Masking, Sequences, and FALCON: A Theoretical Study on Masking Strategies Using Sequences for Non-Linear Operators in the FALCON Post-Quantum Signature*

Pierre-Augustin Berthet[1][0009−0005−5065−2730]✉

Univ. Grenoble Alpes, CEA-Leti, F-38054 Grenoble, France,
pierre-augustin.berthet@cea.fr

Abstract. Post-Quantum Cryptography is now in its deployment phase. Amongst the threats encountered in real-world applications is Side Channel Analysis, a cryptanalysis branch relying on the study of physical leakages from unsecured implementations. However, the FALCON post-quantum signature includes non-linear functions on real numbers, and applying the generic masking countermeasure to these functions has only been recently studied. In this work, we use convergent sequences to approximate the function and a minimax polynomial to compute the first term of the sequence. The method is applied to the computation of the inverse square root and the square root in FALCON, two operators for which masking has not been investigated before. A theoretical analysis of the security in the t-probing model using the NI criterion and its variants is proposed. Compared to the existing state-of-the-art using the same approach which covers the masked inversion for emulated floating-point implementation, this paper is generic and works with any representation and precision for real numbers.

Keywords: Post-Quantum Cryptography · FALCON · Side-Channel Analysis · Masking · Sequences · Arithmetisation · SNI

1 Introduction

Current asymmetric cryptosystems are mainly based on the assumption of the hardness of the Discrete Logarithm Problem (DLP). Solving this problem using a classical computer is not achievable in practical time (to the best of our knowledge). However, if a quantum computer is powerful enough, there is an algorithm to solve DLP instances in practical time, the Shor algorithm [30]. With the steady progress made in quantum computing in the last decade, the existence of such a computer in the near future has become highly likely, posing a serious threat to the security of communications.

* This work is supported by the French National Research Agency (ANR) via the grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

To address the issue, the National Institute of Standards and Technology (NIST) held a competition starting in 2016 to select the next Post-Quantum Cryptography (PQC) standards for asymmetric encryption, as well as for Digital Signature Algorithms (DSA). The first competition ended in 2025 with the selection of 2 Key Encapsulation Mechanisms (KEM), and 3 DSAs, including the signature FALCON [26].

As the initial PQC standards enter their deployment phase, it is essential to confront their design with the threats posed to real-world applications. Amongst them is Side-Channel Analysis (SCA). First published in 1996 by Kocher [22], SCA aims to exploit the physical impact a cryptosystem has on its environment and tries to correlate the observed leakage with the sensitive value computed or stored within the cryptosystem. Therefore, studying the resilience of the implementation of new primitives to SCA is paramount for their deployment in real-world applications.

The side-channel resilience of the FALCON post-quantum signature is an ongoing topic. Several attacks have been proposed against Cortex M4 implementations of the signature, targeting either the key generation procedure [27,28], or different steps in the signature procedure such as preimage computation [8,10,15,19,21] or the Gaussian sampler [15,23,24,31]. In terms of countermeasures to these attacks, many propositions were made [11,15,21,23,24,31] but were often tailored to address a specific leakage. A general consensus is that provable countermeasures, such as masking, should be studied.

Related Work The main challenge of masking FALCON is its use of floating-point arithmetic. Floating-point computations require rounding which can create and propagate errors, and as such, a naive masking approach is not recommended. Existing attempts involve either a complete modification of the primitive into a more masking-friendly design, such as in Mitaka [13], removing the need for floating-point arithmetic such as in [12], or changing the representation to less error-prone fixed-point arithmetic [4]. Several papers [2,3,7,9,20] tackle the challenge by masking parts of the less performant emulated floating-point version of FALCON, with the benefit that their approach does not exacerbate error propagation as it uses integer and boolean maskings. However, these works only address some functions and operators used in FALCON. To assemble a complete masked implementation of emulated floating-point FALCON, it is necessary to investigate masking either the inverse square root or the square root.

Contributions In this work, we detail a generic approach to mask the inverse square root and the square root. More precisely:

– We adapt the methodology introduced in [3] for the masked inversion to these two functions. We analyse possible choices in terms of minimax polynomial order and number of iterations for varying precisions in terms of algorithmic cost.

– We rely on the MIMO-SNI proof framework introduced in [5] to provide an analysis of the formal security requirements of the proposed designs in the t-probing model, trying to relax them as much as possible to reduce the impact that security will have on performance. This includes an analysis of the security requirements of the Horner method due to the use of minimax polynomials. We also propose a more relaxed security requirement for the subgadgets of the masked inversion in [3], hinting at possible improvements in their design.

The paper is structured as follows. Section 2 introduces the necessary background for FALCON, masking, and convergent sequences. The strategy that combines convergent sequences and masking is discussed in Section 3. Several choices for the minimax order and number of iterations are presented in Section 4. Finally, the security requirements for the subgadgets used in the methodology are discussed in Section 5.

2 Background

2.1 FALCON

FALCON [26] is a lattice-based post-quantum signature selected by the National Institute of Standards and Technology as a standard for post-quantum cryptography under the name FN-DSA. It relies on the GPV [14] framework, applied to NTRU lattices [17]. As a signature, it is composed of three procedures: the key generation, the signature itself, and the verification. FALCON also has two different parameter sets, FALCON-512 and FALCON-1024, partially detailed in Table 1. In this work, only the FALCON-512 parameter set is used; however, the methodology can be applied to FALCON-1024 as well.

Table 1. Partial parameter sets for FALCON [26]

	Target NIST Level	Standard deviation σ	σ_{min}	σ_{max}
FALCON-512	I	165.736 617 183	1.277 833 697	1.8205
FALCON-1024	V	168.388 571 447	1.298 280 334	1.8205

The inverse square root function $\frac{1}{\sqrt{x}}$ is used during the key generation procedure[1]. It is used as a normalisation step to adjust the Gaussian coefficients stored within the "FALCON-tree". The equation, from [26] page 33 Algorithm 4 line 7, updates a sensitive value denoted $leaf.value$ (σ is a public constant):

$$leaf.value = \frac{\sigma}{\sqrt{leaf.value}}. \tag{1}$$

[1] See [26], FALCON, supporting documentation, page 33 Algorithm 4, or Appendix A.1 Algorithm 1

No further computation is performed on the FALCON-tree during the key generation. The values stored in it become part of the secret key, which is then used in the signature procedure in a Gaussian sampler, specifically in the `SamplerZ` subroutine. Two equations, from [26] page 43 Algorithm 15 at lines 2 and 7 (or in Appendix A.2 Algorithm 2), apply the inverse function to a sensitive value denoted σ':

$$ccs \leftarrow \sigma_{min}/\sigma' \text{ (Line 2).} \tag{2}$$

$$x \leftarrow \frac{(z-r)^2}{2(\sigma')^2} - \frac{z_0^2}{2\sigma_{max}^2} \text{ (Line 7).} \tag{3}$$

An interesting point is that the σ' value from Equations 2 and 3 is actually the only documented use of the value $leaf.value$ from Equation 1 outside of the key generation. As a consequence, an implementation trick is to modify Equation 1 to store the value inverted from the start (see Equation 4), and replace the inversion in Equations 2 and 3 by a multiplication. However, this trick requires a square root function.

$$leaf.value = \frac{\sqrt{leaf.value}}{\sigma}. \tag{4}$$

At the time of writing this work, a draft of the standard has yet to be published. In particular, FALCON was originally designed to be implemented using double precision floating-point arithmetic (53 bits of precision, which is equivalent to an error of at most $\approx 1.12\mathtt{e}-16$). Currently, there are discussions of switching to fixed-point arithmetic [4]. In addition, recent work by Halmans et al. [16] highlighted security issues due to double precision and proposed a triple-word implementation of FALCON, with 72 bits of precision, thus an error of at most $\approx 2.12\mathtt{e}{-22}$. As the results presented in this paper aim to be generic, that is, independent of the representation used, we are confident that they can be applied to the final version of FN-DSA.

2.2 Masking

A well-studied generic countermeasure to side-channel opponents is masking [6]. Instead of performing the computation on a sensitive value, a *sharing* of the secret is used. A sharing is made up of random *shares* that are processed independently. To recover the final result of the computation, the shares are reassembled to form the secret result. A basic example of masking is Boolean masking, applied to a secret bit x:

$$\mathtt{Mask}(x) = (r_1, r_2, \ldots, r_n, x \oplus \bigoplus_{i=1}^{n} r_i),\ r_i \text{ being random bits.} \tag{5}$$

As the r_i are random and redrawn each time the implementation runs, the side-channel attacker must recover the $n+1$ shares each time, thus significantly increasing the cost and complexity of the attack.

A key to the popularity of masking is the use of formal models and proofs to validate its security. In this paper, we rely on the t-probing threshold model of Ishai, Sahai and Wagner [18]. Conceptually, this model estimates the number of probes required to break the countermeasure, that is, the number of internal values measured by side-channel analysis. A masked implementation of a function, also called a *gadget*, is said to be t-probing secure if the minimum number of probes required is $t + 1$.

In order to prove that large gadgets are t-probing secure, composition criteria are used. In this work, the criteria used are the Non-Interference (NI) criterion and its Strong variant (SNI) [1], which both imply t-probing security. As the composition of NI and SNI gadgets is not trivially NI or SNI, we rely on the graphical approach introduced by Cassiers and Standaert [5] and the stricter criterion they introduced, called Multiple-Input Multiple Output SNI (MIMO-SNI). We now describe their method. A composite gadget, i.e. a gadget composed of several subgadgets, can be represented by a Directed Acyclic Graph (DAG). In this graph, each subgadget constitutes a vertex, and an edge only connects two vertices if the subgadget vertex it ends in takes as one of its inputs an output of the subgadget vertex the edge starts at. The DAG obeys some rules: Each subgadget vertex must have the NI level of security. If an output of a subgadget is used several times throughout the gadget, it is connected to a *Split* vertex which acts as a duplication function with the NI security level. An SNI subgadget vertex is represented as an NI vertex with its output connected to an SNI refresh vertex. This SNI refresh vertex is then connected to the subgadget taking the original vertex output as one of its inputs.

From the DAG, we can derive the Simplified Computation Graph (SCG) which is used to prove the security level of the gadget. The SCG is obtained from the DAG by removing all SNI refresh vertices and their connected edges. Then, one can use the graph properties of the SCG to verify the security level of the gadget:

Proposition 1 (MIMO-SNI security using the SCG[5]). *A composite gadget G is MIMO-SNI if its SCG satisfies the four following conditions. (i) G is Single-Path NI Built (SP-NIB), i.e. for any two vertices u and v in its SCG, there is at most one path from u to v. (ii) For any pair of output vertices $u_1,\ u_2$ in the SCG, there is no vertex v such that there is a path from v to u_1 and a path from v to u_2. (iii) For any pair of input vertices $u_1,\ u_2$ in the SCG, there is no vertex v such that there is a path from u_1 to v and a path from u_2 to v. (iv)[2] There is no path from any input vertex to an output vertex in the SCG.*

Proposition 1 can be used to verify other levels of security than MIMO-SNI:

- If the SCG verifies the first condition (i), it is trivially NI secure, as an SP-NIB SCG does not contain any interference, that is, a situation where two vertices in the SCG are connected through two or more paths.
- If the SCG verifies the second condition (ii), the gadget is said to be Multiple-Output (MO). A Single Output (SO) trivially verifies condition (ii).

– If the SCG verifies the third condition (iii), the gadget is said to be Multiple-Input (MI). If a gadget has a Single Input (SI), it is trivially MI.
– Finally, the fourth condition is mandatory for Strong security. For instance, a gadget that verifies only the first and fourth conditions is SNI secure.

Although the composition of SNI or NI gadgets might not be NI secure, the composition of SISO, MISO, SIMO, and MIMO gadgets is. This can be verified easily by studying the possible interference cases when composing gadgets: If the composition of gadgets generates an interference, it implies that the interference starts within a subgadget and ends within another. The MO security ensures that the subgadget cannot be a starting point for an interference, the MI security implies that the subgadget cannot be an ending point for an interference. Thus, the composition of MIMO-NI subgadgets is MIMO-NI itself, and thus NI. Similarly, the composition of MIMO-SNI gadgets is trivially MIMO-SNI and we have the following relation:

$$\text{MIMO-SNI} \implies \text{SNI} \implies \text{NI} \implies t\text{-probing secure.}$$

The stricter the criterion used, the worse the performance: Strong security often requires sampling fresh randomness and injecting it into the sharing, via the application of an SNI refresh, that is, the identity function with SNI security (hence the representation of SNI vertices in the DAG as NI vertices followed by an SNI refresh). Similarly, MI (resp. MO) security can be achieved naively by applying the same costly SNI refresh to input (resp. output) sharings. As a consequence, in Section 5, we reduce the security requirements of each subgadget as much as possible towards NI security, while ensuring that the gadget remains SISO-SNI secure.

2.3 Real Function and Convergent Sequences

In order to compute a non-linear function on real numbers, one can use the Newton-Raphson or the Halley root finding methods. Let f be a 3 times continuously differentiable function, such that the value r is a root of f but not of its derivative f'. Then, the Newton-Raphson method stipulates that the sequence in Equation 6 converges quadratically to r if the first term of the sequence is close enough to r.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{6}$$

Similarly, the Halley method stipulates that the sequence in Equation 7 converges cubically to r if the first term of the sequence is close enough to r.

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{[f'(x_n)]^2 - \frac{1}{2}f(x_n)f''(x_n)}. \tag{7}$$

These root finding methods can be exploited to approximate functions. For example, to approximate the computation of $\frac{1}{a}, a \in \mathbb{R}^*$, we consider the function $f(x) = \frac{1}{x} - a$. It is trivial that $f(\frac{1}{a}) = 0$ and thus that $\frac{1}{a}$ is a root of f.

3 Masking Using Sequences

3.1 Overall Idea

There are several approaches to masking non-linear operators. The first is to take the unmasked computation of the operator and translate each internal step in its masked equivalent. However, this method might imply switching between different representations and/or between arithmetic and boolean operations. These switches are often costly and might not be available for certain types of masking. Another approach revolves around the arithmetisation of the computations.

Arithmetisation is the process of transforming the computations of a non-linear operator or function into a series of addition, multiplication, addition by a constant and scalar multiplication. These four operators are usually well-studied in terms of implementation and masking countermeasures. Therefore, if a non-linear function can be arithmetised, it can be masked most of the time. Arithmetisation also has the benefit of being generic compared to the "translation" strategy. Thus, the strategies presented in this work are completely independent of the representation or masking used.

Arithmetisation can be done with several different mathematical tools, such as polynomial approximation or Lagrangian interpolation. In this paper, we also use convergent sequences. Although a convergent sequence can itself contain non-linear operators, it is possible to find some sequences which only relies on the four "maskable" operators mentioned before.

3.2 Masked Approximation of the First Term

When using a sequence computed thanks to the Newton-Raphson or the Halley method, a requirement for the sequence to converge is that its first term is close enough to the final result. In an unmasked scenario, this is done by using comparisons and tables. However, this approach is not suitable for a masked implementation. Therefore, we compute the first term with approximation polynomials. Approximation polynomials are easy to mask and can be evaluated with the Horner method (or the Paterson-Stockmeyer method [25] if the cost of multiplication significantly outweighs the cost of addition and scalar multiplication).

Although a simple Lagrangian interpolation might suffice, another solution has been proposed by Berthet and Tavernier [3]. In their work on a masked floating-point inverse for FALCON, they approximate the first term of the sequence with a minimax polynomial. A minimax polynomial provides the best approximation with respect to lowering the maximum difference between the approximated function and its polynomial approximation. The lower this maximum, the smaller the number of iterations to reach the desired precision, thus improving the performance of the gadget. Minimax polynomials are computed[2] using the Remez algorithm [29].

[2] We used the Python implementation of Chun Wang and Wei Jiatong https://github.com/chunwangpro/Remez_Python. Results are available in the Appendix Section B.

Minimax polynomials, similarly to Lagrangian interpolation polynomials, can be of varying orders. Usually, the higher the degree of the minimax polynomial, the better the initial approximation. However, a higher degree also means higher complexity. In Section 4, the algorithmic complexity in terms of order of the minimax polynomial, and number of iterations of the sequence, is detailed. The complexity formulae differentiate between operators applied to two sensitive values (`Add` for addition, `Mul` for multiplication) and applied to a sensitive value and a public constant (`AddCst` for addition, `MulCst` for multiplication). In the case of multiplication by a constant, no difference is considered between the case where the constant is a power of two and the case where it is not. In practice, some representations allow faster computations when the constant is a power of two.

4 Precision

In this section, we follow for each operator studied the same structure: the equation of the sequence is given, as well as its algorithmic complexity. Then, a table is provided for varying orders of the minimax polynomial used to approximate the first term, and numbers of iterations of the sequence. The results in the table are always rounded up. All the results presented in this manuscript use the FALCON-512 parameter set, but the methodology remains the same for the FALCON-1024 parameter set.

Recommendations are also provided in terms of the minimax polynomial order and the number of iterations of the sequence. The baseline for the recommendations is to consider, for a specific precision, the lowest amount of iterations required per minimax polynomial order to achieve the desired precision. We then compare the results on the basis of the number of calls to each subgadget separately and take the choice with the lowest complexity. As the study is generic and thus agnostic to the complexity differences between subgadgets, if different choices achieve the lowest score in different subgadgets, each choice is provided. It will be up to the implementer to choose the correct choice based on the complexity differences between its subgadget implementations.

4.1 Cost of the First Term Approximation

The minimax polynomial is evaluated using the Horner method. As a consequence, the algorithmic cost of evaluating an order o minimax polynomial is the following:

$$\texttt{Minimax}(o) = o * \texttt{AddCst} + (o-1) * \texttt{Mul} + \texttt{MulCst}. \tag{8}$$

As the precision of the first term affects the number of iterations necessary to obtain the desired precision for the sequence, we fix the precision of the minimax polynomials to double precision to reduce the number of unknowns in the analysis. The precision of the sequences is evaluated using the `mpmath` library in Python.

4.2 Inverse Square Root

In this subsection, the inverse square root is computed using the Halley method. To compute the inverse square root of a value a, the Halley method uses the following sequence:

$$\forall n \in \mathbb{N},\ z_n = a * x_n^2,\ x_{n+1} = \frac{x_n}{8}\left(15 - z_n * (10 - 3 * z_n)\right). \tag{9}$$

It converges cubically if its first term x_0 is a good approximation of $1/\sqrt{a}$. The complexity of computing this sequence for i iterations and a minimax polynomial of order o is:

$$\texttt{HalleyInvSqrt}(o, i) = (2i + o)\texttt{AddCst} + (4i + o - 1)\texttt{Mul} + (2i + 1)\texttt{MulCst}. \tag{10}$$

Table 2 gives the precision of applying the sequence to compute the inverse square root in FALCON for different values of i and o. The interval in which the sequences are computed is given by $\left[\left(\frac{\sigma}{\sigma_{max}}\right)^2, \left(\frac{\sigma}{\sigma_{min}}\right)^2\right]$. However, this interval has high bounds, where the inverse square root function "flattens". As a consequence, the Remez algorithm cannot compute second order and above minimax polynomials with the imposed double precision.

Table 2. Precision of the Halley sequence for the inverse square root depending on the number of iterations (columns) and the order of the minimax polynomial (rows)

	0	1	2	3	4
1	2.1e−4	4.0e−7	2.6e−15	6.9e−40	1.4e−113

Taking the full Equation 1, we can directly integrate the multiplication by σ to reduce the limits of the interval on which the sequences are computed:

$$leaf.value = \frac{\sigma}{\sqrt{leaf.value}} = \frac{1}{\sqrt{\frac{leaf.value}{\sigma^2}}} = \frac{1}{\sqrt{y}} \text{ with } y \in \left[\frac{1}{\sigma_{max}^2}, \frac{1}{\sigma_{min}^2}\right]. \tag{11}$$

As a consequence, the Remez algorithm with double precision can compute multiple orders of minimax polynomials, and we can once again study the trade-off possibilities between minimax order and number of iterations. Table 3 provides the precision of computing the inverse square root multiplied by σ.

Recommended Choices According to Tables 2 and 3, the following choices have the lowest number of calls to each subgadget (considered separately):
For double precision (error of at most $\approx 1.12\texttt{e} - 16$),

$$\texttt{HalleyInvSqrt}(o = 2, i = 2) = 6\texttt{AddCst} + 9\texttt{Mul} + 5\texttt{MulCst}. \tag{12}$$

Table 3. Precision of the Halley sequence for the inverse square root depending on the number of iterations (columns) and the order of the minimax polynomial (rows), with the multiplication by σ integrated.

	0	1	2	3	4
1	0.037	6.5e−5	4.1e−13	1.1e−37	1.8e−111
2	0.0070	4.3e−7	1.1e−19	1.5e−57	3.8e−171
3	0.0012	2.5e−9	1.6e−26	5.8e−78	2.7e−232

For triple-word precision (error of at most $\approx 2.12\texttt{e}{-22}$),

$$\texttt{HalleyInvSqrt}(o=3,i=2) = 7\texttt{AddCst} + 10\texttt{Mul} + 5\texttt{MulCst}. \tag{13}$$

Alternatively, one could use the Newton-Raphson method. However, the experimental results[3] show that there is no clear advantage in doing so. Despite having a lower complexity per iteration, the Newton-Raphson method has a slower convergence speed, thus requiring more iterations than Halley.

4.3 Square Root

A well-known methodology to compute the square root of a real number is the Babylonian or Heron method. To compute the square root of a value a, the Heron method uses the following sequence:

$$\forall n \in \mathbb{N}, x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right). \tag{14}$$

It converges quadratically if its first term x_0 is a good approximation of $\sqrt{a}$. The complexity of computing the sequence for i iterations and a minimax polynomial of order o is given by:

$$\texttt{Sqrt}(o,i) = i*(\texttt{Add}+\texttt{Div}+\texttt{MulCst2})+o*\texttt{AddCst}+(o-1)*\texttt{Mul}+\texttt{MulCst}. \tag{15}$$

Table 4 provides the precision of applying the sequence to compute the square root in FALCON for different values of i and o. However, as the interval on which the sequence is computed has high bounds, the Remez algorithm cannot compute third order and above minimax polynomials with double precision.

Table 4. Precision of the Heron sequence depending on the number of iterations (columns) and the order of the minimax polynomial (rows)

	0	1	2	3	4	5
1	0.86	0.0039	8.1e−8	3.7e−17	7.2e−36	2.8e−73
2	0.10	4.2e−5	7.2e−12	2.2e−25	1.9e−52	1.5e−106

[3] Appendix Section C.3

We rely on a similar trick to the one used for the inverse square root. To reduce the interval in which the sequence is computed, the multiplication by $\frac{1}{\sigma}$ in Equation 4 is performed before. Table 5 provides the precision of computing the square root and division by σ directly. The first two rows (minimax polynomials of first and second order) of Tables 4 and 5 validate the efficiency of the trick in the case of double precision.

Table 5. Precision of the Heron sequence depending on the number of iterations (columns) and the order of the minimax polynomial (rows), with the division by σ integrated

	0	1	2	3	4	5
1	0.0052	2.4e−5	4.9e−10	2.2e−19	4.4e−38	1.7e−75
2	5.7e−4	2.2e−7	3.3e−14	7.3e−28	3.6e−55	8.9e−110
3	7.0e-5	3.3e−9	7.0e−18	3.3e−35	7.0e−70	3.3e−139
4	8.9e−6	5.2e−11	1.8e−21	2.4e−42	5.0e−84	2.2e−167
5	1.2e−6	8.9e−13	6.3e−25	3.5e−49	1.1e−97	9.6e−195

Recommended Choices According to Table 5, the choices with the lowest number of calls to each subgadget (considered separately) are as follows:
For double precision (error of at most $\approx 1.12\texttt{e} - 16$):

$$\texttt{Sqrt}(o = 1, i = 3) = 3\texttt{Add} + 3\texttt{Div} + 4\texttt{MulCst} + \texttt{AddCst}.$$
$$\texttt{Sqrt}(o = 3, i = 2) = 2\texttt{Add} + 2\texttt{Div} + 3\texttt{MulCst} + \texttt{AddCst} + 2\texttt{Mul}.$$

For triple-word precision (error of at most $\approx 2.12\texttt{e}{-22}$):

$$\texttt{Sqrt}(o = 1, i = 4) = 4\texttt{Add} + 4\texttt{Div} + 5\texttt{MulCst} + \texttt{AddCst}.$$
$$\texttt{Sqrt}(o = 2, i = 3) = 3\texttt{Add} + 3\texttt{Div} + 4\texttt{MulCst} + 2\texttt{AddCst} + \texttt{Mul}.$$
$$\texttt{Sqrt}(o = 5, i = 2) = 2\texttt{Add} + 2\texttt{Div} + 3\texttt{MulCst} + 5\texttt{AddCst} + 4\texttt{Mul}.$$

No Division Square Root The Heron method requires a division, which is non-linear. In order to have a fully arithmetised calculation of the square root, there is an alternative to the Heron method. One can observe that

$$\forall x \in \mathbb{R}, \sqrt{x} = x * \frac{1}{\sqrt{x}}. \tag{16}$$

Using Equation 16 to compute the square root and the sequences used to compute the inverse square root, it is possible to avoid masking the division. However, if a masked division is available, then a comparison with the complexities of the optimal choices for the inverse square root must be performed. It must be taken into account that using the inverse square root implies an extra `Mul`.

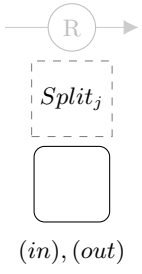
5 Security in the SNI model

One of the key aspects of the masking countermeasure is the use of formal proofs to verify the security of the gadgets. In this section, we discuss the requirements in terms of the level of security for the gadgets used in the sequence as well as in the approximation of the first term. The criteria used are the Strong Non Interference (SNI) of Barthe et al. [1] and its stricter variation, the Multiple-Input Multiple-Output SNI of Cassiers and Standaert [5]. Other criteria exist, such as the Probe Isolating Non Interfering (PINI) criterion of Cassiers and Standaert [5]. It has a more straightforward composition rule and only requires that every part of the gadget be PINI itself. The strategy of Section 3 and the results of Section 4 can be applied to a PINI implementation. This section focuses on composability for NI or SNI implementation.

It is also important to note that we consider in this section the sharing of a sensitive value in FALCON to be "monolithic", i.e. that only one sharing is required to represent the sensitive value in the masked domain for the considered representation of real numbers. As a consequence, evaluating a polynomial for a sensitive value, or computing its inverse, or its square root, results in a SISO gadget, with a target level of security of SISO-NI or SISO-SNI. If several sharings are required to mask one sensitive value, as could be the case in a triple-word implementation like [16], then the target level of security should be MIMO-SNI or MIMO-NI.

In this work, SCG are provided with the following legend (see Table 6):

Table 6. Legend for the Simplified Computation Graph (SCG)



: *t-SNI* Refresh vertex represented as a removed edge.

: $Split_j$ vertex (has j outgoing edges).

: *t-NI* vertex.

: Input, output vertices.

5.1 Minimax Polynomial

Minimax polynomials can be evaluated using the Horner method. Thus, they alternate between masked additions and masked multiplications. However, the input sharing on which the polynomial is evaluated serves as an input sharing to each multiplication used in the polynomial, leading to possible interferences. We start by studying the case of a degree one polynomial, and then build the

remainder of the evaluation gadget for higher degrees. In this work, we choose the target level of security for the first term evaluation to include the Strong property (at least SNI).

For a degree one polynomial P, the equation is as follows :

$$P(X) = p_0 + X * p_1, \text{ with } p_0, p_1 \text{ two public real constants.}$$

As a consequence, the input sharing first passes through a masked multiplication with a public constant, and then an addition with a public constant. Therefore, both gadgets can be considered as SISO gadgets, and their security requirements are relaxed to SISO-SNI. These requirements can be reduced to SISO-NI, as the serial composition of two SISO-NI gadgets is SISO-NI itself. However, as we aim for SISO-SNI security for the gadget, at least one of the two gadgets has to remain SISO-SNI, or an SNI Refresh, that is, an SNI identity function, must be applied to the output sharing.

We now consider the evaluation of a degree k polynomial P such that $P(X) = \sum_{i=0}^{k} p_i X^i$ for $k > 1$ an integer. Following the Horner method, there exists a polynomial Q of degree $k-1$ such that $Q(X) = \sum_{i=0}^{k-1} p_{i+1} X^i$ and:

$$P(X) = p_0 + X * Q(X), \text{ with } p_0 \text{ a public real constant.}$$

As a consequence, the input sharing passes through the masked evaluation of Q but also serves as input to a masked multiplication gadget, the other input being the output of the masked evaluation of Q. The output sharing of this multiplication is then taken as input in a masked addition with the public constant p_0 to produce the final output sharing of the masked evaluation of the polynomial P. A graphical representation of the gadget is available in Figure 1.

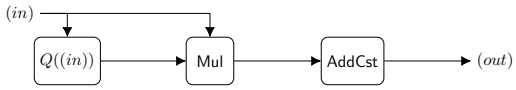


Fig. 1. Diagram for the masked evaluation of a degree k polynomial P

The evaluation of Q and the addition with p_0 can both be considered as SISO gadgets. However, there is a risk of interference as the multiplication takes as input two sensitive sharings which can both be connected to the input sharing.

The evaluation of Q is chosen to be SISO-SNI secure. As a consequence, in the SCG representation of P, there is no more path from the input sharing to the multiplication passing by the evaluation of Q. As the addition subgadget is SISO, the entire SCG of the evaluation of P is SP-NIB. To achieve Strong security, however, there must be no path from the input sharing to the output sharing. Two options are available: either the addition by a constant must be

SISO-SNI, or the multiplication must be SNI. An example of the first option is given in Figure 2. There, the SCG verifies conditions (i) and (iv) of Proposition 1, and the evaluation of P is therefore SISO-SNI.

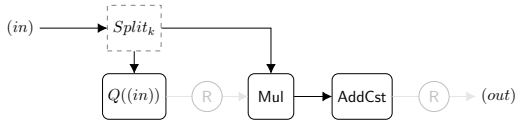


Fig. 2. SCG of the evaluation of $P(X) = p_0 + X * Q(X)$ if the evaluation of $Q(X)$ and the addition by a constant are SISO-SNI

5.2 Inverse Square Root

The calculation of z_n in Equation 9 generates a possible interference point on its own due to the squaring of x_n. A dedicated SISO-SNI squaring gadget must be developed, or the calculation must be performed as $(a * x_n) * x_n$ using an SNI secure multiplication. To avoid developing an additional subgadget, we recommend the latter solution. Thus, this solution is already envisioned in the graphical representation of the inverse square root computation in Figure 3.

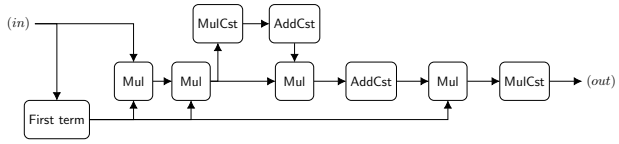


Fig. 3. Diagram of the sequence for the Halley method to compute the inverse square root (one iteration)

Several other interferences are also visible in Figure 3. However, setting the security requirement of `Mul` to be SNI secure, and taking into account that the first term evaluation is set to SISO-SNI, the only remaining interference concerns the central part of the diagram. Even if the multiplication is SNI secure, the output of the calculation of z_n is connected to the next multiplication through two possible paths, a direct path and one through a multiplication by a constant `MulCst` followed by an addition by a constant `AddCst`. As a consequence, it is mandatory that one of these two subgadgets has SNI security to ensure that the SCG is SP-NIB, while the other can be NI secure. This echoes the security requirements of the first term evaluation. In the SCG of the inverse square root

depicted in Figure 4, the choice is to set `MulCst` to be SNI secure. The SCG verifies conditions (i) and (iv) from Proposition 1.

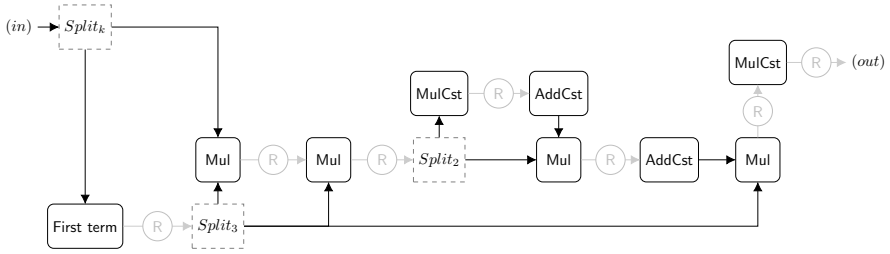


Fig. 4. SCG of the inverse square root using the Halley method and with SNI requirements for all types of multiplications.

5.3 Square Root

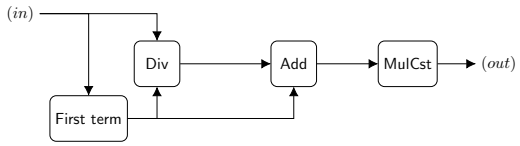


Fig. 5. Diagram of the sequence for the Heron method to compute the square root (one iteration)

A graphical diagram of the sequence is provided in Figure 5. It is apparent that the division and addition used in the sequence are both potential interference points. Thus, their initial security levels are set to MISO-SNI. Multiplication by $\frac{1}{2}$ can be seen as an SISO gadget. We also chose minimax polynomial evaluation to be SISO-SNI secure. It is important to note that the addition used here can differ from the one used in the minimax polynomial evaluation: in the latter constant public values are added to a sensitive sharing. In the sequence, the addition is performed between two sensitive sharings.
The security levels of addition and multiplication by $\frac{1}{2}$ can be reduced to the NI security level as long as the division remains at least SNI (see the SCG in Figure 6). If the division is not SNI, then there are two paths from the split output of the first term evaluation to the addition (one direct path and one through the division). Thus, the division must be SNI. A consequence of having

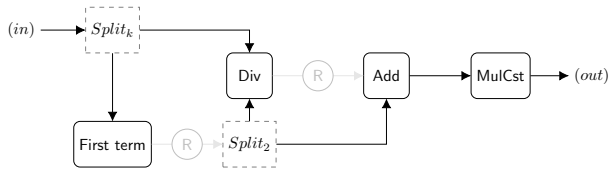


Fig. 6. SCG of the square root sequence with one iteration and with SNI requirements for the division

both the evaluation and the division be SNI is that there is no path from the input of the gadget to the output of the multiplication by $\frac{1}{2}$. The sequence is thus SISO-SNI. Similar reasoning is used to recursively prove that the sequence remains SISO-SNI even for a higher number of iterations as long as the division remains SNI.

5.4 Inversion

Inversion was already covered in the state-of-the-art [3] in the case of floating-point representation and thus double precision. To compute the inverse of a value a, they used the Newton-Raphson method with the following sequence:

$$\forall n \in \mathbb{N}, x_{n+1} = x_n * (2 - a * x_n). \tag{17}$$

However, the authors used the masked addition and multiplication gadgets developed by Chen and Chen [7], which are both SNI secure, and did not develop specific gadgets for the addition by a public constant and for scalar multiplication, nor tried to reduce the security of Chen and Chen's [7] gadgets. As a consequence, we propose a reduction in security requirements to compute the inversion while targeting the same security as [3], that is, SISO-SNI.

There are two possibilities. Reduce either the multiplication, or the addition by a constant, to the NI security level while keeping the other at the SNI level.

- If the multiplication remains SNI, then there is no path from the first multiplication in an iteration to the second multiplication in the same iteration. Also, there is no path between each iteration. This eliminates both possible interferences, and the security of the addition by a constant can be reduced to NI.
- If the addition by a constant is SNI, then there is no path between the first and the second multiplication within an iteration. This also ensures that there is only one path (the direct one) between the input of the gadget and the first multiplication in each iteration. The SCG of this scenario is provided in Figure 7. Conditions (i) and (iv) of Proposition 1 are verified, and thus the gadget is SISO-SNI.

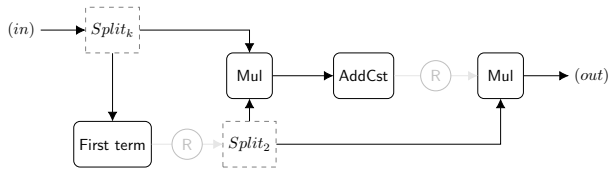


Fig. 7. SCG of the inversion for one iteration if addition by a public constant is SISO-SNI

Both cases are compatible with the SISO-SNI requirements for the first term approximation.

Taking into account the order and number of iterations selected in [3], their design requires 13 SNI gadgets ($5\texttt{Add} + 8\texttt{Mul}$). In our proposal, the most efficient case is to reduce `Mul` and `MulCst` to NI security, implying a design with 5 SNI gadgets and 8 NI ones. The other case aligns with the security requirements for the inverse square root, resulting in a design with 8 SNI gadgets and 5 NI ones. This is especially relevant if the trick from Equation 4 is not used, as the inversion is then part of a sampler called several thousand times during the signature. Thus, we anticipate that both proposed reductions will have a positive impact on the signature performance.

5.5 Recommendations

There are two scenarios in FALCON regarding the sequences presented in this paper: either the implementation sticks to the high-level description of the key generation and the signature, and thus uses Equations 1, 2, and 3, or the square root trick is applied as in Equation 4. In the second case, we consider that a masked division is available and that the Heron method is more efficient than using the inverse square root trick. In addition, an inversion is used in key generation in both cases.

If the inverse square root is used, the recommendations are to have NI security for `AddCst` and SNI security for `Mul` and `MulCst`. If the square root is used, the recommendations are to have NI security for `Add`, SNI security for `Div`, and either NI security for `MulCst` and `Mul`, SNI for `AddCst`, or SNI security for `MulCst` and `Mul`, NI for `AddCst`.

6 Conclusion

In this work, we study a masking strategy for two non-linear functions on real numbers, that is, the square root and its inverse, in the specific case of the FALCON post-quantum signature. To do so, we adapt the methodology introduced by [3] that uses convergent sequences and minimax polynomials to approximate

the first sequence term. For each function, recommendations regarding minimax orders and number of iterations for varying precision levels are proposed. In an other section, the formal security requirements for the different subgadgets used in the functions are analysed. Proposals are made regarding their security using the Non-Interfering (NI) criterion and its stronger variants, with the aim to reduce the requirements as much as possible. We extended our analysis to the masked inversion proposed in [3] and provide a relaxation of their security requirements, hinting at possible improvements in their design.

The square root and its inverse were the last required functions with no prior art on how to mask them in the context of the FALCON signature. Thus, a natural continuity of this work will be its application for different masked implementations of FALCON, whether they will use emulated floating-point, fixed-point, or triple-word representations of real numbers. The methodology presented in this paper, especially the formal security analysis, can also be used to address similar challenges outside of the FALCON context, such as side-channel secure environments for Artificial Intelligence computations using square root or inverse square root.

References

1. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security. pp. 116–129. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978427
2. Berthet, P.A., Paillet, J., Tavernier, C., Bossuet, L., Colombier, B.: Masked computation of the floor function and its application to the FALCON signature. IACR Communications in Cryptology (CiC) **1**(4), 9 (2024). https://doi.org/10.62056/ay73zl7s
3. Berthet, P.A., Tavernier, C.: Improving the masked division for the FALCON signature. Cryptology ePrint Archive, Report 2025/628, Version 20250708:141203 (2025), https://eprint.iacr.org/archive/2025/628/20250708:141203
4. Braga, D.D.A., Fouque, P., Lachguel, B., Prest, T.: Toward a secure fixed-point implementation of the falcon signature scheme. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part III. Lecture Notes in Computer Science, vol. 16802, pp. 166–197. Springer (2026). https://doi.org/10.1007/978-3-032-35377-1_6
5. Cassiers, G., Standaert, F.: Trivially and efficiently composing masked gadgets with probe isolating non-interference. IEEE Trans. Inf. Forensics Secur. **15**, 2542–2555 (2020). https://doi.org/10.1109/TIFS.2020.2971153
6. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology – CRYPTO'99. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_26
7. Chen, K.Y., Chen, J.P.: Masking floating-point number multiplication and addition of falcon first- and higher-order implementations and evaluations. IACR

Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 276–303 (2024). https://doi.org/10.46586/tches.v2024.i2.276-303

8. Chen, K.Y., Ching, M.Q., Chen, J.P., Yang, B.Y.: When masking multiplication isn't enough: Exploiting floating-point leakage in falcon's pre-image computation. IACR Transactions on Cryptographic Hardware and Embedded Systems **2026**(2), 686–708 (2026). https://doi.org/10.46586/tches.v2026.i2.686-708
9. Chen, S., Ming, J., Liu, Y., Gao, Y., Zhou, Y.: Masked Gadgets for Integer-Floating-Point Conversion with Applications to Falcon. In: 2025 IEEE 43rd International Conference on Computer Design (ICCD). pp. 507–514 (Nov 2025). https://doi.org/10.1109/ICCD65941.2025.00078, iSSN: 2576-6996
10. Chen, S., Shao, H., Ming, J., Liu, Y., Gao, Y., Zhou, Y.: SERP-SCA: A Strengthened Side-Channel Attack Framework for FALCON Signature. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **45**(4), 2028–2040 (Apr 2026). https://doi.org/10.1109/TCAD.2025.3608056
11. Choi, H., Noh, J., Lee, S., Shin, D.J.: A Secure and Efficient Implementation of the FALCON BaseSampler Against Side-Channel Attack. In: 2025 16th International Conference on Information and Communication Technology Convergence (ICTC). pp. 1360–1361 (Oct 2025). https://doi.org/10.1109/ICTC66702.2025.11387816, iSSN: 2162-1241
12. Ducas, L., Galbraith, S., Prest, T., Yu, Y.: Integral matrix gram root and lattice gaussian sampling without floats. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 608–637. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45724-2_21
13. Espitau, T., Fouque, P.A., Gérard, F., Rossi, M., Takahashi, A., Tibouchi, M., Wallet, A., Yu, Y.: Mitaka: A simpler, parallelizable, maskable variant of falcon. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part III. Lecture Notes in Computer Science, vol. 13277, pp. 222–253. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_9
14. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th Annual ACM Symposium on Theory of Computing. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407
15. Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M.: The hidden parallelepiped is back again: Power analysis attacks on falcon. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(3), 141–164 (2022). https://doi.org/10.46586/tches.v2022.i3.141-164
16. Halmans, S., van Vredendaal, C., Schneider, T., Custers, F., Güneysu, T.: Twfalcon: Triple-word arithmetic for falcon: Giving falcon the precision to fly securely. IACR Transactions on Cryptographic Hardware and Embedded Systems **2026**(2), 347—-371 (2026). https://doi.org/10.46586/tches.v2026.i2.347-371
17. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Third Algorithmic Number Theory Symposium (ANTS). Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer (Jun 1998)
18. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27

19. Karabulut, E., Aysu, A.: FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 691–696 (Dec 2021). https://doi.org/10.1109/DAC18074.2021.9586131, iSSN: 0738-100X
20. Karabulut, E., Aysu, A.: Masking FALCON's floating-point multiplication in hardware. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(4), 483–508 (2024). https://doi.org/10.46586/tches.v2024.i4.483-508
21. Kim, G., Lee, J., Lee, M., Hong, S., Kim, H.: Secret key recovery of falcon using simple power analysis in conditional calculator. IACR Transactions on Cryptographic Hardware and Embedded Systems **2026**(2), 953–975 (2026). https://doi.org/10.46586/tches.v2026.i2.953-975
22. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO'96. Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer, Berlin, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_9
23. Lin, X., Zhang, S., Yu, Y., Wang, W., You, Q., Xu, X., Wang, X.: Thorough power analysis on falcon gaussian samplers and practical countermeasure. In: Jager, T., Pan, J. (eds.) PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 15674, pp. 229–258. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91820-9_8
24. McCarthy, S., Howe, J., Smyth, N., Brannigan, S., O'Neill, M.: BEARZ Attack FALCON: Implementation Attacks with Countermeasures on the FALCON Signature Scheme:. In: Proceedings of the 16th International Joint Conference on e-Business and Telecommunications. pp. 61–71. SCITEPRESS - Science and Technology Publications, Prague, Czech Republic (2019). https://doi.org/10.5220/0007834800610071
25. Paterson, M., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM J. Comput. **2**(1), 60–66 (1973). https://doi.org/10.1137/0202007
26. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
27. Qiu, J., Aysu, A.: SHIFT SNARE: Uncovering Secret Keys in FALCON via Single-Trace Analysis (May 2025). https://doi.org/10.48550/arXiv.2504.00320, arXiv:2504.00320 [cs]
28. Qiu, J., Aysu, A.: NTT-PEEL: Bit Shift Side-Channel in FALCON's Number Theoretic Transform. IACR Transactions on Cryptographic Hardware and Embedded Systems **2026**(2), 709–735 (2026). https://doi.org/10.46586/tches.v2026.i2.709-735
29. Remez, E.: Sur la détermination des polynômes d'approximation de degré donnée. Comm. Soc. Math. Kharkov (1934)
30. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365700
31. Zhang, S., Lin, X., Yu, Y., Wang, W.: Improved power analysis attacks on falcon. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part IV. Lecture Notes in Computer Science, vol. 14007, pp. 565–595. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30634-1_19

Appendix

This Appendix contains three sections, one presenting algorithmic descriptions for FALCON, one listing the minimax polynomials used in the paper, and one detailing the precision analysis for the inversion and the use of the Newton-Raphson method for the inverse square root, the latter being less efficient than the method presented in the main body of the paper.

A FALCON Algorithmic Descriptions

A.1 Key Generation

The key generation procedure generates the NTRU lattice as well as a tree of coefficients that is used in the Gaussian sampler within the signature procedure, or FALCON-tree. The coefficients of this tree are normalised during the procedure with the application of the inverse square root function.

Algorithm 1: FALCON Key Generation [26]

Data: A monic polynomial $\phi \in \mathbb{Z}[x]$, a modulus q
Result: A secret key sk, a public key pk

$f, g, F, G \leftarrow \texttt{NTRUGen}(\phi, q)$
$\mathbf{B} = \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$
$\hat{\mathbf{B}} \leftarrow \texttt{FFT}(\mathbf{B})$
$\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^{\star}$
$T \leftarrow \texttt{fftLDL}^{\star}(\mathbf{G})$
foreach *leaf* $leaf$ *in* T **do**
  $leaf.value \leftarrow \sigma / \sqrt{leaf.value}$ `// Use of inverse square root`
$sk \leftarrow (\hat{\mathbf{B}}, T)$
$h \leftarrow gf^{-1} \; mod \; q$
$pk \leftarrow h$
return sk, pk

A.2 SamplerZ

The Gaussian sampler used during the signature procedure[4], denoted `SamplerZ`, can be found in [26] page 43 Algorithm 15. The input value σ' comes from the FALCON-tree stored in the secret key and refers to the sensitive variable $leaf.value$ calculated in the key generation procedure.

[4] For the signature, see [26], page 39 Algorithm 10.

Algorithm 2: Discrete Gaussian sampler [26]: `SamplerZ`

Data: Floating-point values μ,$\sigma' \in \mathbb{R}$ such that $\sigma' \in [\sigma_{\min}, \sigma_{\max}]$
Result: $z \in \mathbb{Z}$ sampled from a distribution very close to $D_{\mathbb{Z},\mu,\sigma'}$

$r \leftarrow \mu - \lfloor \mu \rfloor$;
$ccs \leftarrow \sigma_{min}/\sigma'$; // Use of inversion
while *1* **do**
  $z_0 \leftarrow$ `BaseSampler`();
  $b \leftarrow$ `UniformBits`(8) & 0x1;
  $z \leftarrow b + (2 \cdot b - 1)z_0$;
  $x \leftarrow \frac{(z-r)^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{\max}}$; // Use of inversion
  if `BerExp`$(x, ccs) = 1$ **then**
    return $z + \lfloor \mu \rfloor$;

B Minimax Polynomials

In this section, the various minimax polynomials used throughout the paper are listed. The set of FALCON parameters used is FALCON-512.

B.1 Inversion

- 1st order: $P(X) = -0.429\,867\,864\,8X + 1.321\,580\,138$
- 2nd order: $P(X) = 0.279\,644\,610\,1X^2 - 1.292\,951\,889\,1X + 1.977\,223\,230\,6$
- 3rd order: $P(X) = -0.181\,918\,943\,9X^3 + 1.122\,934\,814\,1x^2 - 2.585\,929\,858\,4X + 2.6328663232$
- 4th order: $P(X) = 0.118\,344\,859\,6X^4 - 0.913\,845\,687\,5X^3 + 2.811\,744\,307\,5X^2 - 4.308\,775\,895\,7X + 3.288\,509\,415\,8$
- 5th order: $P(X) = -0.076\,987\,616\,1X^5 + 0.713\,756\,369\,4X^4 - 2.748\,688\,231\,9X^3 + 5.627\,913\,381\,8X^2 - 6.461\,489\,799\,3X + 3.944\,152\,508\,3$

B.2 Inversed Square Root

Without Integrating σ

- 1st order: $P(X) = -3.837\mathtt{e}{-7}X + 0.013\,952\,814\,4$

Integrating σ

- 1st order: $P(X) = -1.746\,633\,755\,6X + 2.312\,492\,255\,4$
- 2nd order: $P(X) = 3.032\,190\,161\,1X^2 - 4.490\,600\,077\,3X + 2.895\,058\,304\,3$
- 3rd order: $P(X) = -5.724\,859\,727\,7X^3 + 10.752\,384\,171\,6x^2 - 7.849\,707\,873\,8X + 3.3667425938$
- 4th order: $P(X) = 11.325\,815\,87X^4 - 26.153\,619\,223\,3X^3 + 24.291\,226\,570\,9X^2 - 11.754\,656\,210\,7X + 3.780\,207\,269\,4$
- 5th order: $P(X) = -23.027\,582\,215\,1X^5 + 63.362\,510\,882\,8X^4 - 72.493\,175\,904\,1X^3 + 44.610\,576\,219\,8X^2 - 16.140\,745\,059\,6X + 4.153\,026\,840\,8$

B.3 Square Root

Without Integrating σ

- 1st order: $P(X) = 0.004\,530\,210\,3X + 54.332\,233\,344\,1$
- 2nd order: $P(X) = -9.4\texttt{e}{-8}X^2 + 0.006\,877\,000\,9X + 40.562\,780\,030\,1$

Integrating σ

- 1st order: $P(X) = 0.750\,821\,723\,2X + 0.327\,822\,748\,3$
- 2nd order: $P(X) = -0.428\,093\,521\,6X^2 + 1.139\,770\,869\,3X + 0.244\,742\,415\,5$
- 3rd order: $P(X) = 0.485\,507\,655\,6X^3 - 1.085\,808\,269\,3x^2 + 1.427\,404\,832\,3X + 0.2041618789$
- 4th order: $P(X) = -0.687\,120\,492X^4 + 1.728\,467\,223\,2X^3 - 1.912\,165\,151\,8X^2 + 1.666\,539\,780\,8X + 0.178\,758\,689\,3$
- 5th order: $P(X) = 1.088\,014\,866\,9X^5 - 3.150\,574\,972\,1X^4 + 3.926\,826\,405\,8X^3 - 2.878\,254\,908\,1X^2 + 1.875\,552\,131\,7X + 0.160\,952\,224\,3$

C Precision

C.1 Inversion Newton-Raphson [3]

Inversion was already covered in the state-of-the-art [3] in the case of floating-point representation and thus double precision. To compute the inverse of a value a, they used the Newton-Raphson method with the following sequence:

$$\forall n \in \mathbb{N}, x_{n+1} = x_n * (2 - a * x_n). \tag{18}$$

It converges quadratically if its first term x_0 is close enough to $\frac{1}{a}$. The complexity of the computation for i iterations and an order o minimax polynomial is:

$$\texttt{NRInv}(o, i) = (o + i)\texttt{AddCst} + (2i + o - 1)\texttt{Mul} + \texttt{MulCst}. \tag{19}$$

Table 7 lists the precision of using the sequence for the inverse in the signature of FALCON for different values of i and o. In the signature, the interval on which the sequence is applied is $[\sigma_{min}, \sigma_{max}]$.

Table 7. Precision of the Newton-Raphson sequence for the inverse depending on the number of iterations (columns) and the order of the minimax polynomial (rows).

	0	1	2	3	4
1	0.011	2.0e−4	6.8e−8	8.4e−15	1.3e−28
2	9.1e−4	1.6e−6	4.2e−12	3.1e−23	1.8e−45
3	8.1e−5	1.2e−8	2.5e−16	1.2e−31	2.4e−62
4	7.1e−6	9.2e−11	1.6e−20	4.2e−40	3.2e−79

Recommended Choices Table 7 corroborates the results from the state-of-the-art [3], that is, the choice with the lowest number of calls to each subgadget (considered separately) for both double and triple-world precision is:

$$\texttt{NRInv}(o = 2, i = 3) = 5\texttt{AddCst} + 7\texttt{Mul} + \texttt{MulCst}. \tag{20}$$

C.2 Inversion Halley

In this section, we detail the alternative to the Newton-Raphson method to compute the inverse of a value a. The Halley method uses the following sequence:

$$\begin{aligned} \forall n \in \mathbb{N},\ z_n &= 1 - a * x_n \\ x_{n+1} &= x_n * (1 + z_n * (1 + z_n)). \end{aligned}$$

It converges cubically if its first term x_0 is a good approximation of $1/a$.

Although its convergence speed is faster than that of the Newton-Raphson method, the Halley method suffers from larger complexity per iteration:

$$\texttt{HalleyInv}(o, i) = (3i + o)\texttt{AddCst} + (3i + o - 1)\texttt{Mul} + \texttt{MulCst}. \tag{21}$$

Table 8 lists the precision of using the sequence for the inverse in the signature of FALCON for different values of i and o. In the signature, the interval on which the sequence is applied is $[\sigma_{min}, \sigma_{max}]$.

Table 8. Precision of the Halley sequence for the inverse depending on the number of iterations (columns) and the order of the minimax polynomial (rows).

	0	1	2	3	4
1	0.011	3.7e−6	1.6e−16	1.3e−47	6.9e−141
2	9.1e−4	2.5e−9	5.1e−26	4.4e−76	2.8e−226
3	8.1e−5	1.8e−12	1.7e−35	1.5e−104	1.2e−311
4	7.1e−6	1.2e−15	5.4e−45	5.2e−133	-

Recommended Choices According to Table 8, the choice with the lowest number of calls to each subgadget (considered separately) for both double and triple-word precision is:

$$\texttt{HalleyInv}(o = 2, i = 2) = 8\texttt{AddCst} + 7\texttt{Mul} + \texttt{MulCst}. \tag{22}$$

$\texttt{HalleyInv}(o = 2, i = 2) - \texttt{NRInv}(o = 2, i = 3) = 3\texttt{AddCst}$. Therefore, the Halley method is less efficient than the Newton-Raphson approach to compute the inversion for double and triple-word precisions.

C.3 Inverse Square Root Newton-Raphson

In this section, we detail the alternative to the Halley method to compute the inverse square root of a value a. The Newton-Raphson method uses the following sequence:

$$\forall n \in \mathbb{N}, x_{n+1} = \frac{x_n}{2}\left(3 - a * x_n^2\right). \tag{23}$$

It converges quadratically if its first term x_0 is a good approximation of $1/\sqrt{a}$.

Although its convergence speed is lower than that of the Halley method, the Newton-Raphson method benefits from lighter complexity per iteration:

$$\begin{aligned}\texttt{NRInvSqrt}(o, i) = i * (\texttt{AddCst} + 3\texttt{Mul} + \texttt{MulCst}) + \\ o * \texttt{AddCst} + (o-1) * \texttt{Mul} + \texttt{MulCst}.\end{aligned}$$

Table 9 gives the precision of applying the sequence to compute the inverse square root in FALCON for different values of i and o. The interval in which the sequences are computed is given by $\left[\left(\frac{\sigma}{\sigma_{max}}\right)^2, \left(\frac{\sigma}{\sigma_{min}}\right)^2\right]$. However, this interval has high bounds, where the inverse square root function "flattens". As a consequence, the Remez algorithm cannot compute second order and above minimax polynomials with the imposed double precision.

Table 9. Precision of the Newton-Raphson sequence depending on the number of iterations (columns) and the order of the minimax polynomial (rows)

	0	1	2	3	4
1	2.2e−4	8.7e−6	1.5e−8	4.2e−14	3.4e−25

Taking the full Equation 1, we can directly integrate the multiplication by σ to reduce the limits of the interval on which the sequences are computed:

$$leaf.value = \frac{\sigma}{\sqrt{leaf.value}} = \frac{1}{\sqrt{\frac{leaf.value}{\sigma^2}}} = \frac{1}{\sqrt{y}} \text{ with } y \in \left[\frac{1}{\sigma_{max}^2}, \frac{1}{\sigma_{min}^2}\right]. \tag{24}$$

As a consequence, the Remez algorithm with double precision can compute multiple orders of minimax polynomials, and we can once again study the trade-off possibilities between minimax order and number of iterations. Table 10 provides the precision of computing the inverse square root multiplied by σ.

Recommended Choices According to Tables 9 and 10, there are several possible choices. For instance, we can achieve both double and triple precision with an order 1 minimax polynomial and 4 iterations for the Newton-Raphson sequence. However, complexity-wise, it can be more interesting to use higher order minimax polynomials to approximate the first term:

Table 10. Precision of the Newton-Raphson sequence depending on the number of iterations (columns) and the order of the minimax polynomial (rows), with the multiplication by σ integrated.

	0	1	2	3	4
1	0.037	1.5e−3	2.4e−6	6.7e−12	5.3e−23
2	0.0070	5.2e−5	2.9e−9	9.1e−18	8.9e−35
3	0.0012	1.6e−6	2.7e−12	8.0e−24	7.2e−47
4	1.9e−4	4.1e−8	1.9e−15	4.1e−30	1.9e−59
5	3.1e−5	1.1e−9	1.3e−18	1.9e−36	4.0e−72

For double precision (error of at most $\approx 1.12\texttt{e} - 16$),

$$\texttt{NRInvSqrt}(o = 2, i = 3) = 5\texttt{AddCst} + 10\texttt{Mul} + 4\texttt{MulCst}.$$
$$\texttt{NRInvSqrt}(o = 5, i = 2) = 7\texttt{AddCst} + 10\texttt{Mul} + 3\texttt{MulCst}.$$

For triple precision (error of at most $\approx 2.12\texttt{e}{-22}$),

$$\texttt{NRInvSqrt}(o = 1, i = 4) = 5\texttt{AddCst} + 12\texttt{Mul} + 5\texttt{MulCst}.$$
$$\texttt{NRInvSqrt}(o = 3, i = 3) = 6\texttt{AddCst} + 11\texttt{Mul} + 4\texttt{MulCst}.$$

An interesting point is that, in the absence of a specialised gadget for multiplication by a constant `MulCst`, there are as many calls to multiplications for $\texttt{NRInvSqrt}(o = 2, i = 3)$ than for $\texttt{HalleyInvSqrt}(o = 2, i = 2)$, the latter even costing an additional `AddCst`. We deemed this scenario unlikely enough to choose the Halley method as the recommended choice due to its lower cost in terms of non-scalar multiplications `Mul`.