [18]. Concretely, δ provides an indication of how reduction algorithms would perform against the hardness assumption underlying the lattice-based construction. Generally, the 'smaller' the δ, the 'more secure' the scheme.

For Boyen's signature [7] and its variant by Micciancio and Peikert [29], we use the parameters provided in Figure 2 in [29]. Since in [29] they set $\delta = 1.007$, to ensure a fair comparison, we compute the parameters Gentry et al.'s signature [20] for the same value of δ. As a result, we observed that for this δ all of the schemes require about the same modulus $q = 2^{30}$ (for $n = 256$). For this set of parameters, our efficient verification provides 80 (resp. 128; 250) bits of security with just $k_0 = 3$ (resp. $k_0 = 5$; $k_0 = 9$). Thus our compiler achieves a $(4, 0.77)$-concrete amortized efficiency (resp.$(6, 0.86)$; $(10, 0.94)$), and a concrete tradeoff between onVer and Ver of $k_0/n < 0.02$ (resp. 0.02; 0.04). In particular, for the lower security settings this means that onVer is about $98\times$ faster than Ver.

The modulus q is a mid-size polynomial in λ: This is the case for Lyubashevsky's signature [26] that relies on the 'Fiat-Shamir with abort' technique. We set n and q as in column V of Figure 2 in [26]. These values are computed for $\delta = 1.007$, and columns IV and V refer to the right variant of the signature according to Section 4.5 of [12]. That yields $q = 2^{26}$ and $n = 512$, and $m = \bar{m} + n\lceil \log q \rceil + n = 27136$, as $\bar{m} = n\lceil \log q \rceil = 2^9 \cdot 26 = 13312$. For this set of parameters, our efficient verification provides 80 (resp. 128; 250) bits of security with only $k_0 = 4$ (resp. 7; 10). Thus our compiler achieves a $(5, 0.81)$-concrete amortized efficiency (resp. $(8, 0.89)$; $(11, 0.93)$), and $k_0/n < 0.01$ (resp. 0.14; 0.02).

We remark that $\delta = 1.007$ was 'enough' back in 2012, but it is now obsolete. As of now, this value of δ probably guarantees less than $\lambda = 60$ bits of security. This does not make our analysis meaningless. Indeed, the value of δ decreases for larger q and n (intuitively, this is because it is harder to find small vectors in larger lattices). In particular, the decrease in the value of δ is more dramatic for larger q than for larger n. Hence, a value of δ that would guarantee 128 bits of (post-quantum) security today implies a larger q than what we consider in this work, thus a k smaller than, e.g., $128/30 \sim 4.3 < 5$.

*For **Rainbow** *, we follow the latest guidelines provided during the second round of the NIST competition. We recall that the Rainbow signature scheme is based on the unbalanced oil and vinegar approach, and it is fully determined by the field on which it operates and a sequence of integer numbers indicating the amount of vinegar variables and oil variables per each layer. One of the current settings utilizes $\mathbb{F} = \mathbb{F}_{2^4}$ and a two-layer oil variable setting with $(v_1, o_1, o_2) = (32, 32, 32)$, which lead to $m = 96$ and $n = 64$ (for consistency in this paper we set n to be the number of rows of a matrix and m to denote the number of columns, classically the variables are swapped for multivariate signatures). A suitable k to achieve NIST security category level II is $k_0 = 32$, since $q^{-k_0} = 2^{-4 \cdot 32} = 2^{-128}$. yields that the minimum number of repetitions r_0 to achieve amortized efficiency (i.e., for which we have $k_0/\mathsf{r} + k_0/n < 1$) is $\mathsf{r}_0 = 65$, the corresponding amortization factor is $\mathsf{e}_0 = 0.9923 = 32/65 + 32/64$. For this set of parameters we have $\frac{\mathsf{cost(onVer)}}{\mathsf{cost(Ver)}} = \frac{k_0}{n} = \frac{32}{64} = 0.5$, in other words, our compiler produces an online verification that is $2\times$ *faster* than the standard verification. For $\mathbb{F} = \mathbb{F}_{2^8}$ and $(v_1, o_1, o_2) = (68, 36, 36)$, we have $m = 140$ and a suitable k in this case would be $k_0 = 16$, since $q^{-k_0} = 2^{-8 \cdot 16} = 2^{-128}$. For the highest NIST security category, [14] suggests to use $\mathbb{F} = \mathbb{F}_{2^8}$ and $(v_1, o_1, o_2) = (92, 48, 48)$. As a result we have $m = 188$, $n = 96$ and again $k_0 = 16$ but a better amortize efficiency factor $\mathsf{e}_0 = 0,9666$ already for $\mathsf{r}_0 = 20$. We remark

that for this set of parameters $\frac{\mathsf{cost(onVer)}}{\mathsf{cost(Ver)}} < 0.166$, i.e., our compiler produces an online verification procedure that is $6\times$ *faster* than the standard verification.

2.3 Generalization to Signatures with Properties

The results presented in the beginning of Sections 2 (i.e., the framework for efficient verification and our generic compiler) generalize easily to the case of signatures with different security notions, such as *strong* or *selective* unforgeability, or with advanced properties. This is of particular interest for LBS, where the versatility of well-established hardness assumptions has already given life to a variety of constructions under different security models and realizing advanced properties, including *homomorphic* [21], *threshold* [3], *constrained* [35] and *indexed attribute based* signatures [23]; and yet relying on an **Mv**-style verification (as introduced in the beginning of the section, and displayed in Figure 2).

Signatures with properties require more complex security definitions than plain existential unforgeability. Figure 4 provides a generic formalism to unify the description of the unforgeability experiments for signatures with properties. In a nutshell the common requires are:

1. If the signature guarantees *selective* unforgeability, the first step in the experiment is for $\mathcal{A}$ to declare the target messages for the forgery; in Figure 4 this is handled via the *val* variable. If unforgeability is adaptive, *val* is set to $val = \perp$.
2. A setup phase, where a probabilistic routine (denoted Setup in Figure 4) generates a set of secret values sval –handed over to the oracles– and other public auxiliary values pval, that include verification keys, delivered to the adversary.
3. A challenge phase, where the adversary is given access to some, possibly stateful, oracles (usually, at least an oracle that returns signatures by honest users), and has to output a message and a forged signature on it. We model this by defining two oracles:
   - $OK(\cdot;\mathsf{sval},\mathsf{st}_K)$: Returns signing/secret keys (of users or other entities that $\mathcal{A}$ may corrupt).
   - $O(\cdot;\mathsf{st})$: Encompasses all the other possible oracles (signing, opening for group signatures, etc.).
4. A check phase, where the experiment checks whether the signature output by $\mathcal{A}$ is valid and if $\mathcal{A}$ won the experiment. The former requires an execution of the verification algorithm; the latter includes a variety of additional checks to ensure the signature is actually a forgery (and

$\mathsf{Exp}^{\mathsf{generic-unf}}_{\mathcal{A},\Sigma}(\lambda)$

$val \leftarrow \mathcal{A}(1^\lambda)$

$(\mathsf{pval},\mathsf{sval}) \leftarrow \mathsf{Setup}(1^\lambda)$

$\mathsf{st}_K \leftarrow \emptyset,\ \mathsf{st} \leftarrow \emptyset$

$(\mu^*, \boldsymbol{\sigma}^*, \mathsf{aux}^*) \leftarrow \mathcal{A}^{OK(\,\cdot\,;\mathsf{sval},\mathsf{st}_K),\ O(\,\cdot\,;\mathsf{st})}(\mathsf{pval}, val)$

$b \leftarrow \mathsf{WinCond}(\mu^*, \boldsymbol{\sigma}^*, \mathsf{aux}^*, \mathsf{pval}, \mathsf{st}_k, \mathsf{st}, val)$

if $\mathsf{Ver}(\mathsf{pk}, \mu^*, \boldsymbol{\sigma}^*, \mathsf{aux}^*) = 1 \ \wedge\ b$

  return 1

else return 0.

Fig. 4: Generic description of the unforgeability under adaptive chosen message attacks experiment for signatures with properties.

is not trivially derivable from the adversary's view, e.g., because it was output by the signing oracle). We model this second check with the $\mathsf{WinCond}$ predicate. Clearly, the specification of $\mathsf{WinCond}$ depends on each primitive, and on the type of unforgeability: If selective, it checks that the queries and the forgery are consistent with the values *val* declared at the beginning of the game. If existential, it checks that μ^* was not queried to the signing oracle. If strong, it checks that the queries to the signing oracle are all distinct.

Adapting the syntax and security experiment of efficient verification to signatures with properties is rather straightforward. Similarly, our compiler of Section 2.1 can be easily adjusted to work on signatures with properties and with **Mv**-style verification, as we discuss momentarily. Regarding security, the core part of the proof of Theorem 1 is information-theoretical, and therefore it does not significantly change when considering signatures that are only selectively unforgeable, or strongly unforgeable. In the following we analyze the impact of our compiler on the efficiency of some schemes whose verification is structured as in Figure 2: the constrained LBS in [35], the (indexed) attribute-based LBS in [23], the homomorphic LBS in [16], the threshold LBS in [3], and the multivariate-based ring signature RingRainbow [30]. This list is by no means an exhaustive list. Indeed, in this work we decided to ignore lattice-based signatures with properties that are obtained using the Fiat-Shamir with abort technique from [26], despite the fact that wherever the result of Chen et al. [12] is applicable, our compiler is too. The reason is that signatures with properties that rely on such technique are many, and the efficiency gain computation is similar to the one performed in Section 2.2 .

Constrained Signatures (CS). CS allow a signer to sign a message only if either the message or the key satisfies certain preset constraints. The verification algorithm of the lattice-based instantiation of CS by Tsabary [35] includes an **Mv**-style check (where the matrix has n rows) and a norm check. Hence, our compiler applies directly to this scheme. Unforgeability requires that $n \geq \lambda$ and $q \leq 2^{\lambda}$, so for an average value $q \sim 2^{32}$, we can set $k = 9 \ll \lambda$ so that the advantage of $\mathcal{A}$ in Theorem 1 is $\frac{q_V+1}{q^k-q_V} < 1/2^{256}$. Remark that larger values of q (that could be required to have higher security guarantees) imply smaller values of k. Therefore, for this less conservative choice of parameters the efficiency gain is $\frac{\mathsf{cost}(\mathsf{onVer})}{\mathsf{cost}(\mathsf{Ver})} = \frac{k}{n} = \frac{8}{256} < 0.036$, i.e., the online verification requires about 3.6% of the computational cost of running the standard verification algorithm.

Indexed Attribute-based Signatures (iABS), and Homomorphic Signatures (HS). iABS allow a signer to generate a valid signature on a message only if the signer holds a set of attributes that satisfy some policy (represented by a circuit C). HS allow a signer to sign messages μ_i so that it is possible to publicly derive a valid signature for a message μ that corresponds to the output of a computation on the original messages, i.e., $\mu = \mathsf{C}(\mu_1, \ldots, \mu_r)$. According to the type of homomorphism supported by the scheme, the circuit C can encode only linear functions, polynomial functions, or any function of bounded multiplicative degree. In both iABS in [23] and HS in [16] the signature verification is composed by three steps:

1. Computation of the public matrix $\mathbf{M}$ from the circuit C (either the policy, or the homomorphic computation specified by the labelled program);
2. An '**Mv**'-style check;
3. A norm check on the signature.

The first step is critical because the public matrix $\mathbf{M}$ can be generated through a non-linear transformation, i.e., it might include multiplications of the public matrix by itself (or by a gadget

matrix). This would not allow to compute the first step online from the $\mathbf{z}_i$'s, but the verifier would have to use $\mathbf{M}$ and the $\mathbf{c}_i$'s instead, defying the purpose of our compiler. Hence, our compiler can be applied to these signatures in an efficient way only if either (1) C involves solely linear operations on the public matrix, or (2) C is fixed, or (3) C is known before running verification[9]. In these cases, we achieve efficient verification by letting offVer take as (additional) input C and compute $\mathbf{M}$ using the algorithm $\mathsf{PubEval}$ from [22]. The vectors $(\mathbf{Z}', \mathbf{v})$ used in the verification might (as in [16]) or might not (as in [23]) depend on the message. In the latter case the subroutine GetZV in onVer simply returns the input.
The impact of the compiler on the efficiency of HS was already analyzed in Section 2.2 . Regarding the iABS, the suggested value of the modulus q is such that $q \geq n^8$. The standard requirement $n \geq 2$ already implies that $1/q^k \leq 1/(2^8)^k = 1/256^k$. However, to guarantee the hardness of lattice-based problems usually n needs to be at least $n = 128$. In this case $q \geq 2^{56}$, hence already $k = 6$ guarantees that $\frac{q_V+1}{q^k-q_V} < 1/2^{305}$, thus the unforgeability of this iABS. As $n = O(d \log d)$ (where d is the depth of C) and the efficiency gain can be bounded as follows: $\frac{\mathsf{cost(onVer)}}{\mathsf{cost(Ver)}} \leq \frac{k}{O(d \log d)} = \frac{6}{O(d \log d)}$. From this inequality is clear that already for a circuit of depth 4 the online verification only requires 75% of the computation required by standard verification; the impact of our compiler increases for larger size of the circuit.

Threshold Signatures (TS). TS allow h out of ℓ parties to produce a signature on a message. Unforgeability is guaranteed for up to t colluding parties. Bendlin, Krehbiel, and Peikert [3] introduced a compiler that allows to distribute the signature generation step of the GPV08 signature, and convert it into a TS. The idea is to share the signing trapdoor among the parties using a h-out-of-ℓ secret sharing scheme. Signing requires at least h parties to come together to generate a signature satisfying a $\mathbf{Mv}$-type equation (where $\mathbf{M}$ is the public verification key). Verification is composed by the standard $\mathbf{Mv}$ equation and norm checks. Therefore, the thresholdizing compiler is composable with our compiler for efficient verification. As neither of them change the parameters of the underlying GPV08 scheme, the efficiency gain is the same (cf. Section 2.2).

RingRainbow [30]. RingRainbow is a ring signature scheme – i.e., a signature that allows a user to sign a message anonymously on behalf of a group – based on multivariate equations. This scheme is a hash-and-sign type of signature built as a modification of Rainbow. Verification requires to check whether the signature satisfies a multivariate quadratic system, and can be converted in a $\mathbf{Mv}$-style verification with the same technique used for Rainbow (cf. Section 2.2). Therefore, our compiler can be applied to RingRainbow as well. To evaluate the efficiency gain due to our compiler, we consider the efficient version of RingRainbow, (whose parameters can be found in Table 2 in [30]). For $\lambda = 128$ and a group of 5 users the authors set $\mathbb{F} = \mathbb{F}_{2^8}$ and $(v_1, o_1, o_2) = (36, 21, 22)$, which yield $m = 5 * (v_1 + o_1 + o_2) = 395$ and $n = 43$. Theorem 1 requires at least $\frac{q_V+1}{q^k-q_V} = 1/2^{256}$ for 128 bits of post-quantum security, which is ensured by $k \geq k_0 = 36$. Plugging these values in our amortized efficiency formula $\frac{k_0}{\mathsf{r}} + \frac{k_0}{n}$ (that is the formula derived from Definition 2 for signatures with $\mathbf{Mv}$-style verification) yields that the minimum number of repetitions r_0 to achieve meaningful amortized efficiency is $\mathsf{r}_0 = 580$, and the corresponding amortized efficiency factor is $\mathsf{e}_0 0.8992 > 36/580 + 36/43$. In this case, our compiler produces an online verification such that

[9] The construction of group signature in [23] has this iABS as building block, but it does not satisfy any of these conditions, as the verification circuit depends strongly on the signature. The authors did not find a straightforward way to modify this construction to have efficient verification without significantly impacting the signature length.

$\frac{\mathsf{cost(onVer)}}{\mathsf{cost(Ver)}} = \frac{k_0}{n} = \frac{36}{43} < 0.86$, in other words, our compiler produces an online verification that requires only 86% of the computation required by the standard verification.

3 Flexible Verification for Digital Signatures

In this section we introduce our framework for *flexible* verification of digital signatures. The basic idea is to design a flexible verification algorithm, $\mathsf{flexVer}$, that instead of returning a binary answer (accept/reject) as done by Ver, outputs a value α expressing the confidence on the validity of the signature. The confidence α is proportional to the amount of computation invested in the verification: the more verification steps performed the higher the confidence. We show how to realize secure flexible verification disregarding performance and efficiency implications. In the next section we address how to securely and simultaneously achieve efficient and flexible verification.
Differently from Le et al. [25], who define flexible signatures as a standalone primitive, we prefer to treat flexible verification as an add-on feature to enhance existing schemes. Compared to [25] our model is more general, yet slightly more involved. In particular, we let $\mathsf{flexVer}$ be a *stateful* algorithm; the state st is maintained by the verifier and should not be disclosed to the adversary. Albeit st can be seen as a secret, updatable, compact verification key, we remark that it is always possible to verify signatures using the signer's public key and the standard verification procedure. Moreover, any verifier can generate a st from the signer's pk, but st does not allow the verifier to forge signatures in the name of the signer.

3.1 Syntax

The core idea behind flexible verification is to derive from the original verification procedure of a signature scheme, Ver, an alternative algorithm, $\mathsf{flexVer}$, with certain properties. Concretely, this task boils down to identifying a sequence of $K_{\mathsf{flex}} + 1$ atomic instructions (that we informally call 'steps') that, if interrupted at any point, lets $\mathsf{flexVer}$ output a meaningful value about the correctness of the signature. In more detail, if one of the steps fails, $\mathsf{flexVer}$ returns $\alpha = \perp$ to reject the input signature. On the other hand, if none of the initial t steps fails, it is possible to derive a real value $\alpha_{\mathsf{flex}}(t) \in [0, 1]$ stating the probability that the given signature is valid. Intuitively, we want that the more steps $\mathsf{flexVer}$ executes successfully, the higher the value of $\alpha_{\mathsf{flex}}(t)$.

Definition 4 (Flexible Verification). *Let $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme. We say that Σ admits flexible verification $\Sigma^F = (\Sigma, \mathsf{flexVer})$ if there exist:*

- a non-negative integer $K_{\mathsf{flex}} \in \mathbb{Z}_{\geq 0}$,

- an efficiently computable confidence function $\alpha_{\mathsf{flex}} : \{0, \ldots, K_{\mathsf{flex}}\} \to [0, 1]$,

- a set of admissible states $\mathcal{S}$, that depends on K_{flex} and a public key pk, and includes the initial state $\mathsf{st} = \emptyset$ and any possible state output by some $\mathsf{flexVer}_i$,

- and an algorithm $\mathsf{flexVer}$ consisting of $K_{\mathsf{flex}}+1$ steps $\mathsf{flexVer}_0, \ldots, \mathsf{flexVer}_{K_{\mathsf{flex}}}$, such that $\mathsf{flexVer}$ has the following syntax:

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

1: $\alpha \leftarrow \perp$
2: **if** $t < 0$: **return** $\perp$
3: **if** $t > K_{\mathsf{flex}}$: **set** $t \leftarrow K_{\mathsf{flex}}$
4: **for** $j = 0, \ldots, t$
5: $(b, \mathsf{st}) \leftarrow \mathsf{flexVer}_j(\mathsf{st}, \mathsf{pk}, \mu, \boldsymbol{\sigma})$
6: **if** $(b = 0)$: **return** $\perp$
7: **else** $(b = 1)$: $\alpha \leftarrow \alpha_{\mathsf{flex}}(j)$
8: **return** α

$\mathsf{flexVer}(\mathsf{st}, \mathsf{pk}, \mu, \boldsymbol{\sigma}, t)$**:** *This a randomized algorithm takes in input a state* $\mathsf{st} \in \mathcal{S}$*, a public key* pk *(generated by* KeyGen*), a message* μ*, a signature* $\boldsymbol{\sigma}$ *and an interruption position* $t \in \mathbb{Z}_{\geq 0}$*. It outputs a value* $\alpha \in [0,1] \cup \perp$*. Concretely,* $\mathsf{flexVer}$ *is made of* $K_{\mathsf{flex}}+1$ *algorithms* $\mathsf{flexVer}_j(\mathsf{st}, \mathsf{pk}, \mu, \boldsymbol{\sigma})$ *that progressively update the state* st *and return a bit* b*. If* $b = 0$ *the flexible verification outputs* $\alpha = \perp$ *(*$\boldsymbol{\sigma}$ *is rejected); otherwise,* $\boldsymbol{\sigma}$ *is valid until step* j *and the flexible verification upgrades the confidence level to* $\alpha \leftarrow \alpha_{\mathsf{flex}}(j)$*.*

Flexible correctness essentially states that on valid signatures, i.e., signatures that would be accepted by the standard verification, $\mathsf{flexVer}$ accepts with confidence $\alpha_{\mathsf{flex}}(t)$.

Definition 5 (Flexible Correctness). *Let* Σ *be a signature scheme that admits flexible verification realized by the tuple* $(K_{\mathsf{flex}}, \alpha_{\mathsf{flex}}, \mathsf{flexVer}, \mathcal{S})$*. Then* $\Sigma^F = (\Sigma, \mathsf{flexVer})$ *satisfies flexible correctness if, for a given security parameter* λ*, for any key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\lambda)$*, for any admissible state* $\mathsf{st} \in \mathcal{S}$*, for any* $\mu \in \mathcal{M}$ *and signature* $\boldsymbol{\sigma}$ *such that* $\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma}) = 1$ *and for any value* $t \in \{0, \ldots, K_{\mathsf{flex}}\}$*, it holds that:*
$\Pr[\mathsf{flexVer}(\mathsf{st}, \mathsf{pk}, \mu, \boldsymbol{\sigma}, t) = \alpha_{\mathsf{flex}}(t)] = 1.$

We follow the approach of [25] and let the flexible verification algorithm output a value α that either rejects the signature ($\alpha = \perp$), or accepts it with certainty α in the real interval $[0, 1]$. We use the same interruption variable t as in [25] to model runtime interruptions of the algorithm execution.[10] We point out that while t is input to $\mathsf{flexVer}$, it is *not* given to each $\mathsf{flexVer}_j$; this syntax models the fact that the algorithm must work without knowing where it will stop, which is essential to capture arbitrary interruptions. In the case of an interruption, $t < K_{\mathsf{flex}}$, the algorithm would return a *premature* decision α on the validity of the signature: if $\alpha = \perp$ the signature is invalid (and this is certain, there are no false negatives); if $\alpha \in [0, 1]$ this means that according to the t steps performed, the signature appears to be valid.

Defining the confidence function $\alpha_{\mathsf{flex}}(\cdot)$ is fundamental to showing a realization of flexible verification Σ^F for a given signature scheme Σ. Any signature scheme Σ admits trivial flexible verification and confidence function. One way to see it is to consider Ver as a sequence of K_Σ instructions (e.g., computation or verification formulas –equations and inequalities); set $K_{\mathsf{flex}} = K_\Sigma$ and $\mathsf{flexVer}_j$ be the j-th step of Ver. So the confidence function can be defined as

$$\alpha_{\mathsf{flex}}(t) = \begin{cases} 0 \text{ for } t \in \{0, \ldots, K_\Sigma - 2\} \\ 1 \text{ for } t = K_\Sigma - 1 \end{cases}$$

A trivial confidence function entails un-interesting flexible verification: on correct signatures, $\mathsf{flexVer}$ returns 0 if interrupted at any step $t < K_\Sigma - 1$. While all signature schemes admit a trivial confidence function, not all support a non-trivial one, (e.g., a step function as shown in Figure 5).We remark that the confidence function $\alpha_{\mathsf{flex}}(\cdot)$ depends both on the scheme Σ and on how each $\mathsf{flexVer}_i$ is defined. In Section 3.3 we show non-trivial realizations for post-quantum secure signature schemes based either on multivariate quadratic equations or on lattices. It is easy to see that, without loss of generality, $\alpha_{\mathsf{flex}}(\cdot)$ has a non-decreasing trend, $\alpha_{\mathsf{flex}}(0) = 0$ and $\alpha_{\mathsf{flex}}(K_{\mathsf{flex}}) \leq 1$. Looking ahead, we will define $\alpha_{\mathsf{flex}}(t)$ to be $1 - \mathsf{bad}(t)$, where $\mathsf{bad}(t)$ represents the probability of accepting a forgery after t verification steps. While $\mathsf{bad}(t)$ decreases, $\alpha_{\mathsf{flex}}(t)$ increases and so does the confidence in the correctness of the signature.

[10] Our $\alpha_{\mathsf{flex}}(\cdot)$ is essentially the inverse of the function $\mathsf{iExtract}_\Sigma(\cdot)$ in [25].

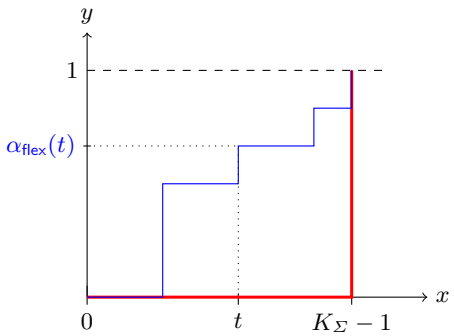


Fig. 5: Examples of confidence functions for flexible verification: the trivial case (thick, red); a non-trivial case (thin, blue).

Flexible Verification vs. Efficient (Offline/Online) Verification. At a first glance flexible and efficient verification seem to have the common goal of reducing the computational cost of a signature verification, however the way this objective is achieved in the two models is quite different. In flexible verification, the verifier (and thus each $\mathsf{flexVer}_i$) is unaware of when the computation will be interrupted, and *its execution is independent of* t. In contrast, in efficient verification the verifier (running offVer) determines the confidence level k prior to any actual verification (running onVer). Considering$k$ as *predictable* interruption value for onVer, shows that flexible and efficient verification are two sides of one and the same coin. The fundamental difference is that in the latter setting, the (online) verification is aware of the confidence level k (seen as interruption value), and *adapts its execution to* k.

Stateful vs. Stateless Verification We choose to define flexible signatures as stateful algorithms to keep the framework as general as possible. The same model could capture stateless flexible verification by considering the special case in which the state st is always $\varnothing$. In the latter case, there is no need for a verification oracle in the security experiment (Figure 6). In this sense, our approach is more general than the one by Le et al. [25] as we can potentially capture more schemes.

3.2 Security Model

Our security model revisits the notion of existential unforgeability under chosen message attack for flexible signatures proposed by Le et al. [25] with three twists.

First, our security game needs to take into account that $\mathsf{flexVer}$ maintains a possibly non trivial state. We handle this by allowing the adversary to interact with the flexible verification oracle $O\mathsf{flexVer}$ during the query phase (similarly to what we did in the efficient verification model of Section 2) and with the signing oracle $O\mathsf{Sign}$, in a concurrent manner.

Second, queries to $O\mathsf{flexVer}$ have the form $(\mu, \boldsymbol{\sigma}, t')$, where t' is the desired interruption value submitted by $\mathcal{A}$ (and chosen adaptively). To make the model generic and reduce the assumptions on $\mathcal{A}$, we include an interruption oracle $O\mathsf{Int}$ that takes as input t' and returns the actual value t to be used in $\mathsf{flexVer}$. We discuss in detail the role of $O\mathsf{Int}$ in a dedicated paragraph after stating our security notion. For the purpose of this work, we consider the strongest security model in which the interruption oracle returns the adversary's value, i.e., $t \leftarrow O\mathsf{Int}(t')$ with $t = t'$. This resembles side-channel attack settings, where $\mathcal{A}$ may try to freeze the execution of the verification.

Finally, a minor change: instead of a single bit, our experiment returns a pair (b, t^*). The bit $b \in \{0, 1\}$ flags the absence or the potential presence of a forgery, while $t^* \in \{0, \ldots, K_{\mathsf{flex}}\}$ reports

the interruption position used in the final flexible verification. Including t^* in the output of the experiment allows us to measure security in terms of *how far* is the probability of $\mathcal{A}$ wins the experiment, from the expected value $\alpha_{\mathsf{flex}}(t^*)$. Concretely, we consider a flexible verification to be *secure* if, for any PPT adversary, the probability that the security experiment returns $(1, t^*)$ is only negligibly higher than the expected probability of non detecting a forgery after t^* verification steps. For the purpose of this work, t^* is the interruption parameter chosen by $\mathcal{A}$ for its forgery.

Our security game and experiment for existential unforgeability under adaptive chosen message *with flexible verification* attack (flexEUF) are reported in Figure 6.

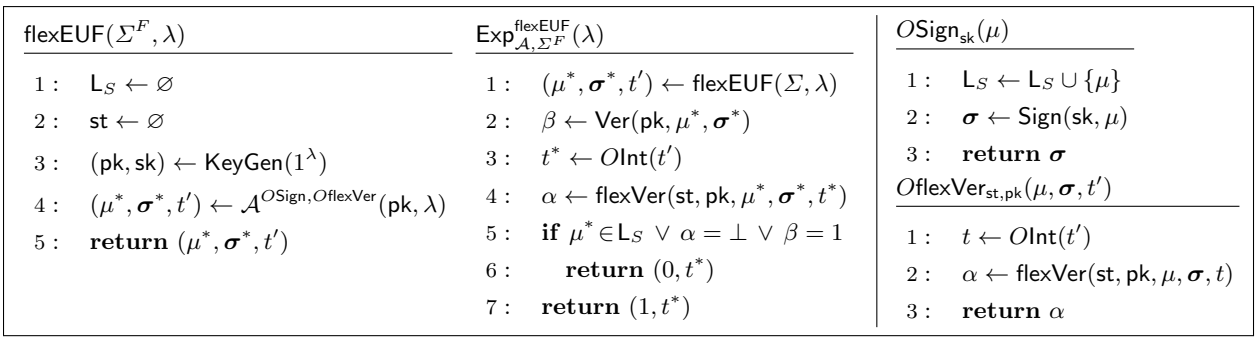


Fig. 6: Security model for flexible verification of signatures: existential unforgeability under adaptive chosen message and flexible verification attack (security game, experiment and oracles).

Definition 6 (Security of Flexible Signatures (flexEUF)). *Let Σ be a signature scheme that admits a non-trivial realization of flexible verification $\Sigma^F = (\Sigma, \mathsf{flexVer})$ with corresponding confidence function α_{flex}. For a given security parameter λ, Σ^F realizes a secure flexible verification for Σ if for all PPT adversaries $\mathcal{A}$ the success probability in the* flexEUF *experiment in Figure 6 is negligible, i.e.,:*

$$Adv^{\mathsf{flexEUF}}_{\mathcal{A}, \Sigma^F}(\lambda) = \Pr\left[\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A}, \Sigma^F}(\lambda) = (1, t^*)\right] - (1 - \alpha_{\mathsf{flex}}(t^*)) = \varepsilon \leq \varepsilon(\lambda).$$

Intuitively, Definition 6 states that an adversary has only negligible probability to make the flexible verification algorithm output a confidence value α^* higher than the expected one. Let $\mathsf{bad}(t)$ denote the probability of accepting a forgery after t verification steps. Then by setting $\alpha_{\mathsf{flex}}(t) = 1 - \mathsf{bad}(t)$, we get $Adv^{\mathsf{flexEUF}}_{\mathcal{A}, \Sigma^F}(\lambda) = \Pr\left[\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A}, \Sigma^F}(\lambda) = (1, t^*)\right] - \mathsf{bad}(t^*) \leq \varepsilon(\lambda)$.

In this work, we prove security in the strongest model where $t' = t$, i.e., $\mathcal{A}$ has the power to choose when to stop the verification. Since we put no restriction on the values t queried by $\mathcal{A}$ to OflexVer during the game, we will see that by running OflexVer on 'too few' steps, $\mathcal{A}$ may learn information about the internal state st.

Modelling Interruptions. In [25], unexpected interruptions are modeled via an interruption oracle $\mathsf{iOracle}(\lambda)$ that returns a value $t \in \{0, \ldots, K_{\mathsf{flex}}\}$ used by the flexible verification. However, it is not clear whether $\mathcal{A}$ may control iOracle. We overcome these ambiguities by letting $\mathcal{A}$ output t' at every flexible verification query. It is possible to relax and generalize our model by introducing an interruption oracle $O\mathsf{Int}$, programmed at the beginning of the game. At each verification query, $O\mathsf{Int}$ takes as input the adversary's suggestion for an interruption position t' and outputs the value t to be used by the flexible verification. In case $t = t'$, we are modelling side channel attacks, but we can also let t be independent of t'. A realistic definition of $O\mathsf{Int}$ is outside the scope of this work.

3.3 A Compiler for Flexible Mv-style Verification

We exhibit a compiler to obtain flexible verification for signature schemes with **Mv**-style verification (as described in Section 2.1). Specifically, our compiler for flexible verification builds on our previous compiler for efficient verification.

Two Insecure Solutions. Let us warm up with an instructive example. For schemes with **Mv**-style verification there is a natural way to divide the verification algorithm into atomic steps: the initial step, $\mathsf{flexVer}_0$, includes the consistency checks (e.g., the norm bound on the signature); while for $i = 1$ to $K_{\mathsf{flex}} = rows(\mathbf{M})$, $\mathsf{flexVer}_i$ performs the check $\mathbf{M}[i, *] \cdot \mathbf{v} = 0 \bmod q$, where $\mathbf{M}[i, *]$ denotes the i-th row of the matrix $\mathbf{M}$. Notably this solution does not require a secret state. Checking only $t < K_{\mathsf{flex}}$ constraints, however, may not be a secure choice. Indeed, an adversary can efficiently produce a message-signature pair that yields a $\mathbf{v}$ that meets the first $t << K_{\mathsf{flex}}$ linear constraints in the verification equation; this trivially breaks flexible existential unforgeability ($\mathsf{flexEUF}$). A naive way to bypass this issue is to randomize the order in which the verification equations are checked, e.g., let $\mathsf{flexVer}_0$ set a permutation of the rows of $\mathbf{M}$ and pass this on as part of st, the internal state of the flexible verification. In this case, setting $t^* = 1$, an adversary that can produce a valid forgery for the i^*-th row of $\mathbf{M}$ wins the $\mathsf{flexEUF}$ game with probability $1/K_{\mathsf{flex}}$ (which corresponds to the permutation picked by $\mathsf{flexVer}_0$ makes $\mathsf{flexVer}_1$ check the i^*-th verification equation $\mathbf{M}[i^*, *] \cdot \mathbf{v} \stackrel{?}{=} 0$). Notably, $1/K_{\mathsf{flex}}$ is not negligible, so neither of the two presented approaches is secure.

Our Compiler. Given a signature scheme Σ with **Mv**-style verification, we define a flexible efficient verification Σ^F for Σ as shown in Figure 7. In more detail, we need to specify the value K_{flex} (total

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

1 : $\mathsf{svk} \leftarrow \mathsf{offVer}(\mathsf{pk}, K_{\mathsf{flex}})$
2 : **parse** $\mathsf{svk} = (K_{\mathsf{flex}}, \mathbf{Z}, PK.\mathsf{aux})$
3 : $b \leftarrow \mathsf{Check}(PK.\mathsf{aux}, \mu, \boldsymbol{\sigma})$
4 : $\mathsf{st} \leftarrow \mathsf{GetZV}(\mathsf{svk}, \mu, \boldsymbol{\sigma})$
5 : **return** (b, st)

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

1 : $b \leftarrow 0$
2 : **parse** $\mathsf{st} = (\mathbf{Z}', \mathbf{v})$
3 : **if** $\mathbf{Z}'[i, *] \cdot \mathbf{v}[*, i] = 0 \mod q$
4 : **return** $(b \leftarrow 1, \mathsf{st})$
5 : **return** $(b \leftarrow 0, \mathsf{st})$

$\alpha_{\mathsf{flex}} : \{0, \ldots, K_\Sigma\} \to [0, 1], \qquad \alpha_{\mathsf{flex}}(t) = (1 - \frac{1}{q^t})$

Fig. 7: Our compiler for flexible verification of signatures with **Mv**-style verification. The algorithms $\mathsf{offVer}, \mathsf{Check}$ and GetZV are defined in Section 2.1, Figure 3. Here $K_{\mathsf{flex}} = rows(\mathbf{M})$.

verification steps), the set $\mathcal{S}$ (valid states), and the confidence function α_{flex}. The value K_{flex} sets the upper bound on the number of linear constraints the verifier wants to check, hence $K_{\mathsf{flex}} = rows(\mathbf{M})$, where $\mathbf{M}$ is the matrix employed in the original signature verification of Σ. The set $\mathcal{S}$ includes $\emptyset$ and any possible state output by some $\mathsf{flexVer}_i$, specifically $\mathcal{S} = \{0, 1\} \times \mathbb{Z}_q^{rows(\mathbf{Z}') \times cols(\mathbf{Z}')} \times \mathbb{Z}_q^{rows(\mathbf{v}) \times cols(\mathbf{v})} \times \{0, 1\}^\lambda \cup \varnothing$. We extract the confidence level from the probability of a flex forgery (as motivated by the proof of security given in Theorem 1). It is easy to see that the probability that an adversary creates a flex forgery for an interruption step t is at most $\frac{q^{n-t}-1}{q^n-1}$, this follows from the same reasoning as in the proof of Theorem 1 for efficient verification. Concretely, the bound is derived from the proof of Lemma 3, where we only consider $\Pr[\mathsf{bad}_1]$ as svk is refreshed with every

new efficient verification query, and so there is no useful cross-query leakage, and we replace the confidence level k of the efficient verification with the interruption parameter t. If the size of the underlying algebraic structure is $q = 2^{\mathsf{poly}(\lambda)}$ this probability is negligible already for $t = 1$. In other words, for signatures with **Mv**-style verification defined on *exponentially large* algebraic structures *efficient verification and flexible verification coincide*, trivially. The interesting case is $q = \mathsf{poly}(\lambda)$, as the adversary could create a flex forgery with non-negligible probability. We remark that in this section we are not targeting efficiency, and our instantiations of flexible verification refresh the svk produced by offVer at every verification query. This way, $\mathcal{A}$ cannot exploit the information possibly leaked by a flex forgery in future forgery attempts.

Theorem 2. *Let Σ be an existentially unforgeable signature scheme with* **Mv***-style verification (as of Fig. 2). Then the scheme Σ^F obtained via our compiler (in Figure 7) is a secure realization of flexible verification for Σ.*

Proof. Recall that an adversary $\mathcal{A}$ wins the security experiment in Definition 6 if it outputs a message-signature pair $(\mu^*, \boldsymbol{\sigma}^*)$ and an interruption t' such that: (1) $(\mu^*, \boldsymbol{\sigma}^*)$ is rejected by Ver, but accepted $\mathsf{flexVer}$ when it is interrupted at step $t^* \leftarrow O\mathsf{Int}(t')$; and (2) the flexible verification algorithm outputs a too high confidence level $\alpha_{\mathsf{flex}}(t^*)$. Following Definition 6, we can realize secure flexible verification by setting $\alpha_{\mathsf{flex}}(t) = 1 - \Pr\left[\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma}(\lambda) = (1, t)\right] + \varepsilon(\lambda)$ for all $t = 0, \ldots, K_\Sigma$. The core part of the proof is to estimate this probability.

Recall that our compiler for efficient **Mv**-style verification (in Figure 7) runs offVer at every verification query (line 1 in $\mathsf{flexVer}_0$). This means that every verification query is answered using a freshly generated svk. In particular, the final verification (line 4 in the $\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma^F}(\lambda)$ in Figure 8) checks $\mathcal{A}$'s output using independent randomness from the previous queries. So, whatever information the adversary may have collected from previous queries is useless to win the experiment. As a consequence, the probability that the adversary wins the game equals the probability that the adversary outputs a valid forgery *without querying* $O\mathsf{flexVer}$. The latter is precisely the probability of the event bad_1 defined in the proof of Theorem 1, where now we consider the matrix $\mathbf{C}$ to have t^* rows instead of k. Hence from Lemma 2 it follows that $\Pr\left[\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma}(\lambda) \leq (1, t^*)\right] = \frac{1}{q^{t^*}}$ and:

$$\begin{aligned} Adv^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma}(\lambda) &= \Pr\left[\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma}(\lambda) = (1, t^*)\right] - (1 - \alpha_{\mathsf{flex}}(t^*)) \\ &\leq \frac{1}{q^{t^*}} - \left(1 - \left(1 - \frac{1}{q^{t^*}}\right)\right) = 0 \ . \end{aligned}$$

□

3.4 Flexible and Efficient Verification

Given the two notions of efficient verification (Section 2) and flexible verification (Section 3), the question naturally rises whether it be possible to combine these frameworks and simultaneously realize flexible *and* efficient verification. Surprisingly, combining flexibility and efficiency in a naive way does not achieve high accuracy and unforgeability. This is essentially because efficiency demands reuse of svk, which makes the confidence function degrade with every new verification, while flexibility allows for premature verification outcomes that may leak a substantial amount of information about svk. To help the reader understand how the two definitions interact, in the following

we show an easy example of efficient compiler than can be proven secure in the flexible model too. The formal discussion of the model and of more generic compilers are deferred to Section 4.

Consider a compiler obtained from our compiler for flexible verification (depicted in Fig. 7) by skipping line 1 of $\mathsf{flexVer}_0$ (where it runs offVer to generate svk) in all verifications except for the first one. This means that svk is created at the first verification and reused in all subsequent runs of $\mathsf{flexVer}$. While this obviously improves the (amortized) efficiency of the compiler, it also impacts security, as every svk reuse leaks information about it. In fact, we can verify the unforgeability of this compiler only assuming q *exponential*: if the adversary can make at most q_V verification queries, the leakage is essentially $\frac{q_V}{q}$ (from Theorem 1 we already know that in this case verification can rely on just one $\mathbf{c}$), which is negligible. Thus the confidence function for this new compiler has to be defined as $\alpha_{\mathsf{flex}}(t) = 1 - \frac{q_V}{q}$. Theorem 3 summarizes the performance and security of the compiler.

Theorem 3 (informal). *For signatures with* $\mathbf{Mv}$*-style verification relying on algebraic structures of size* $q = 2^{\mathsf{poly}(\lambda)}$ *our compiler outlined for efficient flexible verification is unforgeable according to Definition 6 and achieves* $(2, 1/2 + 1/rows(\mathbf{M}))$*-concrete amortized efficiency as per Definition 2.*

Proof. The proof is based on the same argument used in the proof of Theorem 1. Borrowing the same notation from Theorem 1, the security experiment $\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma^F}(\lambda)$ is essentially the same as the experiment $\mathsf{Exp}^{\mathsf{cmvEUF}}_{\mathcal{A},\Sigma}(\lambda, k)$ with a few changes: (1) any appearance of onVer is replaced by $\mathsf{flexVer}$, (2) svk is *not* refreshed at every verification query, and (3) the number k of rows of $\mathbf{C}$ is replaced by the interruption parameters t output by $O\mathsf{Int}$ at each verification query. Since q is exponential in the security parameter, $\mathbf{C}$ can be a single row vector. In other words $K_\Sigma = 1$, thus we can set all interruption values t to 1. Analogously to the proof of Theorem 1, we can bound the adversary's success probability by the occurrence of $\{\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\}_{i=1,\ldots,q_V}$ events. Applying Lemma 2 yields:

$$\begin{aligned}\Pr\left[\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma}(\lambda) = (1, t^*)\right] &\leq \sum_{i=1}^{q_V} \Pr\left[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\right] + \Pr\left[\mathsf{bad}_{q_V+1} = 1 \mid \bigwedge_{j=1}^{q_V} \neg\mathsf{bad}_j\right] \\ &\leq \frac{q_V + 1}{q - q_V} \ .\end{aligned}$$

To conclude, for $q = 2^{\mathsf{poly}(\lambda)}$ and $q_V \leq \mathsf{poly}(\lambda)$ it holds that,

$$\begin{aligned}Adv^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma}(\lambda) &= \Pr\left[\mathsf{Exp}^{\mathsf{flexEUF}}_{\mathcal{A},\Sigma}(\lambda) = (1, t^*)\right] - (1 - \alpha_{\mathsf{flex}}(t^*)) \\ &\leq \frac{q_V + 1}{q - q_V} - \left(1 - \left(1 - \frac{q_V}{q}\right)\right) \\ &= \frac{1}{q - q_V} + \frac{q_V^2}{q(q - q_V)} = \varepsilon(\lambda) \ ,\end{aligned}$$

where the last equality follows from the hypotheses $q = 2^{\mathsf{poly}(\lambda)}$ and $q_V = \mathsf{poly}(\lambda)$.

In this setting the concrete efficiency of our compiler is achieved already with $\mathsf{r}_0 = 2$ repetitions: the generation of the svk and its use to verify a signature takes little more operations than a normal verification, but then the second verification only requires to compute the scalar product of two vectors in $\mathbb{Z}_q^n$. The corresponding amortized efficiency value is

$$\mathsf{e}_0 = \frac{\mathsf{cost}(\mathsf{offVer}(\mathsf{pk}, 1))}{\mathsf{r}_0 \cdot \mathsf{cost}(\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma}))} + \frac{\mathsf{r}_0 \cdot \mathsf{cost}(\mathsf{onVer}(\mathsf{svk}, 1, \mu, \boldsymbol{\sigma}))}{\mathsf{r}_0 \cdot \mathsf{cost}(\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma}))} = \frac{1}{2} + \frac{1}{rows(\mathbf{M})} \ .$$

□

In case $q = \mathsf{poly}(\lambda)$, the freshness argument on svk can no longer be used to bound the probability of finding a forgery: some non-negligible amount of information about the secret combinators $\mathbf{c}_i$ in svk is leaked at every flex-forgery submitted during verification queries. To formally handle these cases, in the next Section we introduce the notion of r-bounded randomness reuse.

4 Modeling Efficient & Flexible Signature Verification with r-Bounded Randomness Reuse

Given the two notions of efficient verification (Section 2) and flexible verification (Section 3), the question naturally rises whether it be possible to combine these frameworks and simultaneously realize flexible *and* efficient verification. Surprisingly, combining flexibility and efficiency in a naive way does not achieve high accuracy and unforgeability. This is essentially because efficiency demands reuse of svk, which makes the confidence function degrade with every new verification, while flexibility allows for premature verification outcomes that may leak a substantial amount of information about svk. In order to formally handle this situation and overcome the issue described above, we introduce the concept of **fl**exible and **ef**ficient verification with r-bounded randomness reuse (or r-flef in short). Similarly to Definition 4 (flexible signatures), this *sustainable* variant is defined for a given value k, that determines the maximum desired confidence level achievable by the verification. In addition to k, we need a second parameter, r, that determines the maximum number of times svk can be reused while guaranteeing unforgeability. For correctness and security, both k and r are input to the confidence function, which now is named α_{flef}.

Definition 7 (Efficient and Flexible Verification). *A signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ admits a (r, k)-efficient and flexible verification realization $\Sigma^{F+E} = (\Sigma, \mathsf{flefVer})$ if there exist*

- two positive integers: r (the number of reuses of the secret randomness) and k (the interruption step);

- an efficiently computable confidence function $\alpha_{\mathsf{flef}} : \{0, \ldots, k\} \times \{0, \ldots, \mathsf{r}\} \to [0, 1]$;

- a set of admissible sequences of states $\mathcal{S} = \{st^{(1)}, st^{(2)}, \ldots\}$ (each sequence $st^{(j)}$ contains $\mathsf{r}+1$ states st_i, i.e., $st^{(j)} = (\mathsf{st}_i)_{i=0}^{\mathsf{r}}$, $\mathsf{st}_0 = \varnothing$); and

- a flexible verification algorithm $\mathsf{flefVer}$ consisting of $k+1$ steps $\mathsf{flefVer}_0$, ..., $\mathsf{flefVer}_k$ with the same syntax as in Definition 4.

Definition 8 (r-Reuse k-Flexible Correctness). *Let Σ be a signature scheme that admits flexible and efficient verification realized by the tuple $(\mathsf{r}, k, \alpha_{\mathsf{flef}}, \mathsf{flefVer}, \mathcal{S})$. Then $\Sigma^{F+E} = (\Sigma, \mathsf{flefVer})$ satisfies (r, k)-correctness if, for a given security parameter λ, for any key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\lambda)$, for any one sequence of admissible states $st \leftarrow \mathcal{S}$, $st = (\mathsf{st}_i)_{i=0}^{\mathsf{r}}$, for any choice of r message-signature pairs $(\mu_i, \boldsymbol{\sigma}_i)_{i=1}^{\mathsf{r}}$ with $\mu_i \in \mathcal{M}$ and $\boldsymbol{\sigma}_i$ such that $\mathsf{Ver}(\mathsf{pk}, \mu_i, \boldsymbol{\sigma}_i) = 1$ and for any sequence of interruption values $(t_i)_{i=1}^{\mathsf{r}} \subseteq \{1, \ldots, k\}$, it holds that:*

$$\Pr[\mathsf{flefVer}(\mathsf{st}_i, \mathsf{pk}, \mu_i, \boldsymbol{\sigma}_i, t_i) = \alpha_{\mathsf{flef}}(t_i, i)] = 1 \quad \text{for all } i = 1, \ldots, \mathsf{r}.$$

Definition 9 (Concrete Amortized Efficiency). *A scheme $\Sigma^{F+E} = (\Sigma, \mathsf{flefVer})$ realizes $(\mathsf{r}_0, \mathsf{e}_0)$-concrete amortized efficiency if, for a given security parameter λ, for any key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\lambda)$, for any pair tuple of pairs $(\mu_i, \boldsymbol{\sigma}_i)$ with $\mu_i \in \mathcal{M}$ and $\boldsymbol{\sigma}_i$ such that $\mathsf{Ver}(\mathsf{pk}, \mu_i, \boldsymbol{\sigma}_i) = 1$, for*

any one sequence of admissible states $(st_i)_{i=0}^{\mathsf{r}} \subseteq \mathcal{S}$*, there exist a small, real constant* $0 < \mathsf{e}_0 < 1$*, and a non-negative integer* r_0 *such that for every* $\mathsf{r} \geq \mathsf{r}_0$ *the following holds true:*

$$\frac{\sum_{i=0}^{\mathsf{r}_0-1} \mathsf{cost}\big(\mathsf{flefVer}(\mathsf{st}_i, \mathsf{pk}, \mu_i, \boldsymbol{\sigma}_i, k)\big)}{\mathsf{r}_0 \cdot \mathsf{cost}\big(\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma})\big)} < \mathsf{e}_0 \tag{12}$$

4.1 Bounded Efficient and Flexible Security

Figure 8 collects a description of our security game and experiment for existential unforgeability under adaptive chosen message attack for signatures with *flexible and efficient verification* (r-flefEUF).

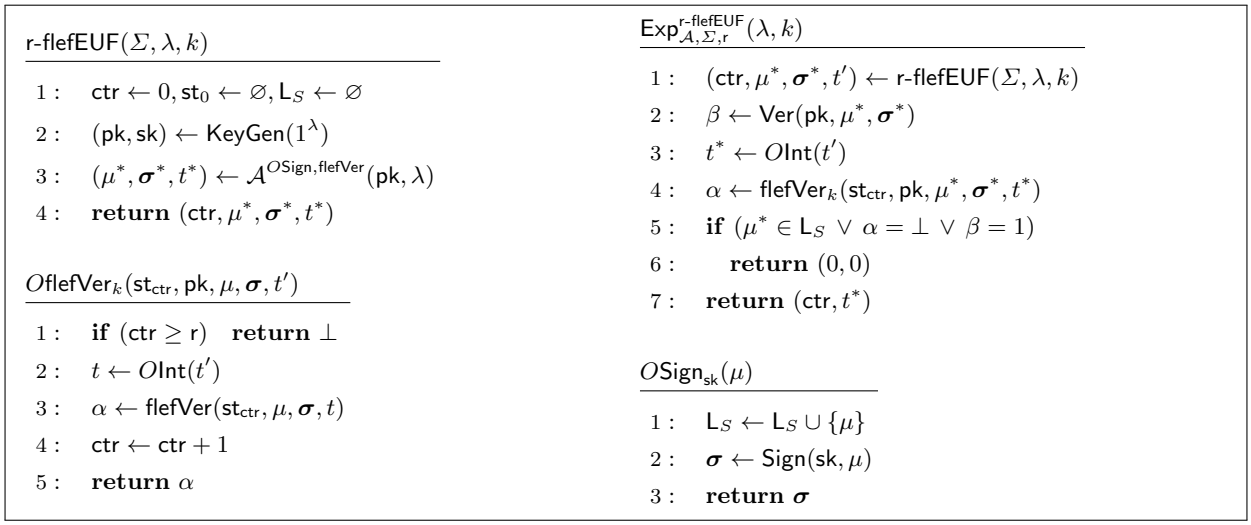


Fig. 8: Security model for existential unforgability under chosen message and flexible verification for signatures with stateful, (k, r)-efficient and flexible verification: queries security game, experiment and oracles.

Definition 10 (r-Bounded Flexible Security (r-flefEUF)). *Let* Σ *be a signature scheme that admits a non-trivial realization of* (r,k)*-efficient and flexible verification* Σ^{F+E}*. Then, for a given security parameter* λ*,* Σ^{F+E} *is existentially unforgeable under adaptive chosen message attack with flexible and efficient verification attack* (r-flefEUF) *if for all efficient PPT adversaries* $\mathcal{A}$ *the success probability in the* r-flefEUF *experiment is:*

$$\Pr\left[\begin{array}{c}\mathsf{Exp}^{\mathsf{r\text{-}flefEUF}}_{\mathcal{A},\Sigma,\mathsf{r}}(\lambda, k, \mathsf{r}) = (\mathsf{ctr}^*, t^*) \\ \wedge\ (\mathsf{ctr}^*, t^*) \neq (0, 0)\end{array}\right] \leq (1 - \alpha_{\mathsf{flef}}(t^*, \mathsf{ctr}^*)) + \varepsilon(\lambda).$$

4.2 A Compiler for r-flef Mv-style Verification

First of all we notice that our compiler for efficient verification described in Section 2.1 is trivially $\infty-$flefEUF (i.e., existentially unforgeable against an unbounded polynomial number of verification queries), by defining $\mathsf{flefVer} = (\mathsf{flefVer}_0, \mathsf{flefVer}_1)$ as shown in Figure 9. Recall that by assumption $\mathsf{st}_0 = \varnothing$ and that for q exponential in the security parameter Theorem 1 shows that we can set $k = 1$ and have unforgeability.

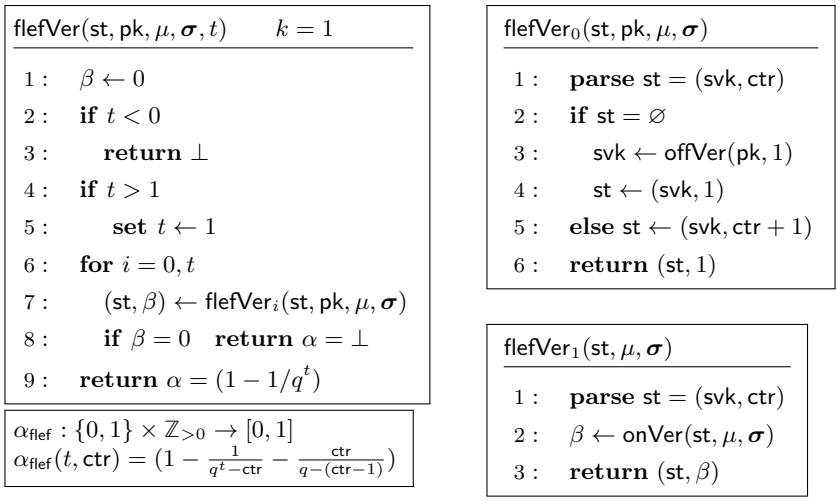


Fig. 9: The trivial two-step flefVer solution based on our compiler for efficient verification (Figure 3).

We now present a compiler for signatures with **Mv**-style verification and $q = \mathsf{poly}(\lambda)$ that realizes efficient bounded flexible verification. Our r-flef compiler builds on top of the two compilers presented in Section 2.1 and 3.3. Intuitively, the problem with flexible verification is that if interrupted after $t < k$ steps the process may erroneously accept an invalid signature with a non-negligible probability $\approx 1/q^t$. In Section 3.3 we mitigate this leakage of information between queries by refreshing the vectors in svk after every verification. This conservative approach clearly impacts efficiency. Here we want to prioritize efficiency at the cost of accuracy, and investigate how the confidence function degrades when the same set of vectors $\mathbf{z}_i$ is used to perform r flex-verifications.

Our r-flef compiler works essentially as the efficient verification compiler in Figure 7, except that the offVer algorithm (that generates a fresh svk) is run only *once every* r *verifications*. To further optimize the scheme, we replace the GetZV algorithm by k algorithms GetZV_i each of which is run by the corresponding $\mathsf{flefVer}_i$. The behavior of GetZV_i depends on the signature scheme and in what follows we define it for each of the three major classes we identified in this paper.

GPV08 [20]: the GetZV_i routine returns $\mathbf{z}'_i = \mathbf{z}_i = \mathbf{c}_i\mathbf{M}$, and $\mathbf{v}_i = [\boldsymbol{\sigma}|\mathcal{H}(\mu)]$.
MP12 [29]: the GetZV_i routine outputs $\mathbf{z}'_i = [\tilde{\mathbf{z}}_i \mid \mathbf{z}_i^0 + \sum_{j=1}^{\ell} \mu[j]\mathbf{z}_i^j|\mathbf{c}_i]$ and $\mathbf{v}_i = [\boldsymbol{\sigma}|\mathbf{u}]$.
Rainbow [14]: the GetZV_i routine outputs $\mathbf{z}'_i = \mathbf{z}_i = \mathbf{c}_i\mathbf{M}$, and $\mathbf{v}_i = [\tilde{\mathbf{s}}|\mathbf{s}|1|\mathbf{h}]$.

Finally, the confidence function $\alpha_{\mathsf{flef}}(\cdot,\cdot)$ is defined as:

$$\alpha_{\mathsf{flef}}(t,\mathsf{ctr}) = \begin{cases} \left(1 - \frac{1}{q^t - \mathsf{ctr}} - \frac{\mathsf{ctr}}{q-(\mathsf{ctr}-1)}\right) & \text{if } t > 0 \\ 0 & \text{if } t = 0 \end{cases} \tag{13}$$

$\mathbf{r}_0$-concrete amortized efficiency. The cost of $\mathsf{flefVer}_i$ varies depending on whether $i = 0$ or $i > 0$. When flefVer is run the first time (or with an empty state), the step $\mathsf{flefVer}_0$ generates the state. This includes computing (knm) multiplications, in the worst case. After that, every step $\mathsf{flefVer}_i$ computes at most $(n + m)$ multiplications (the first term represents the cost of running GetZV_i). Therefore,

$$\mathsf{cost}\big(\mathsf{flefVer}(\mathsf{st}_0, \mu_0, \boldsymbol{\sigma}_0, k)\big) = knm + k(n+m) \ .$$

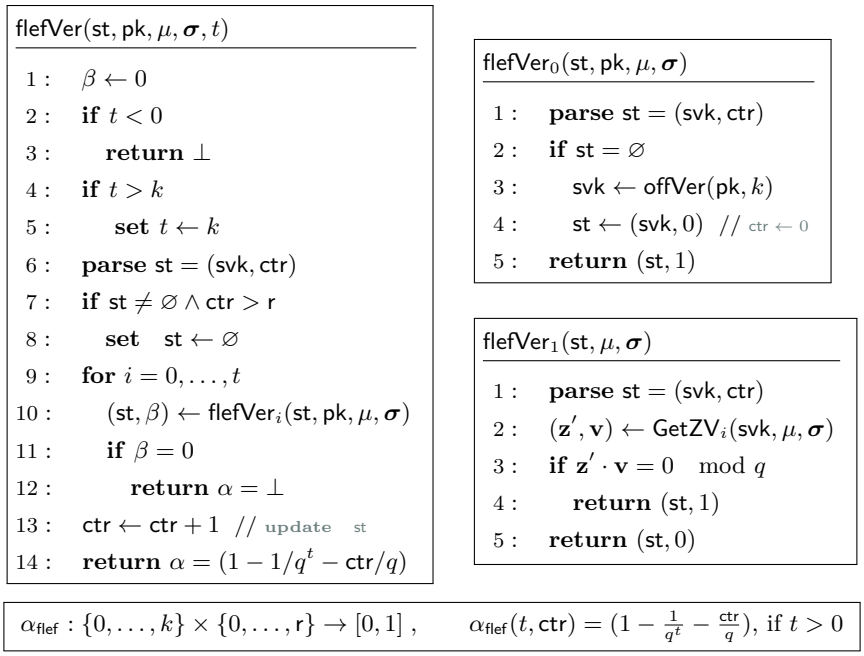


Fig. 10: Generic compiler to obtain efficient and flexible verification of signature schemes with **Mv**-style verification and q polynomial in the security parameter.

However, this is true only for the first execution of $\mathsf{flefVer}$, as when executing the verification $1 < \mathsf{r}_0 < \mathsf{r}$ times, the algorithm $\mathsf{flefVer}_0$ does not refresh the multipliers. Hence, for $i > 0$

$$\mathsf{cost}\big(\mathsf{flefVer}(\mathsf{st}_i, \mu_i, \boldsymbol{\sigma}_i, k)\big) = k(n+m) \ .$$

This yields $\sum_{i=0}^{\mathsf{r}_0-1} \mathsf{cost}\big(\mathsf{flefVer}(\mathsf{st}_i, \mathsf{pk}, \mu_i, \boldsymbol{\sigma}_i, k)\big) = knm + \mathsf{r}_0 k(n+m)$. The cost of a verification is dominated by $\mathsf{cost}\big(\mathsf{Ver}(\mathsf{pk}, \mu, \boldsymbol{\sigma})\big) = nm$ multiplications, in the worst case. Therefore, Equation (12) yields

$$knm + \mathsf{r}_0 k(n+m) < \mathsf{r}_0 nm \quad \Rightarrow \quad \mathsf{r}_0 > \frac{knm}{nm - k(n+m)} \ .$$

From the above formula we can derive a lower bound on values of r that yield efficiency (recall that by definition $\mathsf{r}_0 \leq \mathsf{r}$). A concrete security approach should lead to a meaningful upper bound on the value r that can be safely used in realistic applications. Intuitively, lower values of r yield higher accuracy (and unfogeability), higher ones guarantee better amortized efficiency.

4.3 On the Concrete Security of r-flef

The previous section seems to imply that flexibility and efficiency are in general two mutually exclusive properties of a provably secure signature scheme. This is true when considering asymptotic security definitions for lattice-based parameters that are (only) polynomial in λ. However, this does not imply that efficient flexible signatures are always insecure in practice. On the contrary, in this section we show that by limiting the amount of reuse of a secret verification key we get the best of both worlds. Concretely, we show that it is possible to simultaneously be both efficient and

(arguably) accurate even for those signatures with **Mv**-style verification that rely on algebraic structures of size q polynomial in the security parameter λ. The idea is to limit the amount of verification queries the adversary is allowed to make during the security game. This simulates the fact that, after r flef verifications that return a value $\alpha > 0$, the verifier re-sets svk, i.e., the secret randomness used for the efficient verification is refreshed. We keep track of this quantity via a counter ctr, as done in the compiler presented before.

In the following we formally prove the security of our compiler and conclude with remarks on actual values.

Theorem 4. *Let Σ be a signature with* **Mv***-style verification. Then our compiler for efficient and flexible verification provided in Figure 10 is* r*-bounded flexible secure for* $\mathsf{r} = k$*.*

Proof. The proof proceeds quite similarly to that of Theorem 3. Indeed, the r-$\mathsf{flefEUF}$ experiment in Figure 8 can be seen as the security experiment $\mathsf{Exp}^{\mathsf{cmvEUF}}_{\mathcal{A},\Sigma}(\lambda)$, where the number of queries q_V can be limited now to the number of times svk is reused, i.e., $q_V = \mathsf{ctr}^* \leq \mathsf{r}$, and where the number k of rows of $\mathbf{C}$ used to answer each query (and to check the validity of the forgery) is output by $O\mathsf{Int}$. Let bad_i be defined as in the proof of Theorem 1. Analogously to the proof of Theorem 1 we have the following

$$\begin{aligned}
\Pr\left[\mathsf{Exp}^{\mathsf{r\text{-}flefEUF}}_{\mathcal{A},\Sigma}(\lambda) = (\mathsf{ctr}^*, t^*), \mathsf{ctr}^* > 0\right] &\leq \sum_{i=1}^{\mathsf{ctr}^*} \Pr\left[\mathsf{bad}_i = 1 \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{bad}_j\right] + \Pr\left[\mathsf{bad}_{\mathsf{ctr}^*+1} = 1 \mid \bigwedge_{j=1}^{\mathsf{ctr}^*} \neg\mathsf{bad}_j\right] \\
&\leq \sum_{i=1}^{\mathsf{ctr}^*} \frac{1}{q^{t_i} - (i-1)} + \frac{1}{q^{t^*} - \mathsf{ctr}^*} \\
&\leq \frac{\mathsf{ctr}^*}{q - (\mathsf{ctr}^* - 1)} + \frac{1}{q^{t^*} - \mathsf{ctr}^*}
\end{aligned}$$

where t_i the interruption value used in the i-th verification, t^* is the interruption value used in the verification of the forgery, and the second inequality follows from Lemma 2. Therefore, assuming $t > 0$ (otherwise, by construction $\alpha_{\mathsf{flef}}(0) = 0$) we have:

$$\begin{aligned}
Adv^{\mathsf{r\text{-}flefEUF}}_{\mathcal{A},\Sigma}(\ \lambda\ , k) &= \Pr\left[\mathsf{Exp}^{\mathsf{r\text{-}flefEUF}}_{\mathcal{A},\Sigma}(\lambda) = (\mathsf{ctr}^*, t^*), \mathsf{ctr}^* > 0\right] - (1 - \alpha_{\mathsf{flef}}(t^*, \mathsf{ctr}^*)) \\
&\leq \frac{\mathsf{ctr}^*}{q - (\mathsf{ctr}^* - 1)} + \frac{1}{q^{t^*} - \mathsf{ctr}^*} - \left(1 - \left(1 - \frac{1}{q^{t^*} - \mathsf{ctr}^*} - \frac{\mathsf{ctr}^*}{q - (\mathsf{ctr}^* - 1)}\right)\right) \\
&= 0
\end{aligned}$$

□

Parameters for concrete security depend on the minimum accuracy that the verifier decides to tolerate, i.e., the minimum value of α_{flef} allowed. Such a value comes up in the worst-case scenario, in which the adversary only allows for $t = 1$ verification steps. From the description of α_{flef} given in Equation (13), the bound r on the number of "bad forgeries" (i.e., signatures whose verification with $\mathsf{flexVer}$ returns $\alpha > 0$ while with Ver returns 0) our compiler can tolerate. Hence, r is the maximum number of times the randomness svk can be reused while having a confidence level of at least α_{flef}. In Table 3 we report some values of r and α_{flef} (1) for different parameters of the main LBS considered thus far. We also provide the lower bound e on the amortized efficiency reached by our compiler, if run on r flef verifications. The third column of Table 3 states the following information: if we run our compiler on a lattice-based signature scheme with **Mv**-style verification and modulus $q = 2^{30}$ of polynomial size, $n = 256$ with maximum confidence level $k = 5$, we get:

– A r-flef scheme with minimal accuracy $99.99\% = 1 - 10^{-4}$; in other words flefVer is expected to accept an invalid signature with probability 0.01%, i.e., once every one million verifications.
– For this accuracy level, we can 'safely' run at most $\mathsf{r} = 1072$ flexible and efficient verifications using the same svk.
– For this accuracy level, running flefVer this number of repetitions leads to an efficiency ratio $\mathsf{e} = 0.00857 = 5/1072 + 1/256$, i.e., running our flexible and efficient verification requires less than 0.86% of the cost of running the standard signature verification about a 1000 times.

Overall this implies the verification yields at least 30 bits of security. This number might seem low when compared to asymptotic security. However, let us compare it with another real-world case, and consider a credit card with a 4 digit PIN code. To mitigate the chance for an attack to be successful the bank tolerates up to 3 unsuccessful trials. Thus, if the PIN is chosen at random, a credit card is secured at a level equal to $3/10000 > 2^{-12}$ bits of security.

Moreover, even for high accuracy levels (where we expectflefVer to mistakenly not recognize a forgery among one million signatures), we get astonishing efficiency gain, and can reuse the randomness a surprisingly large number of times.

5 Conclusions and Future Work

We presented a study on how to achieve efficiency and flexibility for signature verification. In addition to putting forth these notions and formal models for them, we presented two compilers that allow one to realize efficient (resp. flexible) verification for a wide class of existing lattice-based signatures. While our constructions show the feasibility of the desired properties, they also raise some natural follow up questions. For instance, is it possible to realize a compiler for LBS with $q \sim \mathsf{poly}(\lambda)$ that simultaneously provides efficient *and* flexible verification? We address this question in a positive way in Section 4, albeit in weaker security model. A solution with full fledged flexible security remains an interesting open problem. Another question is, is it possible to generalize our approach to other classes of digital signatures, e.g., code-based or LBS obtained through the Fiat-Shamir heuristic or from ideal lattices? Finally, it would be worth to explore more applications of

Table 3: Values for concrete accuracy and efficiency. The row r displays the maximum number of flexible and efficient verifications one car run while preserving an accurancy level α_{flef} higher than the desired $\min \alpha_{\mathsf{flef}}$ (displayed in the top row of the Table). The row e reports the corresponding concrete efficiency. In detail, we pick realistic parameters for lattice based signatures with polynomial modulus and set size of $q = 2^{30}$, $n = 256$, $k = 5$. We consider the worst case scenario where $t = 1$. Values are obtained by setting $\alpha_{\mathsf{flef}}(1, \mathsf{r}) = 1 - 1/(q - \mathsf{r}) - \mathsf{r}/(q - \mathsf{r} + 1)$ (cf. Equation (13)) to the desired $\min \alpha_{\mathsf{flef}}$ value, and estracting the corresponding r from the formula (for completenes we only display the closest largest power of 2, and below the actual value in parenthesis). For estimating e we use Definition 9 and observe that for our compiler the formula on the left hand side of the inequality translates to $\frac{knm + \sum_{j=1}^{\mathsf{r}} t_j m}{rnm} = \frac{k}{\mathsf{r}} + \sum_{j=1}^{\mathsf{r}} \frac{t_j m}{rnm}$. We derive e as a tight upperbound on $\frac{5}{\mathsf{r}} + \frac{1}{256}$, which corresponds to setting $n = 256$, $k = 5$ and $t_j = 1$, where r is the number of maximum repetitions allowed for the desired security level as dictated by the cells on that column of the Table.

$\min \alpha_{\mathsf{flef}}$ **desired**	$1 - 10^{-2}$ (99.00%)	$1 - 10^{-4}$ (99.99%)	$1 - 10^{-6}$ (> 99.99%)	$1 - 10^{-8}$ (> 99.99%)
$\max \mathsf{r}$ **supported**	2^{23} (10631100)	2^{16} (107360)	2^{10} (1072)	2^{3} (9)
e **achieved for** $(\min \alpha_{\mathsf{flef}}, \mathsf{r})$	0.00391	0.00398	0.00879	0.62891

flexible and efficient verification. On top of the already mentioned applications to real-time systems, another possible venue is parallel and distributed verification of digital signatures. Consider a public bulletin board that stores authenticated (signed) data. For security reasons, one may be tempted to use post quantum signature schemes such as LBS. However, the large sizes of the public keys and signatures and the slow speed of the verification are notorious bottlenecks to deploy them in such scenarios. Using our approach, a pool of parties –acting as verifiers– can be made in charge of running each a single verification check (i.e., $\mathsf{flexVer}$ includes only $\mathsf{flexVer}_0$ and $\mathsf{flexVer}_1$). In terms of security, although a single verifier may be wrong with non-negligible probability $1/q$, the probability that k honest verifiers are all wrong becomes negligible already for $k = 5$. Finally, we think that it would be interesting to explore the study of efficient and flexible verification also for more cryptographic primitives, such as commitments and zero-knowledge proofs.

Acknowledgments. This work was partly funded by: ELLIIT, the Swedish Foundation for Strategic Research grant RIT17-0035, SNSF, Swiss National Science Foundation project number 182452. Part of this work was made while C.B. was at IBM Research - Zurich (CH) and visiting the University of Aarhus (DK). This work has also received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00), CRYPTOEPIC (ref. EUR2019-103816), and SECURITAS (ref. RED2018-102321-T), and by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339).

References

1. M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the lwe, ntru schemes! In *Security and Cryptography for Networks SCN*, LNCS, 2018.
2. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *2013 ACM SIGSAC CCS.*, pages 863–874. ACM, 2013.
3. R. Bendlin, S. Krehbiel, and C. Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In *ACNS*, 2013.
4. D. J. Bernstein. A secure public-key signature system with sxtremely fast verification.
5. W. Beullens, A. Szepieniec, F. Vercauteren, and B. Preneel. Luov: Signature scheme proposal for nist pqc project. 2019.
6. D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC*, pages 1–16. Springer, 2011.
7. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, pages 514–532. Springer, 2001.
8. X. Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *PKC*, pages 499–517. Springer, 2010.
9. D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 523–552. Springer, 2010.
10. D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Advances in Cryptology – CRYPTO*, 2014.
11. D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology - CRYPTO*, pages 199–203. Springer, 1983.
12. Y. Chen, A. Lombardi, F. Ma, and W. Quach. Does fiat-shamir require a cryptographic hash function? In T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 334–363. Springer, 2021.

13. R. Del Pino, V. Lyubashevsky, G. Neven, and G. Seiler. Practical quantum-safe voting from lattices. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1565–1581, 2017.
14. J. Ding, M.-S. Chen, A. Petzoldt, D. Schmidt, and B.-Y. Yang. Rainbow. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. Accessed: 2020-09-21.
15. J. Ding and D. Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *Applied Cryptography and Network Security, ACNS. Proceedings*, volume 3531 of *Lecture Notes in Computer Science*, pages 164–175, 2005.
16. D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT*, 2016.
17. M. Fischlin. Progressive verification: The case of message authentication. In *International Conference on Cryptology in India*, pages 416–429. Springer, 2003.
18. N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Advances in Cryptology - EUROCRYPT. Proceedings*, pages 31–51, 2008.
19. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, 2010.
20. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *ACM Symposium on Theory of Computing. Proceedings*, pages 197–206, 2008.
21. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *ACM symposium on Theory of computing. Proceedings*, pages 469–477. ACM, 2015.
22. S. Gorbunov and D. Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology - ASIACRYPT. Proceedings*, Lecture Notes in Computer Science, 2015.
23. S. Katsumata and S. Yamada. Group signatures without NIZK: from lattices in the standard model. In *Advances in Cryptology - EUROCRYPT*, 2019.
24. L. Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International, 1979.
25. D. V. Le, M. Kelkar, and A. Kate. Flexible signatures: Making authentication suitable for real-time environments. In *European Symposium on Research in Computer Security, (ESORICS)*, pages 173–193. Springer, 2019.
26. V. Lyubashevsky. Lattice signatures without trapdoors. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, 2012.
27. L. P. Malasinghe, N. Ramzan, and K. Dahal. Remote patient monitoring: a comprehensive study. *Journal of Ambient Intelligence and Humanized Computing*, 10(1):57–76, 2019.
28. R. C. Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology - ASIACRYPT*, pages 218–238. Springer, 1989.
29. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.
30. M. S. E. Mohamed and A. Petzoldt. RingRainbow - An efficient multivariate ring signature scheme. In *Progress in Cryptology - AFRICACRYPT. Proceedings*, volume 10239 of *Lecture Notes in Computer Science*, pages 3–20, 2017.
31. T. Plantard, A. Sipasseuth, C. Dumondelle, and W. Susilo. Drs: diagonal dominant reduction for lattice-based signature. In *PQC Standardization Conference*, 2018.
32. A. Sipasseuth, T. Plantard, and W. Susilo. Using freivalds' algorithm to accelerate lattice-based signature verifications. In *International Conference on Information Security Practice and Experience*, pages 401–412. Springer, 2019.
33. A. R. Taleb and D. Vergnaud. Speeding-up verification of digital signatures. *Journal of Computer and System Sciences*, 2020.
34. S. Tayeb, M. Pirouz, G. Esguerra, K. Ghobadi, J. Huang, R. Hill, D. Lawson, S. Li, T. Zhan, J. Zhan, et al. Securing the positioning signals of autonomous vehicles. In *2017 IEEE International Conference on Big Data*, 2017.
35. R. Tsabary. An equivalence between attribute-based signatures and homomorphic signatures, and new constructions for both. In *Theory of Cryptography TCC*, 2017.