2019

Using Freivalds' Algorithm to Accelerate Lattice-Based Signature Verifications

Arnaud Sipasseuth
University of Wollongong, as447@uowmail.edu.au

Thomas Plantard
University of Wollongong, thomaspl@uow.edu.au

Willy Susilo
University of Wollongong, wsusilo@uow.edu.au

Using Freivalds' Algorithm to Accelerate Lattice-Based Signature Verifications

Abstract

© Springer Nature Switzerland AG, 2019. We present a novel computational technique to check whether a matrix-vector product is correct with a relatively high probability. While the idea could be related to verifiable delegated computations, most of the literature in this line of work focuses on provably secure functional aspects and do not provide clear computational techniques to verify whether a product $$xA = y$$ is correct where x, A and y are not given nor computed by the party which requires validity checking: this is typically the case for some cryptographic lattice-based signature schemes. This paper focuses on the computational aspects and the improvement on both speed and memory when implementing such a verifier, and use a practical example: the Diagonal Reduction Signature (DRS) scheme as it was one of the candidates in the recent National Institute of Standards and Technology Post-Quantum Cryptography Standardization Calls for Proposals competition. We show that in the case of DRS, we can gain a factor of 20 in verification speed.

Disciplines

Engineering | Science and Technology Studies

Publication Details

Sipasseuth, A., Plantard, T. & Susilo, W. (2019). Using Freivalds' Algorithm to Accelerate Lattice-Based Signature Verifications. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11879 LNCS 401-412. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)

Using Freivalds' algorithm to accelerate Lattice-Based Signature Verifications

Arnaud Sipasseuth[1,2], Thomas Plantard[1,2], and Willy Susilo[1,2]

[1] Institute of Cybersecurity and Cryptology
School of Computing and Information Technology
University of Wollongong

[2] [as447,thomaspl,wsusilo]@uow.edu.au

Abstract. We present a novel computational technique to check whether a matrix-vector product is correct with a relatively high probability. While the idea could be related to verifiable delegated computations, most of the literature in this line of work focuses on provably secure functional aspects and do not provide clear computational techniques to verify whether a product $xA = y$ is correct where x, A and y are not given nor computed by the party which requires validity checking: this is typically the case for some cryptographic lattice-based signature schemes. This paper focuses on the computational aspects and the improvement on both speed and memory when implementing such a verifier, and use a practical example: the Diagonal Reduction Signature (DRS) scheme as it was one of the candidates in the recent National Institute of Standards and Technology Post-Quantum Cryptography Standardization Calls for Proposals competition. We show that in the case of DRS, we can gain a factor of 20 in verification speed.

Keywords: Diagonal Reduction Signature, Post-Quantum Cryptography, Lattice-based Signatures, NIST, Delegated Computation Verification, Lattice-based cryptography

1 Introduction

Post-Quantum cryptography is currently being widely researched. Quantum computing is improving and it is not clear how long it would take for most currently used cryptographic primitives to be finally broken by quantum algorithms such as Shor's algorithm [25] or Grover's algorithm [13]. This has forced the National Institute of Standards and Technology (NIST) to call for a standardization of post-quantum primitives in the hope of having the readiness of the standarized algorithm when quantum computers arrive [19]. Nevertheless, research in cryptology still continues outside of the NIST standardization process. One such important result is the work of Gama, Izabachene, Nguyen and Xie [11] which proves that the reduction from average cases to worst cases problems on q-ary lattices, demonstrated by Ajtai [1] and Regev [23], can be extended to most random lattices. Therefore one is not required to rely exclusively on q-ary lattices when building a cryptosystem, unlike all lattice-based candidates in the second round of the NIST process. However, q-ary lattices can obtain a Hermite Normal Form (HNF) as a basis for almost free, which allows very fast verification whether a random vector belongs to

the lattice generated by the said basis, on top of significantly reducing key sizes [17]. Most lattice-based schemes rely on finding a vector of a lattice with specific properties, mostly being short. For non q-ary lattice schemes, obtaining a HNF is often costly. While computing a HNF can be done in polynomial time [20], the actual time used in practice to compute cryptographically secure HNF basis is too long for most applications. We propose in this paper a method to alleviate this issue, using a variant of Freivalds' algorithm [10] to verify the validity of matrix product. While the core ideas are also applicable to q-ary lattices and other HNF basis to some extent, we deem its impact not significant enough to expand on it, and rather we present an application to one of the schemes submitted to the NIST which did not rely on q-ary lattices, the Diagonal Reduction Signature scheme (DRS) [21]. DRS did not use a HNF as a public key: the authors of DRS used an alternative to generate public keys and used another way to verify if a vector belonged to a lattice. Our work provides an alternative to verify lattice signatures if the need arises, showing an interesting trade-off between pre-computation time for signature verification time and memory storage. In this work, we show that by adopting our approach to the proposed DRS parameters [27], we gain a factor of 20 on the verification speed. We also provide another approach on an attack specifically to our modification as the security of our public key remains unchanged. *The rest of this paper is organized as follows.* We will first recall some basic lattice definitions, Freivalds' algorithm and then briefly reintroduce DRS mostly focusing on its verification part. We will proceed with the presentation of our new technique and its results using simple non-optimized implementations, followed by comments on its security and concluding this work by raising open questions.

2 Background

2.1 Lattice basics

We call an integer lattice when a finitely generated subgroup of $\mathbb{Z}^n$. A basis of the lattice is a basis as a $\mathbb{Z} - module$. In our work we only consider full-rank integer lattices, i.e such that any basis B of a lattice $\mathcal{L}$ (we note $\mathcal{L} = \mathcal{L}(B)$) is full rank.

We call the max norm l_∞ norm: $\forall x \in \mathbb{Z}^n, \|x\|_\infty = \max_{i\in[1,n]} |x_i|$.

A valid signature in the DRS scheme solve an instance of $\mathbf{GDD}_\gamma$ (γ-Guaranteed Distance Decoding): given a lattice $\mathcal{L}$, $x \in \mathbb{Z}^n$ and a bound $\gamma \in \mathbb{Z}$, find $v \in \mathcal{L}$ such that $\|x - v\| \leq \gamma$.

2.2 Freivalds' algorithm

Freivalds' algorithm (algorithm 1) for verifying matrix products [10] is one of the first probabilistic algorithms to be introduced to show the

efficiency and practicality of non-deterministic programs to solve decision problems over deterministic ones. Freivalds' technique had a major impact on several research fields and is still an active research topic to this date [8, 9]. The decision problem solved by Freivalds is the following: given A, B, C, three $n \times n$ matrices over an arbitrary ring $\mathcal{R}$, *can we verify that* $A \times B = C$ *with a faster method than recomputing* $A \times B$*?* Freivalds brought a probabilistic solution, which rely on a simple statement: *to check* $A \times B = C$*, we check instead* $A \times (B \times v) = C \times v$ *where* v *is a randomly sampled vector,* and then it follows that the more we run this test, the more we decrease the probability of obtaining a false-positive. This leads to Freivalds' algorithm 1 which is *perfectly complete*: it will

Algorithm 1 Freivalds' algorithm

Require: $A, B, C \in \mathcal{R}^{n \times n}$, $f \in \mathbb{N}$ a failure probability
Ensure: Check the validity of $A \times B = C$ with a chance of false-positive under 2^{-f}

```
i ← 0
while i < f do
    v ← Randomly taken in {−1, 1}^n
    x ← Cv, y ← Bv, y ← Ay
    if x ≠ y then return FALSE                ▷ check validity
    i ← i + 1
return TRUE
```

always output ***TRUE*** whenever $A \times B = C$ is correct. It is also *sound*: the probability of outputting ***TRUE*** for $A \times B \neq C$ is negligible (as much as we want). The gain in efficiency compared to a deterministic method is quite impactful as this shifts the arithmetical computations from a matrix-matrix product to a matrix-vector multiplication. We are not recalling the proof on the original error probability bound in [10] but we will later give a much tighter upper bound which will be more adapted to our case.

2.3 DRS and its verification algorithm

DRS is one of the five lattice-based signature schemes that have been proposed to the NIST PQC competition [19]. The original idea behind DRS stemmed from when Plantard, Susilo and Win [22] suggested to use a diagonal dominant matrix to reduce large message vectors to short signatures within a known hypercube as a countermeasure against parallelogram detection attacks [18]. The security of the scheme has been shown to be reduced by a machine learning method [28] and since then it has been modified to resist against this attack at the cost of a slower secret key generation [27]. We will briefly describe both keys, the signature and the verification, the latter being the only part affected by this work. The secret diagonal dominant matrix (as defined in [7]) S_{key} is

generated using the work in [27]). The public key P_{key} is generated by the multiplication of $P_{key} = S_{key} \times U$ with U unimodular and randomly taken such that $\|P_{key}\|_\infty < 2^{63}$. The signature algorithm makes use of the structure of S_{key} to output (h, s) to solve the $\mathbf{GDD}_D$ problem on m with $hP_{key} = m - s$ and $\|m - s\|_\infty < D$. The verification algorithm, algorithm 2, checks two points: the first ensuring that the vector s is indeed short enough, and the second ensuring that $m - s$ is indeed a vector of the lattice. *This is where DRS differs from other lattice-based cryptographic schemes even from the original concept from [22]: DRS does not use a HNF.* Our understanding is that the computation time of a HNF was deemed too large to be suggested for practical uses, and thus the authors of DRS opted for another solution which impacted the verification algorithm as they could no longer use HNF to check $m - s \in \mathcal{L}$. Another important point about DRS is their choice to fit every computation within 64-bits to ensure that computation of $h \times P_{key}$ does not overflow without the use of multiprecision integers. The principle of this algorithm can be compared to a verification per block: we successively deal with parts of the input, where each part taken h' of h is chosen such that $t = \|h'P_{key}\|_\infty < 2^{63}$ and remove t from $m - s$ until t and h' reach 0 exactly at the same time. While this algorithm could be interesting to improve on its own, we suggest an alternative instance of the scheme where this verification method can be completely discarded.

Algorithm 2 DRS Verify

Require:
The message $m \in \mathbb{Z}^n$, the public key P_{key} both stored by Alice
The signature (s, h) given by Bob
Ensure:
The boolean value ($(hP_{key} = m - s)$ AND $(\|w\|_\infty < D)$)
1: **if** $\|w\|_\infty > D$ **then return FALSE** ▷ Test for max norm first
2: $q \leftarrow h$, $t \leftarrow v - w$ ▷ Loop Initialization
3: **while** $q \neq 0 \wedge t \neq 0$ **do**
4: $r \leftarrow q \mod \|P_{key}\|_\infty$, $t \leftarrow rP_{key} - t$
5: **if** $t \neq 0 \mod \|P_{key}\|_\infty$ **then return FALSE** ▷ Check correctness
6: $t \leftarrow t/\|P_{key}\|_\infty$, $q \leftarrow (q - r)/\|P_{key}\|_\infty$
7: **if** $(t = 0) \veebar (q = 0)$ **then return FALSE** ▷ Check correctness
8: **return TRUE**

3 Modifiying Freivalds' technique for lattice-based signature verification

3.1 The first core idea: Modification of Freivalds' algorithm

In this work, we modify Frivalds' technique to obtain a faster probabilistic verification algorithm. The vector pairs (k, m) to check are not given

by the person who needs the verification but by the signatory, and so is P_{key}. In the case where one public key is re-used over multiple message-signature exchanges, the equality $hP_{key} = m - s = v$ must stand true for all vector triplets (h, s, m) provided. In that case, we can introduce a random vector $x^\top$ and $X = P_{key} \times x^\top$ such that $(h \times X = v \times x^\top)$ which reduce a matrix-vector multiplication check to the comparison of two scalar products (vector-vector multiplications). A difference with the original Frivalds' algorithm is that we don't use $x \in \{-1, 1\}^n$ but rather, we will choose a prime p such that $x \in \mathbb{F}_p^n$ is taken randomly, and project the whole equation over the field $\mathbb{F}_p$. *For sufficiently many vectors x, X and primes p, let's say k primes and k vectors x, our new validity condition is then* $(h \times X_i = v \times x_i^\top \mod p_i)_{i\in[1,k]}$. Note that projecting Freivalds' algorithm over a finite field was proposed in [15] for a non-cryptographic purpose, however to the best of our knowledge there is no work that modify the algorithm in the manner we just described. The choice to use multiple different moduli will be expanded in section 4. Let us compute the probability of failure of our verification algorithm. First of all, the algorithm is *perfectly complete*, i.e it will never output a false negative. The last thing to check is then the probability of a false positive. In that regard, rather than thinking of probability of a false positive, let us first compute the proportion of positive results over all possibilities given a prime p. Let us enumerate all possibilities. If v is fixed, then $v*x^\top = a_p \mod p$ is also fixed. So the proportion of couples (h, m) giving a positive result is the probability of $\sum_{i=1}^{n} h[i]X[i] = a_p \mod p$. Without losing generality, if we choose and index j such that $X[j]$ is non-zero (X being obviously chosen non-zero) and fix every other coefficient h_i such that $b_p = a_p(\sum_{i\neq j} h[i]X[i])^{-1} \mod p$, we obtain the result of a positive output with the same proportion as verifying $h[i] = c_p \mod p$ which is $1/p$ for a given prime p. As this reasoning is sound for any $v = m - s$ and in any triplet (h, m, s), we determine the quantity of false positives being the difference between the amount of positive outcomes and the amount of valid positive outcomes, which set a proportion of false positives of being strictly under $1/p$ (and by extension its probability over all possible samples). If we repeat this process over k different vectors, the false positive probability lowers to below p^{-k}. Generally speaking, if we try the test once per couple prime/vectors over k primes $\{p_i, x_i\}_{i\in[1,k]}$, then *the probability of obtaining a false positive becomes lower than* $\prod_{i=1}^{k} p_i^{-1}$. This is a tighter upper bound over Freivalds' initial upper bound, although our work only concerns our very specific case and does not apply to the general scope of Freivalds' technique.

3.2 The second core idea: Changing the verifier

With our previous idea in mind, we need to explain what we aim to modify in the previous DRS scheme. First let us briefly recall how the sender/verifier Alice and the signatory Bob acts in 5 steps:

1. Bob generates a pair of keys $\{P_{key}, S_{key}\}$.
2. Bob keeps the secret key S_{key}, and sends the public key P_{key} to Alice.
3. Alice sends a random vector m with "large" norm $\|m\|_\infty$ to Bob.
4. Bob uses S_{key} to send the signature $\{h, s\}$ to Alice.
5. Alice verifies that $\|s\|_\infty$ is "low" and $hP_{key} = v = m - s$.

While we can consider the verification process to be entrusted to a third-party like a certification authority, here we restrict ourselves on exclusively modifying the computation of the verification which is step 5, and inserting a precomputation which can be placed after or during step 2. *One important point to stress on is that Alice does not need to communicate to Bob she is using a precomputation.* The whole process is oblivious to Bob and his role does not change at all compared with the existing DRS process. Hence, it seems natural for us to assume Alice will keep her computations secret as there is no apparent benefit in revealing them.

Precomputation The precomputation construct the samples required to apply our modified Freivalds' test and can be described in two halves as follows:

- Generate a family of tuples $(p_i, x_i)_{i\in[1,k]}$
- Compute $T = (p_i, x_i, X_i)_{i\in[1,k]}$ where $X_i = P_{key}x_i \mod p_i$ given P_{key}

The first half of the precomputation do not requires input from Bob as the dimension is supposed to be public, therefore those can even be precomputed before Bob generating his keys in step 1. The choice of random generators for primes and vectors are important for security and efficiency considerations, however those are not the main point of the paper. As far as our experimental results are concerned, we just used the basic random function "**rand()**" of the library "**stdlib.h**" in C with the classical modulo operator % to generate our vectors, and our primes are randomly taken in a set we will discuss in the security section of this paper, using the MAGMA software [6] to pick primes and write them into a header file used by our code before the compilation. Its computation time is negligeable compared to the second part of the precomputation which involves matrix-vectors modular multiplications.

In the second half of the precomputation, Alice does not need to store P_{key} at the end, and furthermore she does not even need to store the whole public key while computing X_i. Since for each row j of P_{key} Alice can independently compute $P_{key}[j] * x_i = X_i[j] \mod p_i$, Alice can discard every

row of P_{key} where the corresponding computation is finished and choose to only receive a certain amount of rows at a time, which would reduce the amount of internal memory required for the whole precomputation (and allow for further parallelism). The cost of the second half of the precomputation is the main cost of the whole precomputation process.

New verification method The new verification method will apply our modification on Freivalds' algorithm using our precomputation step. Alice, at step 5, previously discarded the public key P_{key} and kept some small footprint in the form of a secret list T of triplets and sent a random message m. As she received in step 4 the signature (k, s) from Bob, her verification process is now described by algorithm 3. This new verification algorithm is more compact than the original one and also simpler to understand. The only remaining point to deal with is to choose how large h and the primes p_i need to be. We will discuss that in the next section when discussing security.

Algorithm 3 New Verification

Require:
a list of triplets $T = (p_i, x_i, X_i)_{i\in[1,k]}$ and a message m from Alice
a signature (h, s) from Bob
a public bound D on the signature norm

Ensure:
a boolean R stating whether (h, s) is a valid signature for m and P_{key}
R is a false-positive with probability strictly less than $\prod_{i=1}^{k} p^{-1}$

1: $R \leftarrow \{\|s\|_\infty < D\}$ ▷ Verifies the max norm of the signature
2: **for** $i \in [1, k]$ **do**
3: $\quad R \leftarrow R \wedge \{hx_i = (m - s)x_i \mod p_i\}$ ▷ Verifies modular equalities
4: **return** R

4 Security considerations

While the previous attacks on the old DRS are well-understood heuristics relying either on machine learning [28] or pure lattice reduction as with most other lattice-based schemes (being signature-based or decryption-based), *our modification does not thwart previous attacks nor does it reinforce them and thus rely on the same security assumptions.* However, this is only when considering the only secret was the diagonal dominant matrix S_{key}. Here, we introduce a new secret, which is the list of triplets T generated by Alice. Thus, new attacks venues can be considered which, to the best of our knowledge, were also not considered in others lattice schemes submitted to the NIST. We will consider them in this section. We briefly present the two avenues we found and explain our reasoning on why only the second can be considered, and tackle this issue. Note that

our reasoning discard all attack venues that can affect the old DRS independently of our new method, as this would be out of this paper' scope, and we stress it is hard to construct a security proof when an attack aside from exhaustive search cannot be constructed.

4.1 Attack models

A malicious Bob One attack is to try to guess the triplets generated by Alice, as malicious Bob, by sending carefully crafted keys and signatures. While it is definitively an interesting idea, as long as Alice generates a different triplet for each public key (using a hash of P_{key} as a seed for example) and only answers $True$ or $False$ in the verification, we do not see any gain malicious Bob could have over a honest Bob.

A honest Bob, and a "fake Bob" Eve To the best of our knowledge, the only other attack venue is having a honest Bob, who is giving good signatures, and Eve, who has no knowledge of the secret key S_{key}, wanting to sign as well as Bob but could not in the existing DRS scheme. Let us suppose Eve knows that Alice is using our technique for signature verification, although assuming she has no knowledge of the triplets T and knows as much as Alice concerning Bob. Can Eve make Alice believe Eve is Bob? To this purpose, we assume Eve has to generate a false-positive from Alice's verification algorithm. As Alice can make the primes p_i and their quantity k as large as she wants, it seems unreasonable to assume Eve can randomly fall into the $\prod_{i=1}^{k} p_i^{-1}$ false-positive probability. Eve cannot resort either to a lattice-reduction technique on an easier lattice stemming from T if she has no knowledge of T. Furthermore, building a false-positive for the modified Freivalds' test is not enough: one has to guarantee the vector-signature s is short enough. It is then possible that the number of false-positives drastically decreases, which reinforces the security of our modification however counting the number of false-positives within a bound seems non-trivial. Therefore, we believe that for Eve to be successful she must at least recover T fully. *It is unclear if the knowledge of the primes p_i is enough for Eve to recover the associated vectors x_i. While this is a very obvious overexaggeration on Eve's attack capabilities, we will assume for a simpler analysis that guessing exactly all p_i is sufficient to trigger a false-positive on Alice's side.* We will now explain how to alleviate this (potential) issue.

4.2 How to choose the primes

In order to dissuade Eve from trying to guess the correct set of primes $T_p = (p_i)_{i\in[1,k]}$, we have to make sure the number of possibilities is large

enough. In that regard, *we are considering two objectives: one is to reduce the complexity of arithmetical operations used during the verification algorithm, and the other is to match a chosen level of security.* Which naturally brings us to a natural question: is it easier to trigger a random false-positive, i.e to try our luck with an attacker's success of $\prod_{i=1}^{k} p_i^{-1}$, or to guess T_p? As we will observe later, the set of combinations T_p is picked from is actually far below $\prod_{i=1}^{k} p_i$. We also choose primes to obtain efficient arithmetic. To deal with the second objective, we will just fix the level of security to match the same level of security the original DRS algorithm was aiming to achieve with lattices of dimension $\{1108, 1372, 1779\}$: the NIST security levels $\{3, 4, 5\}$ which is basically requiring $\{128, 192, 256\}$ bits of security. To reach that number, let us present how we determine the number of combinations available when choosing primes of a certain amount of bits. Suppose we have a set S of primes, and pick k primes from it which gives $\binom{S}{k}$ combinations to choose from. We now have to determine both S and k. To give an idea of the numbers required, we give table 1 and refer to a table available online [26] referencing the number of primes. While taking low-bits primes to minimize the amount of modular reductions allows for more efficient arithmetic (see "lazy reductions" in Seiler's work for NewHope [24]), we will very soon show that we have to combine multiple sets of large sizes of primes to achieve a reasonable amount of security. DRS fitted every computation within 64-bits for both speed and convenience, and ideally we should follow that philosophy. Our final choices are:

3 : 128-bits security: $k = 6$ with S the set of 28-bits primes
4 : 192-bits security: $k = 9$ with S the set of 27 and 28-bits primes
5 : 256-bits security: $k = 12$ with S the set of 24 to 28-bits primes

Table 1. Size of set S necessary to achieve $\binom{S}{k} > 2^b$

b \ k	5	6	7	8	9	10
128	132,496,421	7,910,346	1,080,111	-	-	-
192	-	-	610,573,333	63,155,327	10,957,838	2,727,426

b \ k	11	12	13
256	49,751,158	13,974,454	4,801,557

5 Implementation results

5.1 Time results on a basic implementation

To make a fair comparison, we first give the time given by the original algorithm (setup, sign and verify). Time is given as an average in milliseconds and computations were done using a Intel(R) Xeon(R) Gold 6128

CPU @ 3.40GHz processor using a non-optimized C implementation and re-using the code provided by the DRS submitters on their website (see table 2). We then showcase the base case where we do not take account of

Table 2. Average time (in ms) for the existing DRS scheme

Security \ Phase	Setup	Signature	Verification
128	67.5	1.495	0.89
192	102	2.46	1.68
256	162.9	3.82	3.50

the number of combinations and use just enough 32-bit integers to reach the product size needed, and compare them with our choices with smaller primes (28-bits or less) in a larger amount to reach the combination size needed (see table 3). Note that the generation of primes is not included, as we used MAGMA [6] to pick primes and write them into a header file used by our code before the compilation. Picking primes, however did not take any significant amount of time (almost always lower than $10ms$), and we used an external software to select them. Following this, we do not think reporting the time taken for the prime generation is very relevant, as the literature also points out it is on a much lower scale than a matrix-vector multiplication (for our sizes, see [16] and subsequent work on either heuristic or deterministic algorithms). We observe that the pre-

Table 3. Average time (in ms) for the precomputation/verification algorithms

Security \ Phase	128	192	256
Precompute (32-bits)	100.16	223.9	646.69
Verify (32-bits)	0.1328	0.2515	0.4297
Precompute (28-bits)	153.21	363.1	1005.1
Verify (28-bits)	0.1048	0.2006	0.3429

computation is heavier than the generation of the keys. Which is expected as we are dealing with multiple modular matrix-vectors multiplications, whereas the original DRS setup only had to deal with randomized vectors additions. The number of signatures generated per key to break even in time (*including precomputation*) compared with the old DRS is reached for 256-bits of security with 319 signatures (28-bits case) and 211 (32-bits case).

5.2 Memory storage

As we mentioned previously, Alice does not need to store Pk in our alternative scheme. Memory-wise, this showcases an obvious advantage for the verifier to require only a quasi-linear amount of memory in function of the dimension rather than a quadratic amount (i.e full public key). Here

in all 3 cases we store k prime integers of 28-bits, plus $2 * k$ vectors of dimension n containing 28-bits integers, thus the memory taken in bytes is $\lceil 28(k+2kn)/8 \rceil$ (see table 4). Another potential worry in term of mem-

Table 4. Memory storage in bytes for P_{key} and its footprint T

Security	128	192	256
P_k the public key	7,672,900	11,764,900	19,780,257
p_i, x_i, X_i	46,557	86,468	149,478

ory in our modified scheme is that the prime number generation might be taking a lot of memory for the verifier. However, after decades of research on prime number generation we do not believe this is the case [14].

6 Conclusion

In this paper we introduced a modification of Freivalds' algorithm to introduce a faster verification method to DRS. By introducing a precomputation step that is in the same order of magnitude as the setup in time, we gain a factor of almost 20 for the verification part while also heavily reducing its memory cost. This process is done while not modifying any information given by the signatory. Furthermore, more research should greatly improve this new work. **First**, we assumed almost "paranoiac" security requirements, thus a deeper analysis should improve efficiency. **Second**, we can make use of Residue Number Systems: stemming from [12] with several applications [2–5], finding large arithmetically efficient random groups is exactly what we need. **Third**, generalizing to all lattices and HNF keys. It needs the extra vector h, but any party can compute h in polynomial time with no security loss.

References

1. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: STOC '97. pp. 284–293. ACM (1997)
2. Bajard, J.C., Eynard, J., Merkiche, N.: Multi-fault attack detection for rns cryptographic architecture. IEEE 23nd Symposium on Computer Arithmetic (July 2016)
3. Bajard, J.C., Imbert, L.: A full rns implementation of rsa. IEEE Transactions on Computers **53**(6), 769–774 (2004)
4. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full rns variant of fv like somewhat homomorphic encryption schemes. In: SAC. pp. 423–442. Springer (2016)
5. Bajard, J.C., Plantard, T.: Rns bases and conversions. In: Optical Science and Technology, the SPIE 49th Annual Meeting. pp. 60–69 (2004)
6. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. **24**(3-4), 235–265 (1997)
7. Brualdi, R.A., Ryser, H.J.: Combinatorial matrix theory, vol. 39. Cambridge University Press (1991)
8. Dumas, J.G.: Proof-of-work certificates that can be efficiently computed in the cloud (invited talk). In: CASC 2018. pp. 1–17. Springer (2018)

9. Dumas, J.G., Zucca, V.: Prover efficient public verification of dense or sparse/structured matrix-vector multiplication. In: ACISP 2017. pp. 115–134. Springer (2017)
10. Freivalds, R.: Fast probabilistic algorithms. In: International Symposium on Mathematical Foundations of Computer Science. pp. 57–69. Springer (1979)
11. Gama, N., Izabachene, M., Nguyen, P.Q., Xie, X.: Structural lattice reduction: generalized worst-case to average-case reductions and homomorphic cryptosystems. In: EUROCRYPT 2016. pp. 528–558. Springer (2016)
12. Garner, H.L.: The residue number system. In: Papers presented at the the March 3-5, 1959, western joint computer conference. pp. 146–153. ACM (1959)
13. Grover, L.K.: A fast quantum mechanical algorithm for database search. arXiv preprint quant-ph/9605043 (1996)
14. Joye, M., Paillier, P.: Fast generation of prime numbers on portable devices: An update. In: CHES 2006. pp. 160–173. Springer (2006)
15. Kimbrel, T., Sinha, R.K.: A probabilistic algorithm for verifying matrix products using o(n2) time and log2(n)+ o(1) random bits. Information Processing Letters **45**(2), 107–110 (1993)
16. Maurer, U.M.: Fast generation of prime numbers and secure public-key cryptographic parameters. Journal of Cryptology **8**(3), 123–155 (1995)
17. Micciancio, D.: Improving lattice based cryptosystems using the hermite normal form. In: Cryptography and Lattices, pp. 126–145. Springer (2001)
18. Nguyen, P.Q., Regev, O.: Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. Journal of Cryptology **22**(2), 139–160 (2009)
19. NIST: Post-quantum cryptography standardization (2018), `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`
20. Pernet, C., Stein, W.: Fast computation of hermite normal forms of random integer matrices. Journal of Number Theory **130**(7), 1675–1683 (2010)
21. Plantard, T., Sipasseuth, A., Dumondelle, C., Susilo, W.: Drs : Diagonal dominant reduction for lattice-based signature. PQC Standardization Conference, Round 1 submissions (2018), `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/DRS.zip`
22. Plantard, T., Susilo, W., Win, K.T.: A digital signature scheme based on cvp max. In: PKC 2008. pp. 288–307. Springer (2008)
23. Regev, O.: New lattice-based cryptographic constructions. Journal of the ACM (JACM) **51**(6), 899–942 (2004)
24. Seiler, G.: Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography. Cryptology ePrint Archive, Report 2018/039 (2018)
25. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5) (1997)
26. e Silva, T.O.: Tables of values of pi(x) and of pi2(x). http://sweet.ua.pt/tos/primes.html (2018)
27. Sipasseuth, A., Plantard, T., Susilo, W.: Improving the security of the drs scheme with uniformly chosen random noise. In: ACISP 2019. pp. 119–137. Springer (2019)
28. Yu, Y., Ducas, L.: Learning strikes again: The case of the drs signature scheme. In: ASIACRYPT 2018. pp. 525–543. Springer (2018)