The object of research is digital signatures. The Falcon digital signature scheme is one of the finalists in the NIST post-quantum cryptography competition. Its distinctive feature is the use of floating-point arithmetic. However, floating-point arithmetic has so-called rounding noise, which accumulates during computations and in some cases may lead to significant changes in the processed values. The work considers the problem of using rounding noise to build attacks on implementation. The main result of the study is a novel attack on implementation, which enables the secret key recovery. This attack differs from existing attacks in using two separately secure implementations with different computation orders. As a result of the analysis, the conditions under which secret key recovery is possible were revealed. The attack requires 300,000 signatures and two implementations to recover key. The probability of successful attack ranges from 70 % to 76 %. This probability is explained by the structure of the Gaussian sampling algorithm used in the Falcon digital signature. At the same time, a necessary condition for conducting an attack is identical seed during signature generation. This condition makes the attack more theoretical than practical since the correct implementation of the Falcon makes probability of two identical seeds negligible. However, the possible usage of floating-point noise shows potential existence of additional attack vectors for the Falcon that should be covered in security models. The results could be used in the construction of digital signature security models and their implementation in existing information and communication systems

Keywords: quantum-resistant transformations, lattice-based cryptography, attack on implementation, NIST PQC, NTRU

UDC 004.056.5
DOI: 10.15587/1729-4061.2024.295160

DETERMINING THE EFFECT OF A FLOATING POINT ON THE FALCON DIGITAL SIGNATURE ALGORITHM SECURITY

Oleksandr Potii
Doctor of Technical Sciences
Deputy Head of State Special Communications
State Service of Special Communications
and Information Protection of Ukraine
Solomyanska str., 13, Kyiv, Ukraine, 03110
Olena Kachko
PhD
Department of Software Engineering
Kharkiv National University of Radio Electronics
Nauky ave., 14, Kharkiv, Ukraine, 61166
Head of Department
Department of Programming **
Serhii Kandii
Postgraduate Student*
Research Associate-Consultant**
Yevhenii Kaptol
Corresponding author
Postgraduate Student*
Information Protection Systems Analyst**
E-mail: kaptevg@gmail.com
*Department of Security Information Systems and Technologies
V. N. Karazin Kharkiv National University
Svobody sq., 6, Kharkiv, Ukraine, 61022
**Institute of Information Technologies PrJSC
Kolomenska str., 15, Kharkiv, Ukraine, 61166

Received date 06.11.2023
Accepted date 25.01.2024
Published date 28.02.2024

How to Cite: *Potii, O., Kachko, O., Kandii, S., Kaptol, Y. (2024). Determining the effect of a floating point on the falcon digital signature algorithm security. Eastern-European Journal of Enterprise Technologies, 1 (9 (127)), 52–59. doi: https://doi.org/10.15587/1729-4061.2024.295160*

1. Introduction

Cryptographic protection of information is an integral part of modern information and telecommunication systems. In particular, digital signatures are widely used to verify data integrity and authenticate users both on the Internet as components of separate protocols, such as TLS, and in banking and other strategically important areas for the state as part of public key infrastructure. Such widespread use of digital signatures makes it important to study the security of these crypto transformations.

At the same time, existing digital signature standards are vulnerable to attacks using quantum computers, which, against the background of the rapid development of quantum technologies in the future, poses a threat to the security of existing information and telecommunication systems. To overcome this problem, the US National Institute of Standards and Technology held an open NIST PQC competition on quantum-resistant (post-quantum) cryptography. In 2022, a number of candidates for standardization were selected based on the results of the competition. The scientific research of the NIST PQC finalists is important because these crypto transformations will replace today's standards in the future.

The Falcon digital signature scheme [1] is a finalist in the NIST PQC competition [2]. One of the features of Falcon is the use of floating-point arithmetic. Noise in the least significant bits is a special property of floating-point arithmetic, which is determined by the order of calculations [3]. This leads to additional risks and calls for a more detailed study into the impact of this phenomenon on the security of the digital signature scheme.

Summarizing the above, research on the security of quantum resistant digital signatures is important since these cryptographic transformations will be standardized in the near future and they will ensure the security of information and telecommunication systems. In particular, Falcon, as a finalist in the NIST PQC competition, needs careful investigation. Therefore, research into the development of methods for bypassing and exploiting the vulnerabilities of digital signature schemes to design new attacks is relevant. This especially applies to its special property – the use of floating-point arithmetic.

2. Literature review and problem statement

In [4], it was proposed to use the Gaussian distribution on lattices to overcome the problem of information leakage about the secret basis of the lattice. But the questions related to the practical implementation of the proposed method remained unresolved since effective implementation required the use of floating-point arithmetic, and there were no studies on the influence of floating-point in cryptography at that time. An option to overcome the relevant difficulties may be to replace calculations with floating point in calculations with rational numbers. This is the approach used in work [5]. However, this approach turned out to be too impractical. The reason for this was too great a slowdown of implementations. In [6], a new Gaussian sampling algorithm using floating-point arithmetic and an analysis of the required accuracy of this algorithm were proposed. The above research results showed that hundreds of bits of precision are required. It was shown in [7] that the required accuracy could be reduced to 53 bits with the correct selection of parameters. But the result used a number of heuristics and the question of the exact analysis of the general case remained unresolved. In work [8], Renyi divergence was used to justify the safety of using 53 bits of accuracy, which put an end to the theoretical side of the issue.

In [9] it was proposed to use the Fourier transform to speed up work with polynomials in cryptographic systems on lattices. However, there is no analysis of the influence of floating-point arithmetic accuracy in the cited work. In [10], a side channel attack on the Falcon digital signature was proposed, which uses the features of multiplying polynomials in the Fourier representation. It has been shown that it is possible to protect against this attack with the help of the correct implementation of calculations. But in [11], a new attack on the implementation was proposed, which made it possible to recover the secret base using power analysis. The conditions for this attack were improved in work [12]. The authors managed to fully recover the secret key. However, the attack needed 45,000 signatures.

All this gives reason to assert that it is expedient to carry out research into the development of new attacks on the implementation that use the properties of floating-point arithmetic.

3. The aim and objectives of the study

The purpose of our study is to determine the impact of floating-point arithmetic on the security of the Falcon reference implementation. This will make it possible to develop a new attack on the Falcon scheme. The study will enable identification of additional attack vectors on both the Falcon scheme and other digital signature schemes that use floating-point arithmetic, which should be taken into account in security models. That also provides prerequisites for improving both the security of digital signature schemes and security models for digital signature schemes, as well as expanding the possibilities for evaluating digital signature schemes.

To achieve the goal, the following tasks were set:

– to determine the conditions of impact of rounding noise on the security of the Falcon;

– to build an attack on Falcon;

– to estimate the probability of implementation of the proposed attack.

4. The study materials and methods

The object of our research is the security of the Falcon against attacks based on the use of features of the application of floating-point arithmetic.

The main hypothesis of the study assumes the existence of the impact of floating-point computing noise on the Falcon security.

For the research, the Falcon scheme [1, 2] adopted for standardization based on the results of the NIST PQC competition and its reference implementation was used.

The features of the cryptographic transformation from the composition of the Falcon, which was used in the study of the influence of floating point on the security of cryptographic transformations, are given below.

Description of Falcon transformation. Falcon uses the transformation in the field $Z_q[X]/(x^n+1)$, where q=12289 and n is the power of two (512 or 1024). The secret key is the polynomials f, g, F, G$\in Z_q[X]/(x^n+1)$, the coefficients of which are small and the NTRU equation is fulfilled:

$$fG - gF = q \bmod x^n + 1. \quad (1)$$

The public key is a polynomial $h \in \mathbb{Z}_q[X]/(x^n+1)$, for which the equation is fulfilled:

$$h = g \cdot f^{-1} \bmod x^n + 1 \bmod q. \quad (2)$$

The polynomials *f, g, F, G* form the basis of the NTRU lattice:

$$B = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}. \quad (3)$$

For calculations in Falcon, fast Fourier transform is used for optimization. The Fourier representation of an arbitrary polynomial b is denoted as $\hat{b}$. Accordingly, the Fourier representation of the basis B is defined as follows:

$$\hat{B} = \begin{pmatrix} \hat{g} & -\hat{f} \\ \hat{G} & -\hat{F} \end{pmatrix}. \quad (4)$$

The Falcon specification provides for the use of field $\mathbb{Z}_q[X]/(x^n+1)$, properties. In particular, the following isomorphisms are used:

$$\mathbb{Z}^n \cong \left(\mathbb{Z}[X]/(x^2+1)\right)^{n/2} \cong \ldots$$
$$\ldots \cong \left(\mathbb{Z}[X]/(x^{n/2}+1)\right)^2 \cong \mathbb{Z}[X]/(x^n+1). \quad (5)$$

Formula (5) means that any polynomial in $\mathbb{Z}_q[X]/(x^n+1)$ has an isomorphic representation in the form of a vector of

two polynomials y $\mathbb{Z}_q[X]/\left(x^{n/2}+1\right)$, a vector of four polynomials y $\mathbb{Z}_q[X]/\left(x^{n/4}+1\right)$, etc.

In particular, this property was used by the authors of Falcon to build a sampling algorithm. In the general case, the *Sampler* sampling algorithm takes as input some lattice basis *A*, the target vector *cc*, and returns a sufficiently small vector *s*, for which the following is performed:

$$sA = c \bmod q. \tag{6}$$

In Falcon, the sampling algorithm *ffSampling* recursively performs the sampling procedure. Calculations start at $\mathbb{Z}^{\mathbf{n}}$ and recursively climb into the field $\mathbb{Z}_q[X]/(x^n+1)$. To this end, *ffSampling* uses the so-called "Falcon Tree" data structure, which stores information about basis (4) in a computationally convenient form. Implementation details can be found in the specification [1]. The only essential factor for this attack is that *ffSampling* works with Fourier representations and returns the Fourier representation of the polynomial *s*. According to the specification, the structure "Falcon Tree" [1] basis (4) has the notation $T=T\left(\hat{B}\right)$.

The signature procedure (simplified) can be described as follows:

Algorithm Sign.

Input data: message *m* in the format of a bit string, the basis – the NTRU lattice $\hat{B}$.

Output: signature (s_1, *nonce*), where s_1 are polynomials in $\mathbb{Z}_q[X]/(x^n+1)$, *nonce* is a random bit string used to randomize the message:

1. Obtain *nonce* from a uniform distribution.

2. Calculate the random polynomial *c* as a hash of the value from $m\|nonce$.

3. Calculate $\hat{t}=\left(\hat{c},\hat{0}\right)\cdot\hat{B}^{-1}$.

4. Calculate the vector $\hat{z}=\left(\hat{z}_0,\hat{z}_1\right)$ using the *ffSampling* algorithm (and $T=T\left(\hat{B}\right)$) for the target point $\hat{t}$.

5. Calculate $\hat{s}=\left(\hat{s}_0,\hat{s}_1\right)=\left(\hat{t}-\hat{z}\right)\hat{B}$ in the Fourier basis and the corresponding vector $s=(s_0,s_1)$ using the inverse Fourier transform.

6. If the norm of the vector *s* is small enough, then return the signature (s_1, *nonce*), otherwise, return to step 4.

According to the Falcon specification, the signature is checked using the equation:

$$s_0 + s_1 h = c. \tag{7}$$

Equation (7) establishes the relationship between the signature, the message, and the public key. The polynomial s_0 from the signature (s_1, *nonce*) can be reconstructed as $s_0=c-s_1h$. If the polynomial is reconstructed correctly, the signature is considered valid.

5. Results of investigating the influence of floating point on the security of Falcon

5. 1. Determining the conditions of influence of rounding noise on the security of digital signatures

Rounding noise occurs in the least significant bits of variables but tends to increase with the number of calculations, so it makes sense to expect the largest difference in variables for which the most complex transformations are used to calculate. The calculation $\hat{z}=\left(\hat{z}_0,\hat{z}_1\right)$ requires the largest number of operations because the vector $\hat{z}=\left(\hat{z}_0,\hat{z}_1\right)$ is the result of the *ffSampling* algorithm, which actively uses the Fourier transform and is the most complex part of the Falcon from an algorithmic point of view.

Suppose that there are two implementations of the Falcon, which with the same values of *seed* (a random value used in *ffSampling*) and *nonce* for the same keys give different signatures due to noise during calculations. Hereafter, the corresponding variables are denoted by superscripts. For example, $\left(s_0^0,s_1^0\right)$ is the vector *s* in the first signature and $\left(s_0^1,s_1^1\right)$ is the vector *s* in the second signature.

The notation $\delta_0=z_0^1-z_0^0$ and $\delta_1=z_1^1-z_1^0$. is introduced.

Statement 1. For given δ_0, δ_1, the relationship between the public key *h* and the secret key *f, g, F, G* is described by the relation:

$$h=\frac{g\delta_0+G\delta_1}{f\delta_0+F\delta_1}. \tag{8}$$

This result can be obtained from a direct analysis of Falcon transformations.

The calculation of the signature $s=(s_0,s_1)$ on the key *f, g, F, G* is considered below. According to the definition of the signature algorithm, we have:

$$\hat{s}=\left(\hat{t}-\hat{z}\right)\hat{B}=\hat{t}\hat{B}-\hat{z}\hat{B}=(\hat{c},\hat{0})\hat{B}^{-1}\hat{B}-\hat{z}\hat{B}=$$
$$\begin{pmatrix}\hat{c}\\ \hat{0}\end{pmatrix}-\begin{pmatrix}\hat{z}_0\hat{g}+\hat{z}_1\hat{G}\\ -\hat{z}_0\hat{f}-\hat{z}_1\hat{F}\end{pmatrix}=\begin{pmatrix}\hat{c}-\hat{z}_0\hat{g}-\hat{z}_1\hat{G}\\ \hat{z}_0\hat{f}+\hat{z}_1\hat{F}\end{pmatrix}. \tag{9}$$

So, there is equality:

$$s_0 = c - z_0 g - z_1 G,$$
$$s_1 = z_0 f + z_1 F. \tag{10}$$

Considering the connection between the signature, the message, and the public key, described by equality (7), we have:

$$\begin{cases}s_0^0+s_1^0h=c\\ s_0^1+s_1^1h=c\end{cases}\Rightarrow$$
$$s_0^0+s_1^0h-s_0^1-s_1^1h=c-c\Rightarrow$$
$$s_0^0-s_0^1=s_1^1h-s_1^0h\Rightarrow$$
$$h=\frac{s_0^0-s_0^1}{s_1^1-s_1^0}.$$

That is, *h* can be expressed in terms of $\left(s_0^0,s_1^0\right)$ and $\left(s_0^1,s_1^1\right)$.

Substituting (10) into the expression for *h*, we get:

$$h=\frac{c-z_0^0g-z_1^0G-c+z_0^1g+z_1^1G}{z_0^1f+z_1^1F-z_0^0f-z_1^0F}=$$
$$=\frac{g\left(z_0^1-z_0^0\right)+G\left(z_1^1-z_1^0\right)}{f\left(z_0^1-z_0^0\right)+F\left(z_1^1-z_1^0\right)}. \tag{11}$$

Since $\delta_0=z_0^1-z_0^0$ and $\delta_1=z_1^1-z_1^0$, expression (11) implies expression (8), which had to be proved.

If you choose the message *m* so that $\delta_1=0$, and δ_0 is some small enough to use the algorithm for finding the greatest common divisor of a polynomial (ideally – $\delta_0\in\mathbb{Z}$), then it may be easy to recover the secret key. Note that the

situation when $\delta_0=0$, and $\delta_1\neq 0$ is not possible due to the structure of the *ffSampling* algorithm (Fig. 1).

Algorithm 11 $\text{ffSampling}_n(\mathbf{t}, \mathbf{T})$

Require: $\mathbf{t} = (t_0, t_1) \in \text{FFT}(\mathbb{Q}[x]/(x^n+1))^2$, a FALCON tree T
Ensure: $\mathbf{z} = (z_0, z_1) \in \text{FFT}(\mathbb{Z}[x]/(x^n+1))^2$
Format: All polynomials are in FFT representation.

1: **if** $n = 1$ **then**
2: $\quad\sigma' \leftarrow$ T.value ▷ It is always the case that $\sigma' \in [\sigma_{\min}, \sigma_{\max}]$
3: $\quad z_0 \leftarrow \text{SamplerZ}(t_0, \sigma')$ ▷ Since $n = 1$, $t_i = \text{invFFT}(t_i) \in \mathbb{Q}$ and $z_i = \text{invFFT}(z_i) \in \mathbb{Z}$
4: $\quad z_1 \leftarrow \text{SamplerZ}(t_1, \sigma')$
5: $\quad$ **return** $\mathbf{z} = (z_0, z_1)$
6: $(\ell, \mathbf{T}_0, \mathbf{T}_1) \leftarrow$ (T.value, T.leftchild, T.rightchild)
7: $\mathbf{t}_1 \leftarrow \text{splitfft}(t_1)$ ▷ $\mathbf{t}_0, \mathbf{t}_1 \in \text{FFT}\left(\mathbb{Q}[x]/(x^{n/2}+1)\right)^2$
8: $\mathbf{z}_1 \leftarrow \text{ffSampling}_{n/2}(\mathbf{t}_1, \mathbf{T}_1)$ ▷ First recursive call to $\text{ffSampling}_{n/2}$
9: $z_1 \leftarrow \text{mergefft}(\mathbf{z}_1)$ ▷ $\mathbf{z}_0, \mathbf{z}_1 \in \text{FFT}\left(\mathbb{Z}[x]/(x^{n/2}+1)\right)^2$
10: $t_0' \leftarrow t_0 + (t_1 - z_1) \odot \ell$
11: $\mathbf{t}_0 \leftarrow \text{splitfft}(t_0')$
12: $\mathbf{z}_0 \leftarrow \text{ffSampling}_{n/2}(\mathbf{t}_0, \mathbf{T}_0)$ ▷ Second recursive call to $\text{ffSampling}_{n/2}$
13: $z_0 \leftarrow \text{mergefft}(\mathbf{z}_0)$
14: **return** $\mathbf{z} = (z_0, z_1)$

Fig. 1. The *ffSampling* algorithm from the Falcon digital signature scheme

If $\delta_1\neq 0$, then, by definition, the values z_1^1, z_1^0 will differ. Then, at step 10 of the *ffSampling* algorithm, the values t_0', will differ, which in turn will lead to changes in the calculation of z_0^1, z_0^0 at steps 10–13.

5. 2. Building an attack on Falcon

From formula (8) and the defined conditions of significant impact of rounding noise on security, it follows that it is possible to calculate the secret key $\tilde{f}$, $\tilde{g}$ as:

$$\tilde{f} = \left(s_1^1 - s_1^0\right) / \gcd\left(s_1^1 - s_1^0, s_0^0 - s_0^1\right),$$

$$\tilde{g} = \left(s_0^0 - s_0^1\right) / \gcd\left(s_1^1 - s_1^0, s_0^0 - s_0^1\right), \quad (12)$$

where gcd is the greatest common divisor of polynomials.

Reference implementation of Falcon [13] provides two interfaces for users. The first interface calculates $T = T\left(\hat{B}\right)$ in the signature generation procedure. The second interface involves passing $T = T\left(\hat{B}\right)$ as an argument. In implementations, this leads to a change in the order of calculations and accordingly to the occurrence of situations where the attack conditions are fulfilled. The first interface is implemented by the sign_dyn function, the second interface by the sign_tree function. Both interfaces accept a polynomial c instead of a message. The hash_to_point_vartime function is used to form the polynomial c. Keygen generation function is keygen.

All the listed functions take as a parameter a generator of pseudo-random numbers, which deterministically defines their behavior. The generator is implemented by the inner_shake256_context structure. Accordingly, if we fix the generator initialization order before calling each function, then the behavior of each function can be reproduced. The following variables were used in the implementation of the attack:

1) seed – to initialize the generator for keygen;

2) seed2 – to initialize the generator for sign_dyn and sign_tree;

3) nonce – to initialize the generator for hash_to_point_vartime.

At the same time, the initialization of the generator for keygen, sign_dyn, and sign_tree is as follows:

```
inner_shake256_context sc;
inner_shake256_init(&sc);
inner_shake256_inject(&sc, seed, 48);
inner_shake256_flip(&sc);
Ініціалізація генератора для hash_to_point_vartime:
inner_shake256_init(&sc);
inner_shake256_inject(&sc, nonce, 40);
inner_shake256_inject(&sc, message, 33);
inner_shake256_flip(&sc),
```

where message – 33-bit message.

5. 3. Assessing the probability of attack implementation

To detect differences in signatures, signature generation by the sign_dyn and sign_tree functions was run 10^6 times for 10 random key pairs for a random 33-byte message from a uniform distribution. The results for Falcon512 and Falcon1024 are given in Tables 1, 2, respectively.

Table 1

Random signature generation results for Falcon512

Seed of key pair	Number of instances $(\delta_0,\delta_1)\neq(0,0)$	Number of instances $\delta_0\neq 0$, $\delta_1=0$
1	2	3
7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB-505D7CFAD1B497499323C8686325E47	40	35
4AAFF542A5F00256495D2DF0BC45F51FE81D508D2C609F84FA-F1708024A9AA57994E9B452595D0894BE211A3D4759C96	31	28
68F3A6BECC53478296AA6860483396929B42941E2740D932EBB6A5AE-CF7797349AC68E602651A53D58489A9854B541D2	20	17
B5068BF4E738E850F8B8F9BC7D02672E5437B759A7C96080F5DDD4AC6D23D04BC1AF-3580CCE29F3E747757D7BCC84E99	12	8
ED90AA994C90CF9C327D757D9991CC0E28AFA9BC54928BC1558DA8E47FF4E36AA-4351B3926ECF0FAD93F94161A7854AE	42	31
FB044BD398AE04D9D08D2992292E7C913FB2844DC41C7D96D77ADA7B096481252DB-2F366831AD837DD7147FF8071FC7B	41	27
C8E85F2DCA8FB5281781BAF9FC44C0C9A5FDEC8D36EEAF2ECD16D925748260D1DB-43703F056703CE782C79445D94EA28	31	23

Continuation of Table 1

1	2	3
28F4B1954E721DF4F5D58E4D1F0366CFEFCDAE8AA934FE95EBBA753A79C941B76BD-B23E1862136B5DFB4AFB187D29729	36	27
C52535F970985882F44DAC6AB36FDBDC738DF6F2AC096DD75A5D18C95EC3683AD-C56F853DB256459B9E804B7D469A495	23	14
43862DB0C7BE7D28EB9191E1F8735FB24CA4E68F543169F4CA5450B65D34E3ADC4AE-4CF93E3E3FA938E461F8B8556126	37	29

Table 2

Random signature generation results for Falcon1024

Seed of key pair	Number of instances $(\delta_0,\delta_1)\neq(0,0)$	Number of instances $\delta_0\neq 0$, $\delta_1=0$
7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB-505D7CFAD1B497499323C8686325E47	33	22
4AAFF542A5F00256495D2DF0BC45F51FE81D508D2C609F84FA-F1708024A9AA57994E9B452595D0894BE211A3D4759C96	32	28
68F3A6BECC53478296AA6860483396929B42941E2740D932EBB6A5AE-CF7797349AC68E602651A53D58489A9854B541D2	46	37
B5068BF4E738E850F8B8F9BC7D02672E5437B759A7C96080F5DDD4AC6D23D04BC1AF-3580CCE29F3E747757D7BCC84E99	32	26
ED90AA994C90CF9C327D757D9991CC0E28AFA9BC54928BC1558DA8E47FF4E36AA-4351B3926ECF0FAD93F94161A7854AE	28	21
FB044BD398AE04D9D08D2992292E7C913FB2844DC41C7D96D77ADA7B096481252DB-2F366831AD837DD7147FF8071FC7B	26	21
C8E85F2DCA8FB5281781BAF9FC44C0C9A5FDEC8D36EEAF2ECD16D925748260D1DB-43703F056703CE782C79445D94EA28	28	21
28F4B1954E721DF4F5D58E4D1F0366CFEFCDAE8AA934FE95EBBA753A79C941B76BD-B23E1862136B5DFB4AFB187D29729	24	13
C52535F970985882F44DAC6AB36FDBDC738DF6F2AC096DD75A5D18C95EC3683AD-C56F853DB256459B9E804B7D469A495	34	19
43862DB0C7BE7D28EB9191E1F8735FB24CA4E68F543169F4CA5450B65D34E3ADC4AE-4CF93E3E3FA938E461F8B8556126	31	15

From Tables 1, 2, it follows that the probability of $(\delta_0,\delta_1)\neq(0,0)$ is about 0.00003. On average, the attack works 76 % of the time for Flacon512 and 70 % of the time for Falcon1024. Fig. 2, 3 show specific examples of secret key recovery (only polynomials *f* and *g*; polynomials *F*, *G* are not shown because they are uniquely recovered from polynomials *f*, *g*) for Falcon512 and Falcon1024 on the corresponding screenshots of the software implementation. Examples are for seed 7C9935A0B07694AA0C6D10E4DB6B1ADD2F-D81A25CCB148032DCD739936737F2DB505D7CFAD-1B497499323C8686325E47. For clarity, the first 25 coefficient values are given.

The analysis of specific cases showed that all cases $(\delta_0,\delta_1)\neq(0,0)$ can be divided into the following classes.

```
seed2:ED20321FC513FD6CE6BE3B72D30BB9243AF83BE48BB11E8DDA4ACA454827971A6D1DBF79E49D5B487A8911C3FE6C80E8
nonce:CB4872E3C4B517FF810A9ADAE5DFB6DEEC5AB2206A0E7255C9B8F3D553BFC22FBBB0D592652878A7
message:2A3975549DDC553E098247AC3548E7A32A240F2B15BD0FD73E9C34E0421560974C
==========================================================================================
f (first 25 values):1,4,4,2,-4,-4,-2,-2,6,-2,0,-4,5,4,4,2,-3,-2,-3,-11,-2,-2,-2,3,1,
==========================================================================================
recovered f (first 25 values):-1,-4,-4,-2,4,4,2,2,-6,2,0,4,-5,-4,-4,-2,3,2,3,11,2,2,2,-3,-1,
==========================================================================================
g (first 25 values):2,5,5,-5,0,1,5,1,-8,0,5,4,-3,-2,-2,0,1,-4,-6,-3,-1,-1,-3,2,4,
==========================================================================================
recovered g (first 25 values):-2,-5,-5,5,0,-1,-5,-1,8,0,-5,-4,3,2,2,0,-1,4,6,3,1,1,3,-2,-4,
==========================================================================================
```

Fig. 2. Example of key (*f*, *g*) recovery for Falcon512

```
seed2:A92A02571FA5485BD1B0ED910BE9495F197628638A5B72FEA26CDD88C2FEE59167AE5DD828DCCAADE2CCD12D26330B87
nonce:282A86D82E14E0071C7E6F069A3FDEBF363C7B607CC9920677679F5C51F150802C53EB725572340C
message:DE76495BA16EEDFAB50E07BC58DDB7773717727EA4BE942CE0D6E4A3EF2AC3CA22
==========================================================================================
f (first 25 values):-1,4,3,0,-4,2,-4,0,-2,3,6,-2,-4,0,-5,-1,1,1,1,2,1,2,-3,4,2,
==========================================================================================
recovered f (first 25 values):-4,3,3,1,7,4,1,4,-4,8,1,0,3,2,-3,2,3,-1,0,-5,4,3,-2,2,-4,
==========================================================================================
g (first 25 values):2,2,-3,-4,2,-2,1,-1,0,1,1,-2,1,-2,0,-2,4,6,-1,0,-1,0,4,1,-3,
==========================================================================================
recovered g (first 25 values):-7,2,-3,-3,1,0,0,0,-3,0,3,0,0,2,-3,3,1,-4,3,3,-1,0,4,0,-5,
==========================================================================================
```

Fig. 3. Example of restoring key (*f*, *g*) for Falcon1024

Case I. $\delta_0 \neq 0$. Accordingly, the structure of the sampling algorithm is also $\delta_0 \neq 0$, so formula (12) will not give the correct result. Such cases are about 24 % for Falcon512 and about 29 % for Falcon1024.

An example for this case for Falcon512 is:

seed:7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505D7CFAD1B497499323C8686325E47.

seed2:B021D1D163606744EE32CBA2B2A4652E48F3E6F2C808F846167FFFCA7B515690ECACEA9BEA04B5E0FC954FB27347D9D1.

nonce:AE78A186A245729EE3D9F30310B0028F4E316CC3B227D0951058B5CE25CBBC88B8F287046E46CD55.

message:DA3912941533D3BA0E823075C47FCEBAEFC4063EFE6CED337988FE558A6DE7EACE.

An example for this case for Falcon1024 is:

seed:7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505D7CFAD1B497499323C8686325E47.

seed2:D6E73C4B1CB5D4541E2562018FEE6B59469F7689D3FD3923C42D655760F7C0A5964A4D0DD143350B9204FE23EF8AAE88.

nonce:F96DC162AB068C0201399141086B7932B0A737266ECA67870C1366D22A61BC994C4CAFE23BDFB790.

message:D59841626C664717FC1548670C2455EFB14A3285A54EA0AD172EB524CEC3C8382F.

For the $\delta_1=0$, according to the value of δ_0, the following options can be distinguished:

1) Case II-a. When $\delta_1=0$ and $\delta_0=a$, $a\in\mathbb{Z}$. In this case, formula (12) works. Moreover, the value a is not a large number (less than 10). Such cases for Falcon512 make up about 29 % of the total number and about 26 % for Falcon1024.

An example for this case for Falcon512 is:

seed:7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505D7CFAD1B497499323C8686325E47.

seed2:ED20321FC513FD6CE6BE3B72D30BB9243AF83BE48BB11E8DDA4ACA454827971A6D1DBF79E49D5B487A8911C3FE6C80E8.

nonce:CB4872E3C4B517FF810A9ADAE5DFB6DEEC5AB2206A0E7255C9B8F3D553BFC22FBBB0D592652878A7.

message:2A3975549DDC553E098247AC3548E7A32A240F2B15BD0FD73E9C34E0421560974C.

An example for this case for Falcon1024 is:

seed:7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505D7CFAD1B497499323C8686325E47.

seed2:A39A5F21193159E841339E9FFA14928927D3419CF9CC7E7AE060965C62250E37686F97E345A87107DC9D989D3692FA61.

nonce:9BBCAB991352E48EC8A10670BC090A49DB4BB6AAD26B55AADA1EAEB56DC164700DDAB9E68E63D0EF.

message:1579CA95DE67D21B31D052FE6178E7916CB2155092272F49B3C2201C93AA01EB620.

2) Case II-b. When $\delta_1=0$ and $\delta_0=a\cdot x^{n/2}$, $a\in\mathbb{Z}$. In this case, formula (12) holds. Moreover, the value a is not a large number (less than 10). Such cases for Falcon512 make up about 38 % of the total number and about 35 % for Falcon1024.

An example for this case for Falcon512 is:

seed:7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505D7CFAD1B497499323C8686325E47.

seed2:1C010D6173B05C89D4384562986A675BDD6580C9EE81F383E363ECE6CE45F1F3C71673210DFFC33E4803DF1FC6243BF3.

nonce:C1959DD1DEF7AB02A7C452EA192E0CD403CDC825433E51A3CD422F5B300EF7F8391DCFF59FB45CEC.

message:DF17CA32F3FD0D62748606086753CC98A4B073A98E9CAFB1ADB04D62962D83473A.

An example for this case for Falcon1024 is:

seed:7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505D7CFAD1B497499323C8686325E47.

seed2:B3233E8F302418961966363C1F9AAE5F5F010354B30CCB6310C1146B20E865E5588132AA751DC4F8A39DA9ED994AA4D3.

nonce:8E63C1630C0BC417AA166EC74AEE0FC4479C0B7C442CCDEC17662A44D6E789DBF75F4D3FFF4614AA.

message:392FCF43930FB262D8564D8965033F89CA47452EBDD7E6B3686CC315F7E35BA9A7.

3) Case II-c. When $\delta_1=0$ and $\delta_0=a+b\cdot x^{n/2}$, where $a,b\in\mathbb{Z}$**.** This case is a combination of the previous cases and formula (12) will also hold, but this case is interesting because δ_0, is a polynomial. This situation occurs about 7 % of the time for Falcon512 and 9 % of the time for Falcon1024.

An example for this case for Falcon512 is:

seed:4AAFF5422A5F0025649 5D2DF0BC45F51FE81D508D2C609F84FAF1708024A9AA57994E9B4525 95D0894BE211A3D4759C96.

seed2:6F1A9CBCB96BB97F7A8D3BAC7E4097C8B478A7845C586E947F06173E9E1D45E0FF49EF783CE0487FB588CED4C750FB41.

nonce:059347E4990A9A294B99A79A5F03B9078E32A4C9DC57745ADCA5E60ACD1D36F872A7013B9665595B.

message:F65F1869F151887A07982CD9BD46B8093E787B8AD3B91CFBCAD8A36AE0E16297BE.

An example for this case for Falcon1024 is:

seed:7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505D7CFAD1B497499323C8686325E47.

seed2:B0F1BC047ED336FF2923F60D502FC89314CC05B1B0A9E0BD7A8005274D9C7AB60CEE55388A8F15B5C736FFE284420F64.

nonce:05DA9F05EBEC410923479AC5C8B2FC3B-62BEB53AE28C92C78822064C84EBCFFA81DB32BEC-53EFE3F.

message:49DC80378E4716F90FF4CAB599D43AE-C27AA689A5776D9BBFC207B533275514B5D.

Analysis of specific cases showed that the difference arises in the *ffSampling* algorithm. At the deepest level of the recursion in steps 3–4, due to the noise of floating-point arithmetic for individual coefficients, the *SamplerZ* call returns values that differ by ±1, and Fourier representation divergence occurs when ascending to the field $\mathbb{Z}_q[X]/(x^n+1)$. Case I occurs when there is a difference in the calculations in steps 7–9 of the *ffSampler* algorithm. Case II-a in step 7, case II-b in step 12, and case II-c is a combination of II-a and II-b. At the same time, the difference in values caused by rounding noise was 10^{-14}. Coefficients that had less rounding noise did not change.

6. Discussion of results of investigating the effect of floating point on the security of Falcon

An attack on the recovery of secret keys using formula (12) is possible because the reference implementation of the Falcon has two interfaces that use different calculation orders. The difference in calculations is sufficient that the Gaussian sampling algorithm in Fig. 1 at the deepest recursion level returned values that differed by only ±1. Since the probability of a sufficiently large difference in several coefficients is small, an error most often occurs precisely in one coefficient. When performing all calculations, this difference turns into some small integer factor (in practice, the largest value obtained is (7)) of polynomials at the lowest level of recursion. If the difference occurs in the polynomial z_0, then, according to formula (11), the situation $\delta_0 \neq 0$, $\delta_1 = 0$ arises, which is a necessary condition for the operation of formula (12) and the implementation of the attack.

The specificity of the proposed attack is the use of two different software interfaces of the same scheme. Attacks in works [10, 11] also used features of floating-point arithmetic, but previous attacks focused only on the features of one interface. Compared to [12], the proposed attack uses fewer signatures and has a higher probability of recovering the secret key. The proposed attack shows that two individually safe implementations can be dangerous together.

The practical utility of the attack is that it shows how to use the rounding noise itself to cryptanalyze digital signatures. The attack reveals that the Falcon needs to specify more precise conditions for using floating-point arithmetic than specified in the specification because in existing implementations implemented according to the specification, rounding noise is an additional source of randomness that is not taken into account by existing models, making additional attacks possible. When developing new digital signatures that use floating-point arithmetic, there is a need for additional analysis that will demonstrably guarantee the absence of influence of the order of floating-point arithmetic on the results of calculations.

A limitation of our results is the low probability of an attack, which leads to the need to obtain a large number of signatures. Another, more significant, limitation of the attack is the need for the same seed, which makes the attack purely theoretical.

Among the shortcomings of our study, one may include the lack of research into the necessary accuracy of calculations under which the attack will stop working. For the further development of this direction, it is important to obtain an assessment of the conditions for applying the described approach.

7. Conclusions

1. Rounding noise in a floating-point calculation can lead to two similar but different signatures. If the difference is small enough, the secret key can be recovered. A small noise in the arguments of the sampling algorithm of order 10^{-14} allows one to get exactly such a situation.

2. Reference implementation of Falcon provides two interfaces with different calculation order. This results in varying rounding noise in the least significant bits of the variables, causing the sampling algorithm to produce values that differ by ±1. After passing this difference through all calculations, the two signatures differ by some small factor.

3. The probability of an attack is small enough. Given this, about $3 \cdot 10^5$ signed messages are needed for implementation. Although the attack is only possible when the Falcon is misused, it shows that floating-point computations can lead to attack vectors that were previously impossible.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

The data will be provided upon reasonable request.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

Acknowledgments

The authors express their gratitude for the support provided in the preparation of the current paper, as well as useful comments and suggestions, to the research team at Institute of Information Technologies PrJSC.

References

1. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. Available at: https://falcon-sign.info/
2. Post-Quantum Cryptography. NIST. Available at: https://csrc.nist.gov/projects/post-quantum-cryptography

3. Tran, T., Liu, B. (1977). Accumulation of roundoff errors in floating point FFT. IEEE Transactions on Circuits and Systems, 24 (3), 132–143. https://doi.org/10.1109/tcs.1977.1084316
4. Gentry, C., Peikert, C., Vaikuntanathan, V. (2008). How to Use a Short Basis:Trapdoors for hard lattices and new cryptographic constructions. Available at: https://eprint.iacr.org/2007/432.pdf
5. Lyubashevsky, V., Prest, T. (2015). Quadratic Time, Linear Space Algorithms for Gram-Schmidt Orthogonalization and Gaussian Sampling in Structured Lattices. Lecture Notes in Computer Science, 789–815. https://doi.org/10.1007/978-3-662-46800-5_30
6. Ducas, L., Nguyen, P. Q. (2012). Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic. Lecture Notes in Computer Science, 415–432. https://doi.org/10.1007/978-3-642-34961-4_26
7. Prest, T. (2015). Gaussian Sampling in Lattice-Based Cryptography. Paris: ENS PARIS. Available at: https://theses.hal.science/tel-01245066
8. Prest, T. (2017). Sharper Bounds in Lattice-Based Cryptography Using the R nyi Divergence. Lecture Notes in Computer Science, 347–374. https://doi.org/10.1007/978-3-319-70694-8_13
9. Ducas, L., Prest, T. (2016). Fast Fourier Orthogonalization. Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation. https://doi.org/10.1145/2930889.2930923
10. Karabulut, E., Aysu, A. (2021). FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks. 2021 58th ACM/IEEE Design Automation Conference (DAC). https://doi.org/10.1109/dac18074.2021.9586131
11. Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M. (2022). The Hidden Parallelepiped Is Back Again: Power Analysis Attacks on Falcon. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2022 (3), 141–164. https://doi.org/10.46586/tches.v2022.i3.141-164
12. Zhang, S., Lin, X., Yu, Y., Wang, W. (2023). Improved Power Analysis Attacks on Falcon. Lecture Notes in Computer Science, 565–595. https://doi.org/10.1007/978-3-031-30634-1_19
13. Falcon source files (reference implementation). Available at: https://falcon-sign.info/impl/falcon.h.html