Fast verification and public key storage optimization for unstructured lattice-based signatures

Jean-Claude Bajard, Kazuhide Fukushima, Thomas Plantard, Arnaud Sipasseuth

Fast Verification and Public-Key storage optimisation for Unstructured Lattice based Signatures

Jean-Claude Bajard[1], Kazuhide Fukushima[2], Thomas Plantard[3] and Arnaud Sipasseuth[2]

[1]IMJ-Paris Rive Gauche, Sorbonne Université-CNRS-INRIA, Paris, France.
[2]Information Security Laboratory, KDDI Research, Inc, Fujimino, Saitama, Japan.
[3]Nokia Bell Labs, Murray Hill, New Jersey, United States.

Contributing authors: jean.bajard@inria.fr; ka-fukushima@kddi-research.jp; thomas.plantard@gmail.com; ar-sipasseuth@kddi-research.jp;

Abstract

A recent work of Sipasseuth, Plantard and Susilo proposed to accelerate lattice-based signature verifications and compress public key storage at the cost of a precomputation on a public key. This first approach, which focused on a restricted type of key, did not include most NIST candidates, or most lattice representations in general. In this work, we first present a way to improve even further both their verification speed and their public key compression capability by using a generator of numbers that better suit the method needs. We then also generalize their framework to apply to $\boldsymbol{q}$-ary lattice schemes as well as classical lattices using Hermite Normal Form, improving their security and applicable scope, thus exhibiting potential trade-offs to accelerate lattice-based signature verification in general and compression of the public key on the verifier side for unstructured lattices.

Keywords: Lattice-Based Cryptography, Digital Signature, Residue Number Systems, Probabilistic Verification

1 Introduction

The National Institute of Standards and Technology (NIST) Post-Quantum Cryptography Standardization process is still going on [37] to determine security standards for the next few decades, whether for encryption schemes or signature schemes. In the case of the NIST standardization, there is a security requirement of leakage resilience over 2^{64} signatures. Thus, the option to *be able to* reuse keys is implicitly chosen. If one opts in the option to reuse the same key to save key setup time, or to lower communication costs when several signatures are needed for a single user, then a solution is to not store the key but a footprint of the key, which still allows for the verification procedure to go through with a comparable efficiency, or maybe even better. Theoretically, replacing a deterministic verification procedure by a probabilistic one should pose no problem in the complexity hardness of the initial computational problems[13, 16] for which the security assumptions are most often heuristic. For lattice-based signature schemes, a lot of work has been done to promote leakage resilience even in the usage of an exponential amount of signatures[23].

Very recently, NIST reiterated their desire for new signature schemes that would not be based on structured lattices [34, 38]. It is important to notice that in their announcement, NIST asked for possible new signature schemes, making the distinction between structured and unstructured lattice based schemes, as quoted "NIST is primarily looking to diversify its signature portfolio, so signature schemes that are not based on *structured* lattices are of greatest interest. NIST would like submissions for signature schemes that have short signatures and fast verification", thus *unstructured* lattices should *still* be of interest. This is probably because the remaining lattice-based signature candidates [14, 18] heavily rely on cyclotomic rings structure which have yet to prove itself to be as secure as unstructured lattices. It also seems that the structure allows some computational problems to be easier to solve [11, 39]. However, ideal lattices allowed to gain a linear factor on both the size of the public key and the verification speed by the use of specialized algorithms on polynomial multiplication [28, 33, 41].

Other ways exist to add structure to a lattice therefore minimizing its representation; it is often based on adding polynomial structure. Consequently, the Hermite Normal Form (HNF) basis of a structured lattice can be compressed and therefore a more compact representation can be used. This more compact version allows to divide the size of the representation by up to a factor equivalent to the dimension (in the case of ideal lattice for example). Obviouly, this extra structure opens the door to new attack and therefore further weakness of the cryptographic scheme. By opposition, one will call an unstructured lattice a lattice which HNF basis is hard to distinguish from a random lattice basis picked uniformly from the set of all possible HNF given a fixed dimension and determinant, i.e respecting [25] distribution. Accelerating computations and compressing keys *without introducing serious security flaws* has always been a challenge, but some trade-offs which heuristically do not significantly reduce the security have been recently proposed for unstructured lattices.

A probabilistic verification trade-off technique was recently proposed [44] based on Freivalds' probabilistic verification algorithm [19]. Their work consisted of taking *any* classic signature scheme, and in that scheme replace *only* the verifier's means of verification computations, including a precomputation. Let the classic signature protocol be as follows:

1. Bob sends to Alice a public key $\mathbf{A}$
2. Alice sends to Bob a message $\mathbf{m}$
3. Bob signs $\mathbf{m}$ with his secret, outputting his signature $\mathbf{s}$
4. Alice verifies that $\mathbf{s}$ is valid for $\mathbf{A}, \mathbf{m}$

In the case where a same $\mathbf{A}$ is reused for multiple different messages $\mathbf{m}$, [44] propose that step 4 is computationally modified to accelerate all subsequent verifications at the cost of a precomputation. They based its security on the security of the initial scheme as it was globally left unchanged, and measured the security of the new verifier with the probability of recovering a "hidden" set of prime numbers by random tries. Their work effectively transforms a quadratic complexity (in the lattice dimension) for both the public key storage and the verification speed into a linear one *for the verifier*. However, [44] applied only to lattice-based signature schemes such as [40] that were not using a HNF as a public key. In particular, [44] required some lattice information from the signature of [40] to enable probabilistic verification. This information is missing from the signatures from all other lattice-based candidates, as such information is unnecessary when using a HNF for verification. Thus, their work did not apply to the lattice-based signature schemes that passed the NIST selection process, which exclusively used structured q-ary lattices that have HNF as public keys.

Our contributions

In this work, we propose two results to improve *unstructured* lattice-based signatures:

- We improve their precomputation for efficient probabilistic lattice-based signature verification. The result also allows for a stronger key compression and verification speed for similar security parameters.
- We generalize the work of [44] to apply to the case of public keys using HNF, which also includes the majority of lattice-based signatures in the NIST competition, although our work mainly focuses on unstructured lattices.

More precisely,

1. We improve precomputation for efficient probabilistic lattice-based signature verification. Our modification, just like [44], projects the computation over $\mathbb{Z}$ over multiple small rings $\mathbb{Z}/\omega\mathbb{Z}$ using the Chinese remainder theorem (CRT). The security of our modification follows the same security model as [44] and thus rely on the amount of possible combinations of such ω. We provide an efficient generator of such combinations of values ω, which can be considered fast as times are in the order of μs (see table 3). This generator is also lightweight: on $\lambda = 256$ bits security (i.e 2^λ possible moduli combinations for an attacker to guess from) example, at most 21 integers of 33-bits or less are required to be stored in the ω generator (see appendix table 7 for $(d, k) = (8, 10)$ i.e 8 moduli coprime to 10 first primes), which in returns offers the possibility to replace the storage of each public key by a storage of 10 vectors of n entries of size ω (regardless of the dimension n). For structureless public keys in lattice-based cryptosystems, this can save memory by a factor up to n. We show that after generating enough values ω to meet the required security requirements, those ω have a high-chance to be pairwise coprime. The generator is fairly simple and rely on simple tricks that were used in [17, 27].
2. We generalize the work of [44] to apply to the case of public keys using a Hermite Normal Form which includes the majority of lattice-based signatures in the NIST competition with an extra cost for the signatory and the signature size.

   In particular, we show our technique very well applies to lattice-based signature frameworks with proven leakage resilience such as [23] or [31]. We show linear complexity for our verifications, showing that for a dimension n a $O(n^2)$ complexity for structureless verification (often replaced by $O(n \log(n))$ by using polynomial structures) becomes $O(n)$, but at the cost of a precomputation which can be as costly as a matrix-vector product.

   This is also the case for the public key storage for the verifier, transforming a $O(n^2)$ storage into a $O(n)$ storage for unstructured q-ary lattices.

   Since lattice cryptosystems typically have $n > 500$ for minimal security requirements $\lambda = 128$ bits, this consequently exhibits several options to accelerate the verification speed, leaving the original security unaffected but at the cost of increasing the signature size.

   While the extra cost in signature length can appear as a drawback, the fact that our number generator is lightweight and that the key storage is heavily compressed open possibilities in the case of unstructured lattices: for low-memory devices that are only allowed extensive communications with a few other devices (home devices, internet of things, etc) or maybe even low-memory devices that still have to store several different public keys (one might imagine novel public key infrastructures with dedicated low-memory verifiers). The extra cost on the signature size is *at most linear* in the dimension in the case of q-ary lattices and *at best a single integer of no more than* 11 *bits for most of the highest existing security parameters* in the case of co-cyclic lattices, which is acceptable in some specific communication scenarios such as when a signature request does not occur immediately after a setup.

Note that our work *does not* make unstructured lattices as efficient as structured lattices: our work mostly helps the verifier, the signatory *still* has to manage the full size of his public key. We do not know how the communication architecture is going to look like in 40 years or more, and it is possible that in a future where structured lattices are severely weakened, such techniques might not be necessary. However, this should help researching ways to lessen the cost of deploying unstructured lattices in some cases if the need arises. If the NIST is indeed looking for “short signatures and fast verification” [38], we believe our framework gives an interesting trade-off on the signature size to accelerate verification speed.

Organization of the paper

In section 2, we reintroduce the concept of lattice-based signatures, and the work of [44]. In section 3, we present how to improve the work of [44] using a random number generator, which has very useful properties to enhance both its memory and computational cost for a given security. In section 4, we present how to generalize the work of [44] for both q-ary lattices and more generic lattices, where the aforementioned generator is also applied. In section 5, we conclude with open questions and propose future research directions.

2 Background

We will denote vectors in bold $\mathbf{v}$, matrices in capital bold $\mathbf{M}$, and integers as non-bold minuscules i. We denote v_i the i-th coefficient of a vector $\mathbf{v}$ and p_i the i-th element of a set P.

2.1 Lattices-based signatures

Definition 1 (Lattice).
We call lattice a discrete subgroup of $\mathbb{R}^n$ where n is a positive integer. We say a lattice is an integral lattice when it is a subgroup of $\mathbb{Z}^n$. A basis of the lattice is a basis as a $\mathbb{Z}-$module. If $\boldsymbol{A}$ is a matrix, we define $\mathcal{L}(\boldsymbol{A})$ the lattice generated by the rows of $\boldsymbol{A}$.

A specific subclass of lattices have been very popular in lattice-based cryptography: q-ary lattices.

Definition 2 (q-ary lattices). *We say a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ is a q-ary lattice if there exists $q < \det(\mathcal{L})$ such that for all $\boldsymbol{v} \in \mathbb{Z}^n$, we have $q\boldsymbol{v} \in \mathcal{L}$.*

The q-ary lattices in cryptography have determinants over $\mathbb{Z}$ that are typically a power of q and can be seen as subgroups of $\mathbb{Z}_q^n$, so we often represent their basis as matrices over $\mathbb{Z}_q$. A core argument for their popularity is their average to worst-case reduction [2].

Another subclass of lattices, representing approximately 85% of all lattices [36] and also benefiting from average-case to worst-case hardness reduction [20]: "co-cyclic lattices". We reuse the definitions used by [20, 36] to define those lattices:

Definition 3 (co-cyclic lattices). *We say a lattice is co-cyclic if $\mathbb{Z}^n/\mathcal{L}$ is cyclic.*

Because of their proportion in the set of all lattices, it is not unreasonable to assume co-cyclic lattices represent the average case of all lattices, structured or not. For simplicity we will only consider full-rank lattices, and in the case of co-cyclic lattices we will consider only the lattices that admit a specific shape as a basis.

Definition 4 (Hermite Normal Form). *We say a non-singular matrix $A \in \mathbb{Z}^{m\times n}$ is in Hermite Normal Form when:*

- *There exists $1 \leq i_1 < ... < i_h = m$ such that $a_{i,j} \neq 0 \implies (j < h) \wedge (i \geq i_j)$*
- *For all $k > j$, $0 \leq a_{i_j,k} < a_{i_j,j}$.*

Note that while a lattice has an infinity of basis, only one unique HNF basis exists. The HNF is a basis form that easily allows to test whether any $\mathbf{v} \in \mathbb{Z}^n$ verifies $v \in \mathcal{L}$ [9]. For simplification, we will consider in the case of co-cyclic lattices only full-rank lower-triangular basis where only the first column differ from the identity matrix $Id_n \in \mathbb{Z}^{n\times n}$. In fact, if a full-rank lattice admits such a basis, then it is likely to be co-cyclic. Note that q-ary lattices cannot be co-cyclic.

Definition 5 (γ-Guaranteed Distance Decoding ($\mathbf{GDD}_\gamma$)).
Given a basis $\boldsymbol{A}$ of a lattice $\mathcal{L}$, and a bound $\gamma > \lambda_1(\mathcal{L})/2$ where $\lambda_1(\mathcal{L})$ is the size of a shortest vector of $\mathcal{L}$, for any point $\boldsymbol{m}$ find a lattice vector $\boldsymbol{s} \in \mathcal{L}$ such that $\|\boldsymbol{s} - \boldsymbol{m}\| < \gamma$.

Most lattice-based signature schemes are based on the above problem, which is a stronger version of the Bounded Distance Decoding (**BDD**) problem[30, 32]. The public key is generally a basis $\mathbf{A}$ of the lattice, and the message is a target vector $\mathbf{m}$. In our above description in the introduction, Alice at step 4, i.e the verification step to check if the signature s is valid for key A and message m, verifies whether or not Bob solved the $\mathbf{GDD}_\gamma$ instance associated to lattice $\mathcal{L}(\mathbf{A})$ with target $\mathbf{m}$. The problem is known to be NP-hard for some values of γ and easy for some others, depending of the norm chosen [30, 32]. Note that in most applications, the target vector is not the message itself: the message $\mathbf{m}$ is a parameter of a hash function h which outputs target points $h(\mathbf{m}) \in \mathbb{Z}^n$, allowing $\mathbf{m}$ to be of any shape (thus, m might not be a vector but $h(m)$ is). The security assumptions often work on those arguments: the problem is easy to solve for the one who created $\mathbf{A}$ (i.e Bob, often by means of a trapdoor), but should not be for any other person: thus, only Bob should be able to produce valid signatures for Alice. In our work, we assume the initial cryptosystem is hard and present no obvious security flaws, and we will show on our paper that our modifications should not affect the initial security.

2.2 Freivalds' algorithm and application to lattice-based signatures

In [44], it was shown that the seminal work of Freivalds [19] could be adapted for verification of lattice-based signatures. Freivalds' seminal algorithm is often considered as the first algorithm to demonstrate the superiority of non-deterministic algorithms over deterministic ones in certain cases. It allowed to quickly verify if the result of the product of two real matrices was correct without computing the product itself. To summarize [44]'s adaptation of Freivalds' algorithm, for a lattice with public key $\mathbf{A} \in \mathbb{Z}^{r,c}$ (with $r = c$ in their case) and a signature $\mathbf{s}_{original} = (\mathbf{s}_A, \mathbf{s})$ for a target point $h(\mathbf{m})$ where h is a public hash function and m the message, the following equation used for verification

$$\mathbf{s}_A\mathbf{A} - \mathbf{s} = h(\mathbf{m}) \text{ with } s_A, s \in \mathbb{Z}^r$$

can be transformed, using a large integer Ω being the product of randomly picked primes ω (i.e $\Omega = \omega_1\omega_2...\omega_k$), to a much cheaper probabilistic verification involving

$$\begin{aligned} \mathbf{s}_A\mathbf{v}^{\top}_{(A,\omega_1)} - \mathbf{s}\mathbf{v}^{\top}_{\omega_1} &= h(\mathbf{m})\mathbf{v}^{\top}_{\omega_1} \mod \omega_1 \\ &\vdots \\ \mathbf{s}_A\mathbf{v}^{\top}_{(A,\omega_d)} - \mathbf{s}\mathbf{v}^{\top}_{\omega_d} &= h(\mathbf{m})\mathbf{v}^{\top}_{\omega_d} \mod \omega_d \end{aligned}$$

where $\mathbf{v} \in \mathbb{Z}^c_{\omega_i}$ are randomly picked and kept secret by Alice, and $\mathbf{v}^{\top}_{(A,\omega_i)} = \mathbf{A}\mathbf{v}^{\top}_{\omega_i}$. In such a modification, [44] proved that the probability that Alice can be cheated by an invalid signature, i.e the probability that Alice accepts a signature as correct where it would have been proven incorrect in the original scheme, is strictly inferior to Ω^{-1}, assuming Ω and $\mathbf{v}$ are *hidden and unrecoverable by external parties.*

Revealing Ω alone allows an external party to mount an attack where the underlying $\mathbf{GDD}_\gamma$ problem is changed. Instead of attacking the $\mathbf{GDD}_\gamma$ challenge on the original lattice $\mathcal{L}(\mathbf{A})$, an attack could be mounted on $\mathcal{L}(\mathbf{A}_\Omega)$ which can often prove to be a much easier challenge heuristically. The attack would not break Bob's original key $\mathbf{A}$, but would make it computationally feasible to forge wrong signatures $\mathbf{s}_{false}$ that Alice accepts but would otherwise be rejected in the original scheme.

On the other hand, it seems difficult for an attacker to forge invalid signatures while having no knowledge of Ω and v. It is even an open question if being able to give valid signatures allow to construct invalid signatures that would pass the test: however this scenario while interesting in theory has no practical interest as one might have to assume that the original cryptosystem is insecure in the first place (by assuming forgeability on the original scheme).

Thus, [44] put emphasis on setting and hiding Ω rather than $\mathbf{v}_{\omega_i}$ as the amount of available $\mathbf{v}_{\omega_i}$ is ω_i^c thus it is already naturally hard to guess if Ω is large, and the efficiency of this approach also depends on Ω. Naturally, they observe that for λ-bits of heuristic security, we must require $\log_2(\Omega) > \lambda$, but they show that for efficiency purposes Ω must be split into prime factors ω of a preferably much inferior size to Alice's machine word size. The rest of [44]'s work then focused on describing a methodology on how to choose the distinct prime factors $\omega_1\omega_2...\omega_d = \Omega$ that are both optimal for efficiency given a fixed machine word size and that kept the initial security of the original scheme prior to the modification intact. In particular, they state that $\log_2(\Omega) > \lambda$ is not enough. They take inspiration of [4] where random combinations of d numbers randomly selected among $2d$ pairwise coprime numbers gave $\binom{2d}{d}$ possible combinations as a layer of security against several forms of side-channel attacks. To this day, this approach is still secure. Thus, [44] follows [4]'s approach and state that given S_ω the set of all possible values ω the minimum bound of security is rather $\binom{|S_\omega|}{d} > 2^\lambda$, which also consequently forces $\log_2(\Omega) > \lambda$.

[44]'s modified signature protocol then proceeds as follows:

* *(new offline step)* Alice compute the couples $(\omega_i, \mathbf{v}_{\omega_i})$.
1 Bob sends to Alice a public key $\mathbf{A}$
* *(new offline step)* Alice compute $\mathbf{v}^{\top}_{(A,\omega_i)}$.
2 Alice sends to Bob a message $\mathbf{m}$
3 Bob signs $\mathbf{m}$ with his secret, outputting his signature $\mathbf{s}_{original}$
4 *(modified step)* Alice verifies that $\mathbf{s}_{original}$ is valid for $\mathbf{m}, \mathbf{A}, \mathbf{v}_{\omega_i}, \omega_i$ for all i.

3 Improving Sipasseuth-Plantard-Susilo's public key precomputation

We present how to improve [44] in the general case with a random number generator. First, explaining the properties we need from such a generator. Then we explain the basic idea behind this generator and how to construct it. We then explain how to customize it for specific integral forms if it is ever needed, provide experimental data on its efficiency and finish this part by providing justifications and tables for setting generator parameters to match security requirements. Note that for some readers, this whole section is possibly trivial: we do need, however, to exhibit the *exact values* given by this generator as they are necessary to set security parameters and optimize the gain in speed and memory on unstructured lattice signature schemes thus "just" giving intuitive parameters are not enough.

3.1 Constructing the number generator

Required properties

A simple observation which was left as an open question in [44] allow to improve their original work: we do not need prime numbers, coprimes are sufficient. In fact, we can as far as to say randomly picked ω do not need to be pairwise coprime: for every $x \in \mathbb{Z}$ projected over $\prod \mathbb{Z}_{\omega_i}$, we have an information redundancy for every pair of integers that are not coprime. In fact our approach depends on the size of the least common multiple of these ω_i i.e. $\text{lcm}(\{\omega_i\})$ close to Ω. When some ω_i are not coprime, we have a redundancy of the information that can be evaluated by $\log_2\left(\frac{\prod \omega_i}{\text{lcm}(\omega_i)}\right)$.

Note that picking pairwise coprime numbers also means picking prime numbers, so let us reclarify why [44] proposed to search for coprime numbers as an open question: for a matter of efficiency and combination security. In the current hardware today, machines have a certain bit-size. To increase the efficiency of the computations, [44] enforced every moduli picked to be within a certain size that would allow to reduce as much as possible the amount of modular operations needed to avoid overflows. Their work only considered primes as it was easy to count them for security purposes, and did not have a way to randomly generate pairwise coprime sets and let alone count them: opening the path to pairwise coprime sets, which *includes* sets of prime numbers, would allow them to keep a similar level of security with *smaller* sets or *smaller* moduli as the number of possible combinations *increases*. The approach of using large combination sets to enhance the security is not new: it has been used to increase the amount of possible number representations possible to thwart side-channels attacks [4, 10, 35].

In [4], random combinations of precomputed pairwise coprime numbers were used for efficiency and security against side-channel attacks. $2k$ pairwise coprime numbers were precomputed and the security was based on $\binom{2k}{k}$, where guessing the k numbers used for the computations among the $2k$ would break. In our case, we need k to be as low as possible to optimize computational costs, so maximizing the size of S_ω the set of all possible values ω to minimize the required value k to reach $\binom{|S_\omega|}{k} > 2^\lambda$ is still a valid approach. We then need to populate the set S_ω with as many efficient pairwise coprime numbers. Making S_ω as large as possible can be seen as obtaining a Residue Number System (RNS) base [1, 22] of maximal size given some constraints: the very recent work of [5] give us very efficient algorithms to do so. However, [44] already listed the *minimal* size has to reach S_ω in function of k to meet the minimal security requirements for each λ, and it seems a few millions are always necessary. In particular, for $k = 7$, S_ω must at least contain 610573333 pairwise coprime integers to achieve $\lambda = 192$-bits of security, and this independently of the bit size of the ω picked. Maybe this option is plausible for servers with large amounts of fast accessible memory, but for relatively small devices (maybe for IoT applications) this does not seem a feasible option.

Thus, generating numbers on the fly seem to be a more reasonable option. We wish to generate numbers to avoid any form of storage, from an original set as large as possible. We also know our approach depends of the lcm, thus we require a few things: generation must be reasonably fast, lightweight, and produce numbers such that their lcm is as close as possible to their product Ω.

It would also be desirable to have larger S_ω to lower the amount d of moduli required, smaller ω_i to reduce the number of potential overflows thus increasing efficiency per moduli, and both would allow for a lower space requirement for storing $\mathbf{v}_{(A,\omega_i)}, \mathbf{v}_{\omega_i}$ on top of accelerating further computations by lowering the arithmetic cost.

In this section, we propose such a generator with all of the aforementioned qualities, using fairly simple tricks.

Main Idea

The basic idea is fairly simple and was at least previously used in prime number generation [17, 27], although unlike previous works we do not plan to generate large prime numbers but rather multiple integers of similar and small size that are pairwise coprime. Let us take two random numbers. It was proven by Euler [26] (Theorem 332, p. 269 in the 4th edition for proof) that the chance of two integers being coprime is $\zeta(2)^{-1} = 6/\pi^2$ where ζ is the Riemann Zeta function[1]. Now, if the two of them are odd, the chances increases slightly. We can extend the idea to several numbers: while it is known that a set of d integers have 1 as a gcd with probability $\zeta(d)^{-1}$, it is a bit less known that the probability for such a set to consist exclusively of *pairwise* coprime numbers is:

$$P(d) = \prod_{p \text{ prime}} \frac{(d+p-1)(p-1)^{d-1}}{p^d}$$

which is the sum for each existing prime of the probability of being all coprime to p (i.e $(p-1)^d/p^d$) and the probability that only one of the d numbers is a multiple of p (i.e $d(p-1)^{d-1}/p^d$).

[1]this is known as the Basel's problem

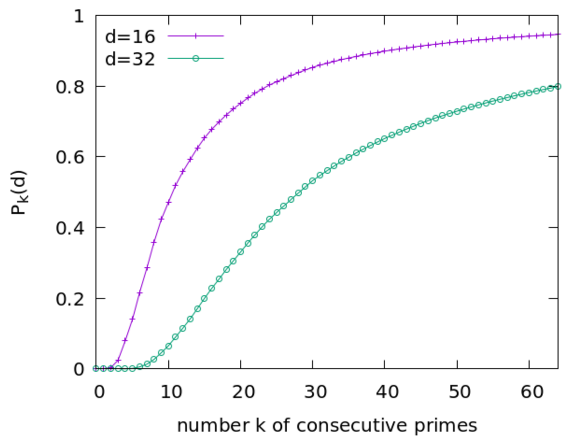

Fig. 1 Experimental approximation of $P_k(d)$ for $k \in [0, 64]$

We know that giving assumptions about the integers being not purely random, but being coprime to the first successive primes, increase the odds of the resulting set being composed of pairwise coprimes: in a different context and objective, this observation helped generating large primes [17, 27]. Let us denote $P_k(d) = \prod_{i \leq k} \frac{(p_i-1)^d}{p_i^d} \prod_{i>k} \frac{(d+p_i-1)(p_i-1)^{d-1}}{p_i^d}$ the probability of d integers being pairwise coprime given that all d integers are coprime with the first successive k primes. We provide a simple experiment to measure the probability of success to obtain d pairwise coprimes by randomly sampling d numbers coprime with the first k prime numbers millions of times for each pair (k, d), and show the results in figure 1. As we need to generate as little as possible moduli w_i for as little as possible d *while* maintaining some security, we do not need to study $P_k(d)$'s asymptotical case, but computations on relevant parameters would at least show that our approach is valid and help to properly set parameters. Furthermore, in our approach, $P_k(d)$ corresponds to a best case when the $\text{lcm}(omega_i) = \Omega$.

Those results give us a method to sample such integers: we exploit the set $\mathcal{M}_k = \{p_1, ..., p_k\}$ composed of the first consecutive k primes and use the CRT to build the actual numbers that would be part of a set of size d of probably pairwise coprime. Let us note $x_{\mathcal{M}_k} = < x_1, ..., x_k >$ the residues of x for each moduli of $\mathcal{M}_k$. If for all k, $x_i \neq 0$, then we know the CRT reconstruction would give x lower than $M_k = p_1...p_k$ such that it is coprime to the first k primes. To simplify algorithms descriptions, introduce the **Rand** function, which takes into entry a set and outputs a random element of the set. This gives us the sampling algorithm basic coprime algorithm (**BaseCop**) (Algorithm 1).

Algorithm 1 Basic Coprime Algorithm

```
Input: $\mathcal{M}_k = \{p_1, ..., p_k\}$
Output: an integer $x < M_k$, with $\gcd(x, M_k) = 1$
1: $< x >_{\mathcal{M}} \leftarrow < 0 >_{\mathcal{M}}$        ▷ (Allocate memory)
2: for $i \in [1, k]$ do
3:     $x_i \leftarrow$ Rand$([1, p_i - 1])$        ▷ $(p_i \nmid x)$
4: end for
5: $x \leftarrow$ CRT$(< x >_{\mathcal{M}_k})$
6: return $x$
```

In particular, let x, y be outputs of **BaseCop**$(\mathcal{M}_k)$, then $\gcd(x, y) = 1$ or $\gcd(x, y) > p_k$. Experimentally, $\gcd(x, y) = 1$ becomes more common as k grows. Note that by removing numbers divisible by small primes, we are thinning the amount of numbers to sample from. We previously roughly measured $P_k(d)$, but if we aim for guaranteed pairwise coprimality instead of probable coprimality, we might not want to discard d integers and generate d new samples. In that regard, we will now present our tests on how successful we would be to discard a "bad" integer and reinsert a "good" one as we generate integers successively. Our testing method is the following:

- Generation: We fix k and sample D integers by successive calls to **BaseCop**(B).
- Verification: We go through the samples in incremental order. If it is coprime with the previously saved samples, we save it, otherwise we discard it.

The first generated is always counted as coprime and saved, and Basel's theorem states the second one has roughly 61% chances to be coprime to the first one *if sampled without our method*. However, this value is an asymptotical value. Experimentally, we obtain a discard rate above 79.2% when simply using random bounded numbers. **BaseCop**(B) yields better results: for $k = 10$ and 100 samples, we only discard less than 20% of the samples. More detailed experimental results can be found in table 1 below.

Table 1 Average discards after 10^4 calls to **TestGen**(**BaseCop**$(\mathcal{M}_k), D)$

k	5	6	7	8	9	10
$D = 100$	36.0	28.4	25.1	22.3	20.01	18.3
$D = 200$	93.5	72.1	65.0	59.3	54.7	51.0
$D = 300$	161.2	121.1	109.6	101.3	94.2	88.6

Remark: to increase the chances of generating successive coprimes, a first improvement is to allow integers to be divisible by any prime possible in the basis composing $\mathcal{M}_k$, and every time we sample an integer divisible by primes composing $\mathcal{M}_k$, we start excluding integers that are multiples of said primes. The improvement was however not significant in our experiments, in particular it has almost no impact on later security parameters while complexifying the algorithms, thus we will not document the numbers in this paper.

Coprimes of Pseudo-Mersenne, Montgomery-friendly and customization

Here we present a specialization of our previous algorithm to generate integers of the shape $K - c$. The main application for this specialization is that we could create pseudo-Mersenne numbers [12] by setting K as being a large power of 2, or just take any K to control the bit size of the output and output integers of the shape $K - c$.

Instead of sampling c such that residues c_i are non-zero, this generalization samples c such that $K - c_i$ have non-zero residues. Which leads to algorithm 2 we denote Mersenne coprime algorithm (**MersCop**). We also repeat the previous test procedure by replacing **BaseCop** by **MersCop** our experimental results show similar coprimality probability and rejection rates. Detailed results can be found below in table 2 : we can see the difference with 1 is negligible.

Algorithm 2 is slightly slower than 1, as 2 it involves the same operations plus extra operations to deal with $K \neq 0$. However, this small extra

Algorithm 2 Pseudo-Mersenne Coprime Algorithm

```
Input: $K \in \mathbb{N}$, $\mathcal{M}_k$
Output: $c \in ]0, \prod p_i]$ such that $\gcd(K - c, p_i) = 1$
1: for $i \in [1, k]$ do
2:     $c_i \leftarrow \mathbf{Rand}(\{c_i \in [0, p_i - 1], K \not\equiv c_i\ [p_i]\})$
3: end for
4: $c \leftarrow \mathbf{CRT}(< c >_{\mathcal{M}_k})$
5: return $c$
```

Table 2 Average discards after 10^4 calls to $\mathbf{TestGen}(\mathbf{MersCop}(2^n, \mathcal{M}_k), 100)$

k	5	6	7	8	9	10
$n = 32$	36.50	28.52	25.12	22.29	20.01	-
$n = 64$	36.59	28.41	25.14	22.38	20.05	18.40

cost during precomputation allows to use Pseudo-Mersenne which offers significant gain during modular reductions. A simple way to make the **Rand** function in algorithm 2 almost as simple as in algorithm 1 is to use $\mathbf{Rand}([0, p_i - 2])$ and then add 1 if $c_i \geq K \mod p_i$ (that ensures the same probability for all values in $[0, p_i - 1] \backslash \{K \bmod p_i\}$). In the C language the output of "x >= y" is 0 or 1, thus we can directly add the result of the comparison and keep the same distribution. In practice, we want our random component to be of a certain bit size or to have specific arithmetic properties. Further customization/optimization for the algorithms can be done for all forms of first order equations: for example, if K_1, K_2, e are constants (e can be as small as c), and we want to generate *probably coprime* numbers of the form $x = K_1(K_2 - c) - e$, we just need to make sure $c \neq K_2 - e \times K_1^{-1}$ modulo the p_i. This includes Montgomery-friendly numbers [3]), but for our purposes simpler integer shapes are sufficient. Experimental results on different integer shapes also show the same results as in algorithm 1 for the basic case, thus we are not listing the results there.

3.2 Experimental efficiency and application to Sipasseuth et al's work

Experimental results

We conducted experiments on a machine using an Intel(R) Xeon(R) E-2246G CPU @ 3.60GHz processor with a simple C program using GMP and compiling with the optimization option -O3. Unsurprisingly our generator take around $1\mu s$ for $(k, d) = (5, 6)$ and up to $7\mu s$ for $(k, d) = (15, 16)$. Timings can be found below in table 3.

Table 3 Average time in μs for extracting d samples from $\mathcal{M}_k^*$

d \ k	5	6	7	8	9	10	11	12	13
6	1.25	1.31	1.38	1.44	1.49	1.57	1.64	1.70	1.77
9	1.44	1.53	1.62	1.76	1.81	1.94	2.02	2.12	2.21
12	1.62	1.76	1.87	2.00	2.12	2.31	2.41	2.54	2.68
15	1.82	1.97	2.13	2.28	2.43	2.66	2.80	2.97	3.13

In comparison, the fastest structured lattice-based signature schemes can verify up to 27933 signatures per second [18] so approximately $36\mu s$ per verification. Although their testing processor (i5-8259U clocked at 2.3 GHz with "TurboBoost" disabled) runs at a lower frequency as reported in 2022, it is still not in the same order as our proposition. Furthermore, their code includes optimized assembly code, while we use purely unoptimized C code. We also conducted experiments with checking pairwise coprimality at each generation and retry until we get a set of pairwise coprime: the generation time is multiplied by between 2 or 3 in average. The point of those timings is purely informative: as far as we know there is no pertinent comparison we can make, but we can at least exhibit its low cost compared to a typical public key lattice-based signature scheme setup (see [37]). Since the generation time is already low and can be achieved offline (before receiving the public key), check-and-retry is still a reasonable approach.

More importantly, we measured the average redundancy size, its median size, its worst-case size and its lowest size when it is non-zero. Unsurprisingly most sample groups end up being pairwise coprime, the median value for redundancy is automatically zero: thus, we make the minimum redundancy we measure is the *minimal non-zero redundancy* measured. Our experimental results show that the redundancy is usually low with our generator: more often than not the redundancy is zero, and when it is not zero it is often less than 10% of the total size and and after 2^{20} tests the redundancy does not peak at more than a third of the total size, while using random numbers provide high redundancy. Tables can be found below, namely table 4 with randomly picking numbers in table 5.

Table 4 bit size redundancy when using **BaseCop**($\mathcal{M}_k$) $\log_2\left(\frac{\prod \omega_i}{\mathrm{lcm}(\omega_i)}\right)$ on 2^{20} tests

d	k	Average	Median	min $\neq 0$	max
6	5	1.756275	0	4	32
6	6	1.344412	0	5	33
6	7	1.110809	0	5	31
6	8	0.913920	0	5	29
9	5	3.993797	0	4	49
9	6	3.083662	0	5	45
9	7	2.560968	0	5	44
9	8	2.140330	0	5	40
12	5	6.969599	5	4	54
12	6	5.417657	5	5	50
12	7	4.531733	5	5	51
12	8	3.815385	0	5	55
15	5	10.589952	10	4	65
15	6	8.284572	7	5	63
15	7	6.971303	6	5	66
15	8	5.905113	5	5	59

Table 5 bit size redundancy when using random numbers mod M_k
$\log_2\left(\frac{\prod \omega_i}{\mathrm{lcm}(\omega_i)}\right)$ on 2^{20} tests

d	k	Average	Median	min $\neq 0$	max
6	5	9.328667	9	1	55
6	6	9.285503	9	2	69
6	7	9.288168	9	2	89
6	8	9.286525	9	2	111
9	5	18.060690	17	1	86
9	6	17.975490	17	2	110
9	7	17.961327	17	2	139
9	8	17.964048	17	2	176
12	5	28.312817	27	1	117
12	6	28.151557	27	2	150
12	7	28.125704	27	2	191
12	8	28.135508	27	2	242
15	5	39.670601	39	1	148
15	6	39.435517	39	3	189
15	7	39.389500	39	4	243
15	8	39.394647	39	2	308

While the end result of the generator is more efficient than the previous precomputations of [44], a cost comparison is seemingly hard as [44] did not list any particular method to uniformly pick random primes within a fixed set. However we believe the generator we exhibited, which costs a CRT reconstruction from small size moduli with operations that fit within a machine word-size, should be hopefully more efficient in implementation space and speed than any possibly known prime picking technique. Figure 2 presents the result of our experiment made to compare different methods by counting how many clock cycles it would take to generate random pairwise coprime integers, sampling integers one by one and rejecting every integer that is not pairwise coprime with the others. To do so, we calculate the product of every pairwise coprime, memorize the product and compute its gcd with every newly generated number we need to add. If the number is coprime with the product, we update the product and continue. To generate primes, we used the GMP function `mpz_probab_prime_p` to discard non-prime numbers and to generate random numbers we just used the `rand()` function in C (we do not believe another random number generator such as `mpz_urandomb` would yield significant differences in our experimentations). All experimentations are done on a machine using an Intel(R) Xeon(R) E-2246G CPU @ 3.60GHz processor with a simple C program using GMP and compiling with the optimization option -O3.

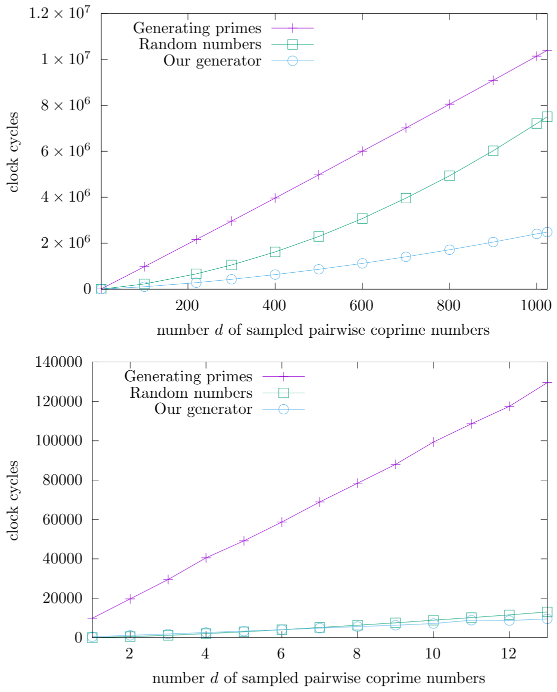

Fig. 2 Cycles to create d 28-bits pairwise coprime integers

One can remark that the prime number generation grows quasi linearly because we do not check the gcd but rather the equality with other previously sampled numbers, while the k-th number generated need to check if it is coprime with the $k-1$ previously stored integers, and gets rejected if not. Thus, prime numbers *might become* more efficient for (an unknown) large amount d of pairwise coprime numbers: no gcd computations are needed to check the results, unlike other methods. However, since we do not usually need more than $d = 12$ numbers for the largest security parameters $\lambda = 256$, our method proves more effective, especially since our tests up to $d = 1024$ shows our approach remains more effective.

Practical parameters and application to [44]

As we stressed out earlier, one of the main security requirement was to have S_ω as large as possible. Our new method indeed gives a large set S_ω but they are no longer pairwise guaranteed coprime. Of course, we can generate more integers to compensate which is what we measured in the previous subsection, but as the generation of a pairwise coprime set is fast itself (the amount of resampling was proven to be low) the option to use strictly pairwise coprime sets can be a security choice. Thus, enforcing pairwise coprime sets only give a fraction of $\binom{S_\omega}{k}$ as possibilities. Typically, given d integers sampled from a RNS basis $\mathcal{M}_k$ (hence couples (d,k)), we estimate the bit security parameter λ by the amount of valid combinations

$$\left\lfloor \log_2 \left(\binom{\mathcal{M}_k^*}{d} \times \alpha \right) \right\rfloor = \lambda$$

where α is the amount of valid sets where d samples of $\mathcal{M}_k^*$ are coprime. Tables 7 and 6 should help setting parameters.

In setting parameters, we also need to be careful not to create any form of overflows that would reject otherwise valid signatures. Thus, we are also giving in table 8 below the size of our outputs.

Notice that our generator parameters scale very well when we aim to increase combination security: jumping from $\lambda = 128$-bits of security to $\lambda = 256$ often requires less than doubling the amount d of vectors. In practice, recommended lattice dimensions tend to change with λ so a reevaluation of optimal parameters (d,k) is almost always necessary. Nevertheless, our results allow to use smaller (d,k) compared to the prime approach of [44] for [43], leading to smaller memory requirements for [43], less precomputations and faster verification:

- For $\lambda = 128$, [44] proposes $d = 6$ with 28-bits primes: with our generator, $d = 6$ with $k = 8$ (24-bits integers) is enough and so is $d = 5$ with $k = 9$ (28-bits integers).
- For $\lambda = 192$, [44] proposes $d = 9$ with 27/28-bits primes: with our generator, $d = 7$ and $k = 9$ is enough and so is $d = 9$ and $k = 8$.
- For $\lambda = 256$, [44] proposes $d = 12$ with 24 to 28-bits primes: with our generator, $d = 10$

Table 6 Amount of sets of d pairwise coprime out of 10000 calls to $(\mathbf{BaseCop}(\mathcal{M}_k))^d$

d \ k	5	6	7	8	9	10	11
4	8792	9144	9243	9469	9480	9602	9640
5	8078	8626	8779	9105	9171	9338	9435
6	7327	8035	8303	8679	8838	9035	9144
7	6506	7357	7767	8234	8449	8707	8844
8	5725	6652	7214	7727	8016	8293	8482
9	4980	5964	6604	7180	7551	7884	8094
10	4243	5298	6036	6668	7089	7439	7712
11	3618	4636	5423	6096	6617	6998	7266
12	3025	4066	4795	5575	6117	6509	6833
13	2491	3497	4243	5086	5644	6078	6394

Table 7 Value of $\lfloor \log_2 \left(\binom{\mathcal{M}_k^*}{d} \right) \rfloor$

d \ k	5	6	7	8	9	10	11
4	46	62	79	96	116	135	156
5	56	76	97	119	143	168	194
6	66	90	115	142	171	200	231
7	76	104	133	164	198	232	268
8	85	117	150	186	225	264	305
9	94	130	168	208	251	296	342
10	104	144	185	230	278	327	379
11	113	157	203	252	304	358	415
12	122	170	220	273	331	390	452
13	130	182	237	295	357	421	488

Table 8 Size of M_k in bits (lowest upper-bound of the size of ω_i), using k consecutive primes

k	1	2	3	4	5	6	7	8
k-th prime	2	3	5	7	11	13	17	19
$\lceil \log_2 M_k \rceil$	2	3	5	8	12	15	19	24
k	9	10	11	12	13	14	15	16
k-th prime	23	29	31	37	41	43	47	53
$\lceil \log_2 M_k \rceil$	28	33	38	43	49	54	60	65

and $k = 9$ is enough, and so is $d = 12$ and $k = 8$.

Using Barrett's algorithm [6] for modular reduction (thus consider it as equivalent to a multiply-and add), we can estimate the arithmetic operation cost of a verification process using our procedure. Note that a linear increase on a security parameter λ does not continuously increase the number of required operations: the number of operations is based on finding the combination (d, k) with the lowest computational cost such that the security is above λ bits. Hence it is natural that whole intervals of security parameters share the same minimal choice (d, k) for 64-bits processor using a process call "lazy reduction": we only proceed to a modular reduction to avoid overflows. Figure 3 fixed the security parameter at some of the security levels requested by the NIST ($\lambda = 128, 192, 256$) and shows we perform less operations for verifications than [44]'s approach from dimensions 512 to 1024 which are also the lowest and highest lattice dimensions proposed by [18]. The gain is not so noticeable for $\lambda = 128$, but the gap tends to widen as security parameters increase: our approach scales better.

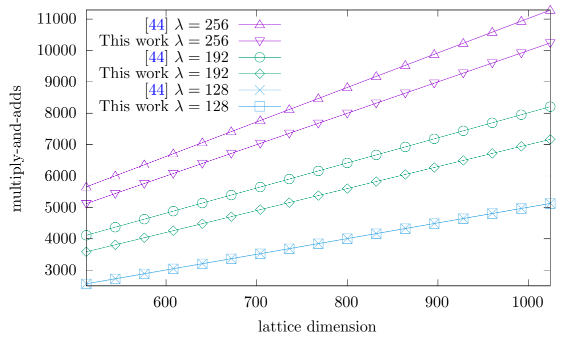

Fig. 3 Approximate number of multiply-and-adds between this paper and [44] during a verification

Once the parameters are set, we can also somehow approximate the memory needed for this generator to run. While it is impossible to predict the program size or the size of the random bit generator as those can be implemented at the hardware level, we can determine the memory needed for the variables and intermediate computations: it is roughly the memory needed for a CRT. Assuming sequential operations, so first we need the combined bit size of all k moduli, plus the size of the largest one to manage the random bits, and given s the bit size of their product, we need an extra $(k+2)s$ (for the modular inverse constants, the moduli product, and the size of intermediate results including the final value). So for $\lambda = 128$ with $k = 8$ and thus $s = 24$ (see table 8), the random moduli generator theoretically needs no more than 272-bits. For a parallel operation (i.e treating all components at the same time), twice this memory might be needed at most, as most intermediate results can be bounded by the moduli or their products.

4 Application to unstructured lattices

We extend in this section this work by applying the compression to schemes that would rely on a *non-algebraically* structured lattice signature scheme. Ignoring the process of both key generation and signature, we focus on the storage of the public key and the efficacy of verification algorithm, in which we recall the structure and the computations. We will first deal with the case of co-cyclic lattices, and then deal with the case of q-ary lattices. We will show that our adaptation is not straight-forward: while it was possible to improve the work of [44] without much drawbacks with a simple number generator, it is not the case for the following generalizations to other unstructured lattices. In both q-ary lattice and co-cyclic lattice cases, we will show that requesting extra information from the signatory is necessary. Thankfully, there should not be any loss of security as this extra information was easily computable from any party in the original scheme. The size of this overhead however, is large and further work will be needed to reduce its size, especially in the q-ary case.

Without going into details of computational lattice theory, the heuristic security of unstructured lattice-based schemes is often based on two parameters: the dimension and the determinant. Thus, if we wish to compare both q-ary lattices and co-cyclic lattices of equal dimension n over $\mathbb{Z}$ with the same security parameters, we could fix both determinants to be q^l some power of q (and $l < n$). For the representation of a q-ary lattice over $\mathbb{Z}^{n\times n}$, it follows that if the determinant is q^l, the "interesting block" has size $(n-l)\times l$. For simplification, we would note D the determinant for a co-cyclic lattice.

4.1 Co-cyclic lattices with perfect Hermite Normal Form

In a hash-and-sign approach of signature schemes with a lattice generated by a perfect HNF $\mathbf{A}$, the validity for a message $\mathbf{m}$, a signature $\mathbf{s}$ and a hash function h is determined by verifying the following:

$$\alpha = \mathbf{s} - h(\mathbf{m}) \in \mathcal{L}(\mathbf{A})$$

and s "small enough". As q-ary lattices make up for the large majority of recent lattice-based cryptography publications (and thus computations over $\mathbb{Z}_q$ rather than $\mathbb{Z}$), we believe it might be worthwhile to give some reminder and graphic representation of how traditional membership tests are done for classical lattices. We show below an example of checking whether $\alpha \in \mathcal{L}$:

Example 1. *Is* $\alpha = [1 \ -1 \ 1 \ 1] \in \mathcal{L}$*?*

$$\left[\begin{array}{cccc} 17 & 0 & 0 & 0 \\ 10 & 1 & 0 & 0 \\ 15 & 0 & 1 & 0 \\ 13 & 0 & 0 & 1 \\ \hline 1 & -1 & 1 & 1 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 17 & 0 & 0 & 0 \\ 10 & 1 & 0 & 0 \\ 15 & 0 & 1 & 0 \\ 13 & 0 & 0 & 1 \\ \hline -12 & -1 & 1 & 0 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{cccc} 17 & 0 & 0 & 0 \\ 10 & 1 & 0 & 0 \\ 15 & 0 & 1 & 0 \\ 13 & 0 & 0 & 1 \\ \hline -27 & -1 & 0 & 0 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 17 & 0 & 0 & 0 \\ 10 & 1 & 0 & 0 \\ 15 & 0 & 1 & 0 \\ 13 & 0 & 0 & 1 \\ \hline -17 & 0 & 0 & 0 \end{array}\right]$$

The final step is easy to guess (reduce to 0*) so the answer is yes.*

In the above example the determinant is 17 for easy manual verification but it is easy to see that in the parameters we described, the verification is done with $n-1$ additions and $n-1$ multiplications over $\mathbb{Z}_D$. If we denote $\alpha = [\alpha_1, ..., \alpha_n]$, we can denote $\beta = [0, -\alpha_2, ..., -\alpha_n]$ and notice the above computations can be represented by verifying

$$\beta\mathbf{A} + \alpha = \mathbf{x} \text{ with } \mathbf{x} = [\mathbf{x}_1, 0, ..., 0] \in \mathbb{Z}^n$$
$$\mathbf{x}_1 \equiv 0 \mod D \text{ i.e } \mathbf{x}_1 = \delta \times D$$

If we wish to apply the work of [44] the transformation gives

$$\beta\mathbf{v}^\top_{(A,\omega_1)} + \alpha\mathbf{v}^\top_{\omega_1} = [\delta D, 0, ..., 0]\mathbf{v}^\top_{\omega_1} \mod \omega_1$$
$$\vdots$$
$$\beta\mathbf{v}^\top_{(A,\omega_d)} + \alpha\mathbf{v}^\top_{\omega_d} = [\delta D, 0, ..., 0]\mathbf{v}^\top_{\omega_d} \mod \omega_d$$

Note on the acceleration of the verification At first sight this might look worse as it performs 2 scalar products per moduli ω (the right hand side of the equations are actually just a multiplication between 3 integers) and the original verification was already somewhat efficient with a complexity close to a single scalar product. However $\Omega = \omega_1...\omega_d$ is in fact *much* lower than the determinant D, and this allows parallelization while D might need multiprecision arithmetic. Without our modifications, the verification computations are done modulo the determinant D which is typically in the order of several thousands of bits, while in our modification Ω is typically less than 300 bits and allows to use the CRT to parallelize the operations [2].

As one of the main point of this paper is to avoid the storage of the public key $\mathbf{A}$, it is not possible to compute δ without $\mathbf{A}$. We need the signatory to provide δ to the verifier: this is a change compared to [44] that did not require extra information. However, δ was available to all parties in the original schemes: thus there should not be any loss of security. If the old signature had the form $\mathbf{s}_{old} = \{\mathbf{s}\}$, the new signature has the form $\mathbf{s}_{new} = \{\delta, \mathbf{s}\}$, where δ is an integer of size $\log_2(n)$ supposing $h(\mathbf{m})$ is already reduced to a single non-zero coefficient. A tighter upper-bound for the extra integer's size is $\log_2(\|\mathbf{s}\|_1)$ but we prefer to avoid mixing norms. Other approaches than hash-and-sign exist, but as unstructured co-cyclic lattices have not seen a practical use since the work of [20], we do not feel the need to present how the generalization would apply in other signature constructions. In particular, there is no round 1 candidates in the NIST competition that rely on co-cyclic lattices. As its core, we believe the verification of the equality is easily adaptable.

[2]see values of q, l for [14, 18] compared to matching security values d, ω_i in tables 8,6,7

4.2 Application to q-ary signature schemes

In the case of q-ary lattices, it would be tempting to not use random moduli and keep the operations over q, as q is typically chosen for efficiency purposes (as a power of 2 for example). However, according to [44], that drastically decreases the security: an attacker would have a probability of $1/q$ to output a valid signature in the eyes of the verifier' side. Since typically $q \leq 2^{64}$, that is a major security concern. Thus, we still need to carry operations over $\mathbb{Z}$, and extend the signature by a number of bits of the order of the dimension where we request from the signatory to give us *the amount of modular reduction modulo q that a regular verifier would do.* To the best of our knowledge, there is currently two main frameworks for the verification algorithm concerning q-ary lattices: one is the famous upgraded hash-and-sign from [23], the other one is the Fiat-Shamir application of [31]. In both cases, we can consider that the public key is a full-rank matrix $\mathbf{A}$ over $\mathbb{Z}_q$, and that all computations are over $\mathbb{Z}_q$ and h is a secure hash function.

The respective verification processes are as follows:

- In [23], given a signature $\mathbf{s}$ from a message m, the verification process check that $\mathbf{s}$ is short and that $\mathbf{sA} = h(m)$.
- In [31], given a signature $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$ from a message m, the verification process check that $\mathbf{s}_3$ is small with $\mathbf{s}_3 = h(\mathbf{s}_1\mathbf{A} + \mathbf{s}_2 - \mathbf{s}_3\mathbf{T}, m)$ where $\mathbf{T}$ is another public matrix.

Both verifications involve a multiplication of vector and a matrix, thus at first sight, Freivalds' algorithm can apply. Thus, we will attempt to apply our techniques to the hash-and-sign approach of [23]. For the Fiat-Shamir approach of [31], we refer the readers to the appendix. Of course, our technique mostly applies if verifiers do not have to reconstruct the short vectors from the public key, or hash parameters, as the goal is to not store the public key.

Let us denote $\mathbf{A}_\mathbb{Z}$ the representative of $\mathbf{A}$ over $\mathbb{Z}$. Let us denote $\mathbf{s}_q$ the vector representing the amount of reduction modulo q per coordinate that are done during a classic verification over $\mathbb{Z}_q$. The previous verification over $\mathbb{Z}_q$ whether or not $\mathbf{s} \times \mathbf{A} = h(m)$ becomes

$$\mathbf{s} \times \mathbf{A}_\mathbb{Z} - q\mathbf{s}_q = h(m) \text{ over } \mathbb{Z}$$

thus increasing the size of the signature by the size of $\mathbf{s}_q$. Thus, to apply our modification the signature has to change shape from $\mathbf{s}_{old} = (s)$ to $\mathbf{s}_{new} = (\mathbf{s}_q, s)$. This does not reduce the security as $\mathbf{s}_q$ could be computed in polynomial time in the original scheme by any party from a public perspective. We then need to precompute $\mathbf{v}_{A,\omega_i} = \mathbf{A}_\mathbb{Z} \times \mathbf{v}_{\omega_i}^\top \mod \omega_i$ and $\mathbf{v}_{q,\omega_i} = q \times \mathbf{v}_{\omega_i}^\top \mod \omega_i$, and the new verification process becomes

$$\begin{aligned}
\mathbf{s} \times \mathbf{v}_{A,\omega_1} - \mathbf{s}_q \times \mathbf{v}_{q,\omega_1} &= h(m) \times \mathbf{v}_{\omega_1}^\top \mod \omega_1 \\
&\vdots \\
\mathbf{s} \times \mathbf{v}_{A,\omega_d} - \mathbf{s}_q \times \mathbf{v}_{q,\omega_d} &= h(m) \times \mathbf{v}_{\omega_d}^\top \mod \omega_d
\end{aligned}$$

The main drawback to this approach is the size of k: while reducing the storage from a quadratic requirement to a linear requirement is attractive, we impose on the signatory to adds an extra computation step that produce an extra information (which do not reduce the security) potentially larger than the initial signature itself. Compared to the co-cyclic case, the extension of the signature is not limited to a single integer but is represented by a whole vector $\mathbf{s}_q$ which has size $(n-l)^2 \log_2(\|\mathbf{s}\|_\infty)$(there is also a tighter upper-bound using the taxicab norm) for [23]. For [31], this is even worse (see appendix). Decreasing the size of this extra information seems to be a challenging task and we leave it as an open question.

4.3 Extended framework

In both applications to the q-ary case and the co-cyclic case, the framework of [44] is slightly changed by requiring the signatory to add some extra information $\mathbf{s}_{extra}$ to the signature to ensure the verification is valid. In both cases, $\mathbf{s}_{extra}$ was easily computable in the original scheme thus there is no apparent loss of security. The new framework is then

* *(new offline step)* Alice compute the couples $(\omega_i, \mathbf{v}_{\omega_i})$.
1 Bob sends to Alice a public key $\mathbf{A}$
* *(new offline step)* Alice compute $\mathbf{v}_{(A,\omega_i)}^\top$ (and $\mathbf{v}_q$ if needed).
2 Alice sends to Bob a message $\mathbf{m}$

3 *(modified step)* Bob signs $\mathbf{m}$ with his signature $\mathbf{s}_{new} = (\mathbf{s}_{extra}, \mathbf{s}_{original})$
4 *(modified step)* Alice verifies that $\mathbf{s}_{new}$ is valid for $\mathbf{m}, \mathbf{A}, \mathbf{v}_{\omega_i}, \omega_i$ for all i.

4.4 Comparisons

We evaluate in this section the size of the different storages of information on the public key and the size of the signatures of the concerned schemes.

For security reasons, we know that $\|\mathbf{s}\|_\infty \sim D^{1/n}$ (where D is the lattice determinant) to guarantee that the corresponding lattice problem is hard to break. Consequently, we obtain $O(\log(D)) = n\log(\|\mathbf{s}\|_\infty)$. As the public key is in $O(n\log(D))$, we instead store an information of the public key in size $n^2\log(\|\mathbf{s}\|_\infty)$.

For the case of $q - ary$ lattices, the determinant is given by $D = q^l$ and the public key size is in $O((n-l)\log(D))$, we instead store an information on the public key in the size of $O((n-l)n\log(\|\mathbf{s}\|_\infty)$.

Furthermore, in the Freivalds' version, recall that $\Pi_{i=1}^{d}\omega_i \sim 2^\lambda$, therefore we can estimate that $d\log(\omega_i) = O(\lambda)$. We also recall that for security reason, we have $\lambda << n$. and that generally $\log(\|\mathbf{s}\|_\infty) = O(n)$.

We present the different magnitudes in terms of storage (or number of integral operations required) *for the verifier* as well as the size of signatures in table 9 below.

4.5 Discussion on comparisons

In this subsection we briefly discuss the downsides given to the signatory and eventual scenarios where such a trade-off would be relevant. Signature verification speed, before our work, was already believed to be fast compared to the setup and signing methods used in several lattice-based signature schemes. Thus, it is legitimate to wonder in which scenarios is the extra cost on the signatory negligible.

First, the extra cost on the signatory on our proposed trade-off is roughly equivalent to the cost of an original verification. Thus, if the original verification was already fast and lightweight, then so is the extra computation required on the signatory. Furthermore, our work is focused on *unstructured* lattice schemes: managing keys on unstructured lattices was already deemed to be heavy, thus we have to suppose that the signatory must have significant capabilities in both storage and computational capability to handle unstructured lattices (unless new research on unstructured lattices prove it otherwise). In this case, adding a small extra cost on the signatory is seemingly acceptable as we can then allow the use of unstructured lattices on low capability verifiers.

Second, this can also be a gain in a multi-user scenario where every user generate privately their own large unstructured key. Supposing there is a large amount of users k where n is the dimension of the lattice, then managing your own key has a memory cost of n^2, while managing the public key of the other users you communicate with then has a cost of kn^2. With our proposed modification, verifying other users' signatures only require a cost of kcn (where c is the number of moduli ω_i determined by the security parameter λ, and not directly linked to n). While this does not remove the fact that each user has to deal with his own unstructured key, at least the storage requirement on other users' keys is comparable to one of a structured lattice.

Practical comparisons done with experimental implementations of lattice-based schemes is currently hard to achieve: the main reason being the lack of *recent unstructured* lattice-based signature schemes. The work of [44] provided experimental time and memory space comparisons, using an open source implementation of preexisting scheme. In our case, there is *no open source implementation* of a HNF based unstructured lattice-based signature scheme available online. Nevertheless,

- Coprime generation is experimentally faster than prime generation and require *way less* memory, this is an improvement to [44].
- Smaller parameters than [44], given at the end of section 3, automatically implies less operations, thus faster operations and better trade-off for subsequent operations.
- If an unstructured co-cyclic lattice has dimension n, the signature only carries $\log_2(n)$ as an extra size-cost: this is relatively

Table 9 Verifier Public key and Signature size

	Classic		Freivalds' version	
Key Shape	Public Key	Signature	Public Key	Signature
Co-cyclic	$n^2 \log(\|\mathbf{s}\|_\infty)$	$n \log(\|\mathbf{s}\|_\infty)$	$n\lambda$	$n \log(\|\mathbf{s}\|_\infty)$
q-ary [23]	$(n-l)n \log(\|\mathbf{s}\|_\infty)$	$(n-l) \log(\|\mathbf{s}\|_\infty)$	$(n-l)\lambda$	$(n-l) \log(\|\mathbf{s}\|_\infty) + l \log(n-l)$
NTT-based	$n \log(\|\mathbf{s}\|_\infty)$	$n \log(\|\mathbf{s}\|_\infty)$	-	-

small compared to the original size of classically $n \log_2(n)$ of lattice-based signatures.

Whenever an open-source implementation of an unstructured lattice-based signature scheme is published, a practical comparison using our modification can be easily implemented; the setup of the key is unchanged, and the signature process only requires to store the amount of modular reduction and concatenate it with the signature instead of discarding it.

5 Open questions

We end this paper by remarks on the security model of [44] where they assume that guessing moduli is sufficient to break the system. As this assumption is "paranoiac" in their own words to ensure security against a yet-to-be-known attack, we can choose to shift the security requirements to a looser requirement. We want to stress that as this point, the security is unknown: the "public lattice" problem might have changed and its theoretical security would need to be studied. However, we explain here how those changes would apply if no security concerns are raised. We also mention other relevant topics as alternative probabilistic and deterministic verifications, structured lattices, and side-channel attacks.

Freivalds' technique to co-cyclic signatures without hidden $\mathbf{v}_{\omega_i}$

As shown previously, the original verification on perfect HNF of co-cyclic lattices already seemed fast enough. If the security is based on $\Omega = \omega_1 ... \omega_d$ anyway, we can transfer the verification over q^l to a verification over Ω using the exact same operations. δ the amount of reductions modulo q^l is still necessary, however this allows to cut the amount of operations and the amount of memory required by half. It is unclear how much this modification decreases the actual security.

Freivalds' technique to q-ary signatures without hidden ω

We choose here that the requirement be only the probability of setting false positives and keeping the same moduli q of the original lattice. This would allow to discard the heavy vectors $\mathbf{s}_q$. Like above, it is unclear how much this decreases the actual security. Suppose that instead of multiplying by a vector $\mathbf{v}$, we multiply by α distinct vectors (i.e a matrix), and keep the computations modulo q as per the original system. This sets the probability of getting a false positive to q^α, and if we assume that the dimension is large enough and the vectors random enough to be secure this leads to a new construction, where the original signature is left unchanged. Given $\mathbf{A} \in \mathbb{Z}_q^{n \times c}$ the public key, note $\mathbf{B} \in \mathbb{Z}_q^{c \times \alpha}$ a random matrix. A new verification process we use for [23] is

$$\mathbf{s} \times \mathbf{B}_A = h(m) \times \mathbf{B} \mod q$$

where $\mathbf{B}$ and $\mathbf{B}_A = \mathbf{A} \times \mathbf{B}$ are precomputed at verifier's setup. It is very important that the moduli q is kept the same in this case, as replacing by a divisor makes the problem easier for an attacker: it opens a way for an attacker to guess a false positive by solving a lattice problem over a lattice which determinant is lower than the original determinant (a divisor of the original determinant). Here, the acceleration only makes sense if it is guaranteed that $\mathbf{B}$ is *much* smaller than the public key $\mathbf{A}$, as their coefficients and arithmetical properties are basically similar (assuming $\mathbf{A}$ is structureless). In this new case, we just need to sample α vectors such that $q^\alpha > 2^\lambda$ for λ-bits of security.

Finer analysis of the generated set

As shown in table 6, the set of lists of pairwise coprimes is slighty smaller (a fraction). Cryptographicly, this difference is consequently negligeable (few bits of security).

However, to obtain a complete security evaluation, one could further study the distribution of the co-primes used. At this stage, we cannot anticipate any futher weakness; as long as the cost of guessing correctly a moduli randomly *by* forging a signature on a "weaker" lattice is higher than the security parameter, this will ensure safety; every potential attempt on guessing the set of moduli in the current heuristic security model is *assumed to be made* by submitting a valid signature for the secret moduli that would be incorrect for the original scheme without our modification. Nervertheless, the study of the distribution of generated co-primes remains an interresting question especially if one will use this technique for other applications that cryptographic ones.

Side-channel attacks

Adding a secret to a verifier can sound counter-intuitive and it is natural to ponder the question of whether or not additional countermeasures against side-channel attacks for verifiers would make the verifier *slower* than originally. Some points come to mind. First, depending on the use-case, if high-speed is not a necessity but memory storage is, our technique is still useful for unstructured lattices. It seems unlikely that potential countermeasures will offset the linear factor gained both in key compression and verification speed. Second, if novel knowledge shows that coprimality is not required and that some redundancy is tolerable, generators and all operations can be made in constant-time. In particular, we can generate random pseudo-mersenne, thus it is possible that there is no need for looping inversion algorithms (and conditionals) as long as proper targeted reductions to avoid overflows are done. Third, random draws of moduli is a technique used in RNS to protect against side-channel attacks [4]. As we do not require modular inverses or base extensions, we could also use redundant arithmetic or random draws of moduli without resorting on coprimality. Like in [4, 10, 35], a high number of possible combinations is at the core of the security, and our generator provide it. The issue is the lack of guaranteed coprimality *and* maybe the computation of modular inverses *in constant time without branching*, but this could be a (very difficult) further research direction. If this last point is solved, the generator could prove to be a direct side-grade to [4] using [7] for modular inversion, with the extra cost of rejecting and resampling the non-coprime numbers. There might be other concerns, but setting secrets on verifiers seems relatively uncommon and can be interesting to study.

Other open questions

Many q-ary lattice schemes nowadays rely on polynomial structures[8, 15, 18, 21]: matrices $\mathbf{A}$ are stored in two polynomials (or group of polynomials) (f, g) where $f \in \mathbb{Z}_q[X]/g$ (ring version) or $(\mathbb{Z}_q[X]/g)^b$ (module version). g is often a parameter of the cryptosystem, leaving f as the only variable public key. Each row of $\mathbf{A}$ is of the form $X \times f \mod g$. Verification of equalities and computations are done over $\mathbb{Z}_q[X]/g$, which leads to compressed key sizes and accelerations. On those particular cases, it is possible that our storage requirement becomes larger than the original scheme: the public key can be seen as a single vector (i.e a single polynomial), while our precomputed vectors over multiple moduli can be seen as many. However, our efficiency for verification should remain faster: whether the cryptosystem uses Karatsuba's algorithm [28], the Number Theoretic Transform (NTT) [41] or a combination of both [33] to verify equations, it should not be asymptotically faster than a constant number of scalar products. Furthermore, NTT operations often require the precomputation and storage of roots of unity, thus store more than just the key: for a lattice of dimension n, a full NTT requires operations in the order of $n \log_2(n)$ (a multiple of n for each level of the "butterfly" step, and there is exactly $\log_2(n)$ steps). Our approach requires cn, where c *does not depends* on n but on the property of the number generator and the required security level λ. Overall, at this current state of research the trade-off is not interesting for structured lattices for now. In fact, it is possible that those structures can be exploited to accelerate the verifier setup and the extra signature information required for our work to apply. For now we keep our work generic as its core application is unstructured lattices. Note that there have been some attempts to use a probabilistic verification for polynomial structures that could apply for cryptography: [24] adapted the probabilistic evaluation of polynomials to middle products [42]. It is unclear if their technique actually outperforms ours or how to evaluate its security when modifying the verifier. Another direction is to pursue the generic approach of probabilistic or deterministic verification [29].

6 Conclusion

We exhibited a random number generator that has good properties to *almost* qualify as a random RNS base generator, i.e generates sets of pairwise coprimes with high probability. This directly allows us to improve [44]'s original work. We also generalize [44]'s work to apply to q-ary lattices which includes all lattice-based NIST candidates since round 1 although most are structured lattices, and showcase interesting performance tradeoffs for faster verification and lower public key storage for unstructured lattice signature schemes when a key is reused at the cost of a precomputation. Overall, we show that unstructured lattice can be, larger initial communications aside, *almost* as efficient as structured lattices for the verification processes of lattice-based signatures. This can help the engineering challenge of designing a public key infrastructure with low-capacity verifiers. As we stated in the introduction, there is currently *no unstructured lattice-based signature scheme* selected in the round 3 of NIST. If another lattice-based signature has to be submitted for the round 4, then according to their most recent announcement *it has to be unstructured* [38]: in this case, this work can apply and further improvements can be done.

Appendix A Adapt to [31]

For [31], we reuse the same notations, $\mathbf{s}_q$ representing the modular reductions, $\mathbf{T}_\mathbb{Z}$ the representative of $\mathbf{T}$ over $\mathbb{Z}$, and the verification over $\mathbb{Z}_q$ of $\mathbf{s}_3 = h(\mathbf{s}_1\mathbf{A} + \mathbf{s}_2 - \mathbf{s}_3\mathbf{T}, m)$ becomes:

$$\mathbf{s}_3 = h(\mathbf{s}_h, m) \text{ and } \mathbf{s}_h = \mathbf{s}_1\mathbf{A}_\mathbb{Z} + \mathbf{s}_2 - \mathbf{s}_3\mathbf{T}_\mathbb{Z} - q\mathbf{s}_q \text{ over } \mathbb{Z}$$

We prefer not to alter the hash function or its output as it is a core security part of the original construction, thus we only change the computation of the parameter within the hash function. $\mathbf{s}_{old} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$ changes to $\mathbf{s}_{new} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_h, \mathbf{s}_q)$ where $\mathbf{s}_h = \mathbf{s}_1\mathbf{A} + \mathbf{s}_2 - \mathbf{s}_3\mathbf{T}$. Since $\mathbf{A}, \mathbf{T}$ are public and $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3$ are the components of $\mathbf{s}_{old}$, there is no loss of security. Noting $\mathbf{v}_{T,\omega_i} = \mathbf{T}_\mathbb{Z} \times \mathbf{v}_{\omega_i}^\top \bmod \omega_i$ we obtain

$$\mathbf{s}_3 = h(\mathbf{s}_h, m)$$

$$\mathbf{s}_h = \mathbf{s}_1\mathbf{v}_{A,\omega_i} + \mathbf{s}_2\mathbf{v}_{\omega_1}^\top - \mathbf{s}_3\mathbf{v}_{T,\omega_1} - \mathbf{s}_q\mathbf{v}_{q,\omega_1} \mod \omega_1$$
$$\vdots$$
$$\mathbf{s}_h = \mathbf{s}_1\mathbf{v}_{A,\omega_i} + \mathbf{s}_2\mathbf{v}_{\omega_d}^\top - \mathbf{s}_3\mathbf{v}_{T,\omega_d} - \mathbf{s}_q\mathbf{v}_{q,\omega_d} \mod \omega_d$$

References

[1] Aiken H, Semon W (1959) Advanced digital computer logic. Comput Lab, Harvard Univ, Cambridge, Mass, Rep WADC TR-59-472

[2] Ajtai M (1996) Generating hard instances of lattice problems. In: STOC '96: the twenty-eighth annual ACM symposium on Theory of Computing, pp 99–108

[3] Bajard JC, Duquesne S (2021) Montgomery-friendly primes and applications to cryptography. Journal of Cryptographic Engineering 11:399–415

[4] Bajard JC, Imbert L, Liardet P, et al (2004) Leak resistant arithmetic. In: Cryptographic Hardware and Embedded Systems - CHES 2004, LNCS, vol 3156. Springer, pp 62–75

[5] Bajard JC, Fukushima K, Kiyomoto S, et al (2021) Generating residue number system bases. In: 28th IEEE Symposium on Computer Arithmetic

[6] Barrett P (1986) Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In: Conference on the Theory and Application of Cryptographic Techniques, Springer, pp 311–323

[7] Bernstein DJ, Yang BY (2019) Fast constant-time gcd computation and modular inversion. IACR Transactions on Cryptographic Hardware and Embedded Systems 3:340–398

[8] Bernstein DJ, Chuengsatiansup C, Lange T, et al (2018) NTRU prime. NIST Post-Quantum Cryptography Standardization, URL https://ntruprime.cr.yp.to/

[9] Cohen H (1993) A Course in Computational Algebraic Number Theory, vol 138 of Graduate Texts in Mathematics. Springer-Verlag

[10] Courtois J, Abbas-Turki L, Bajard JC (2019) Resilience of randomized rns arithmetic with respect to side-channel leaks of cryptographic computation. IEEE Transactions on Computers 68(12):1720–1730

[11] Cramer R, Ducas L, Wesolowski B (2021) Mildly short vectors in cyclotomic ideal lattices in quantum polynomial time. Journal of the ACM 68(2):1–26

[12] Crandall R (1992) Method and apparatus for public key exchange in a cryptographic system. US Patent 5,159,632. US Patent and Trade Office (Oct 1992)

[13] Dinur I (2007) The PCP theorem by gap amplification. Journal of the ACM 54(3):12–es

[14] Ducas L, Kiltz E, Lepoint T, et al (2021) Crystals-dilithium algorithm specifications and supporting documentation. NIST Post-Quantum Cryptography Standardization, URL https://pq-crystals.org/

[15] Erdem A, Roberto A, Joppe B, et al (2018) NewHope Algorithm Specifications and Supporting Documentation. 1st edn., https://newhopecrypto.org/data/NewHope_2018_06_14.pdf

[16] Feige U, Goldwasser S, Lovász L, et al (1991) Approximating clique is almost NP-complete. In: 32nd Annual Symposium of Foundations of Computer Science, pp 2–12

[17] Fouque PA, Tibouchi M (2019) Close to uniform prime number generation with fewer random bits. IEEE Transactions on Information Theory 65(2):1307–1317

[18] Fouque PA, Hoffstein J, Kirchner P, et al (2018) Falcon: Fast-fourier lattice-based compact signatures over NTRU (specification v1.2). NIST Post-Quantum Cryptography Standardization, URL https://falcon-sign.info/(asof2022)

[19] Freivalds R (1979) Fast probabilistic algorithms. In: Mathematical Foundations of Computer Science, LNCS, vol 74. Springer, pp 57–69

[20] Gama N, Izabachene M, Nguyen PQ, et al (2016) Structural lattice reduction: generalized worst-case to average-case reductions and homomorphic cryptosystems. In: Advances in Cryptology – EUROCRYPT, LNCS, vol 9666. Springer, pp 528–558

[21] Garcia-Morchon O, Zhang Z, Bhattacharya S, et al (2018) Round5: merger of hila5 and round2. Post-Quantum Cryptography Standardization, URL https://github.com/round5/

[22] Garner HL (1959) The residue number system. In: Western Joint Computer Conference, ACM, pp 146–153

[23] Gentry C, Peikert C, Vaikuntanathan V (2008) Trapdoors for hard lattices and new cryptographic constructions. In: STOC '08: the fortieth annual ACM symposium on Theory of computing, pp 197–206

[24] Giorgi P (2018) A probabilistic algorithm for verifying polynomial middle product in linear time. Information Processing Letters 139:30–34

[25] Goldstein D, Mayer A (2003) On the equidistribution of Hecke points. Forum Mathematicum 15(2):165–189

[26] Hardy G, Wright E (First Edition 1938) An Introduction to the theory of numbers. Oxford University Press, London

[27] Joye M, Paillier P (2006) Fast generation of prime numbers on portable devices: An update. In: Cryptographic Hardware and Embedded Systems - CHES 2006, LNCS, vol 4249. Springer, pp 160–173

[28] Karatsuba AA, Ofman YP (1963) Multiplication of multidigit numbers on automata. Soviet physics doklady 7:595–596

[29] Korec I, Wiedermann J (2014) Deterministic verification of integer matrix multiplication in quadratic time. In: SOFSEM 2014: Theory and Practice of Computer Science, pp 375–382

[30] Liu YK, Lyubashevsky V, Micciancio D (2006) On bounded distance decoding for general lattices. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, LNCS, vol 4110. Springer, pp 450–461

[31] Lyubashevsky V (2009) Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Advances in Cryptology – ASIACRYPT 2009, LNCS, vol 5912. Springer, pp 598–616

[32] Lyubashevsky V, Micciancio D (2009) On bounded distance decoding, unique shortest

vectors, and the minimum distance problem. In: Advances in Cryptology - CRYPTO 2009, LNCS, vol 5677. Springer, pp 577–594

[33] Moenck RT (1976) Practical fast polynomial multiplication. In: SYMSAC '76: the third ACM symposium on Symbolic and algebraic computation, ACM, pp 136–148

[34] Moody D (2021) Status update on the 3rd round. Third PQC Standardization Conference, https://csrc.nist.gov/Presentations/2021/status-update-on-the-3rd-round

[35] Negre C, Perin G (2015) Trade-off approaches for leak resistant modular arithmetic in rns. In: Information Security and Privacy. ACISP 2015, LNCS, vol 9144. Springer, pp 107–124

[36] Nguyen PQ, Shparlinski IE (2016) Counting co-cyclic lattices. SIAM Journal on Discrete Mathematics 30(3):1358–1370

[37] NIST (2018) Post-quantum cryptography standardization. URL https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[38] NIST (2022) Post-Quantum Cryptography Standardization. URL https://csrc.nist.gov/news/2022/pqc-candidates-to-be-standardized-and-round-4

[39] Pellet-Mary A, Hanrot G, Stehlé D (2019) Approx-SVP in ideal lattices with preprocessing. In: Advances in Cryptology – EUROCRYPT 2019, LNCS, vol 11477. Springer, pp 685–716

[40] Plantard T, Sipasseuth A, Dumondelle C, et al (2018) DRS : Diagonal dominant reduction for lattice-based signature. NIST Post-Quantum Cryptography Standardization, URL https://thomas-plantard.github.io/drs/

[41] Pollard JM (1971) The fast fourier transform in a finite field. Mathematics of computation 25(114):365–374

[42] Roşca M, Sakzad A, Stehlé D, et al (2017) Middle-product learning with errors. In: Advances in Cryptology – CRYPTO 2017, LNCS, vol 10403. Springer, pp 283–297

[43] Sipasseuth A, Plantard T, Susilo W (2019) Improving the security of the DRS scheme with uniformly chosen random noise. In: Jang-Jaccard J, Guo F (eds) Information Security and Privacy. Springer International Publishing, Cham, pp 119–137

[44] Sipasseuth A, Plantard T, Susilo W (2019) Using freivalds' algorithm to accelerate lattice-based signature verifications. In: Information Security Practice and Experience, LNCS, vol 11879. Springer, pp 401–412