Number "Not" Used Once - Key Recovery Fault Attacks on LWE Based Lattice Cryptographic Schemes

Prasanna Ravi[1], Shivam Bhasin[1], and Anupam Chattopadhyay[2]

[1] Temasek Laboratories, Nanyang Technological University, Singapore
[2] School of Computer Science and Engineering
Nanyang Technological University, Singapore

PRASANNA.RAVI@ntu.edu.sg sbhasin@ntu.edu.sg anupam@ntu.edu.sg

Abstract. This paper proposes a simple single bit flip fault attack applicable to several LWE (Learning With Errors Problem) based lattice based schemes like KYBER, NEWHOPE, DILITHIUM and FRODO which were submitted as proposals for the NIST call for standardization of post quantum cryptography. We have identified a vulnerability in the usage of nonce, during generation of secret and error components in the key generation procedure. Our fault attack, based on a practical bit flip model (single bit flip to very few bit flips for proposed parameter instantiations) enables us to retrieve the secret key from the public key in a trivial manner. We fault the nonce in order to maliciously use the same nonce to generate both the secret and error components which turns the LWE instance into an exactly defined set of linear equations from which the secret can be trivially solved for using Gaussian elimination.

1 Introduction

There have been a number of schemes proposed for NIST's standardization process for post quantum cryptography [26]. This initiative is due to the emerging threat of quantum computers to classical public key cryptographic primitives which provide the essential security services to almost all known digital infrastructures [7,20,33]. Among all the known strains of post-quantum cryptography, Lattice based cryptography has attracted a lot of interest due to its very good efficiency and unique security guarantees. A significant amount of research has been done with respect to lattice based cryptography in terms of efficient implementations [28,32], security analysis [2,12,13,14,16,22] and implementation attacks [11,17,18,30,31].

In this short paper, we focus on identifying fault vulnerabilities across many LWE (Learning with Errors Problem) based lattice key encapsulation mechanisms (KEM) like KYBER [10], NEWHOPE [4], FRODO [9] and also the DILITHIUM [15] signature scheme. The afore mentioned references of the schemes denote the earlier versions of the respective schemes, but every analysis in this paper is done with reference to the latest version of the algorithm submitted to the NIST standardization conference for post quantum cryptographic standards. One can find reference implementations and specifications for all the submitted proposals in [27]. We have identified a fault vulnerability in handling the nonce during the key generation and propose a single bit flip attack that could generate the public key as a trivially solvable LWE instance in both the `NEWHOPE` and `FRODO` KEM schemes. Nonces are predominantly used in all the afore mentioned schemes in order to reduce the amount of randomness required to generate the secret and error components used in the generation of the LWE instance during key generation. Similar attacks can also be applied over schemes like `DILITHIUM` and `KYBER` which may not lead to trivial key recovery as in `NEWHOPE` and `FRODO`, but could severely weaken the security of the scheme. Thus, these fault vulnerabilities arise as a single point of failures that have the potential to break down many other LWE based lattice cryptographic schemes. This vulnerability stems only from the implementation in the view of improving performance and hence we believe that this can be easily corrected albeit with a small increase in performance overhead.

2 Background

2.1 Preliminaries

We separately describe in brief, the four schemes `NEWHOPE`, `KYBER`, `FRODO` and `DILITHIUM` with main focus on the key generation algorithm, as that is the target of our attack. We do not describe in detail all the functions that are used in their key generation procedures as they are out of scope for this work. For the exact description of all the functions used in the key generation procedure of the aforementioned schemes, please refer to the reference documentation of the corresponding schemes submitted to the NIST standardization procedure available in [27]. We denote $\mathbb{Z}[X]/(X^n+1)$ and $\mathbb{Z}_q[X]/(X^n+1)$ for $q \in \mathbb{Z}$ as rings R and R_q respectively. We also use $\times$ to denote the formal multiplication between any two entities (matrix-vector, polynomial multiplication in ring), but use $\cdot$ to denote point-wise multiplication of two vectors. We also refer to the elements (matrices of polynomials) in R_q^k as modules and polynomials in R_q using Bold upper case letters. We also use $\mathbf{x} \leftarrow S_\eta$ to denote the module $\mathbf{x}$ whose coefficients lie in the range $[-\eta, \eta]$.

2.2 `NEWHOPE` KEM scheme

The `NEWHOPE` KEM scheme is a triplet of probabilistic algorithms (KeyGen, Encaps, Decaps) that are based on conjectured quantum hardness of the RLWE (Ring Learning With Errors) problem. The `NEWHOPE` KEM scheme consists of both the CPA (Chosen Plaintext Attack) secure and CCA (Chosen Ciphertext Attack) secure variants which are built with the previously proposed NEWHOPE-SIMPLE [3] modelled as a semantically secure public-key encryption (PKE) scheme secure in the CPA model. The ring under operation is $R_q = \mathbb{Z}_q[X]/(X^n+1)$ where n is a power of 2 and q is prime thus allowing the use of the Number

Theoretic Transform (NTT) operation for polynomial multiplication. Refer to Alg.1 for the key generation procedure of the NEWHOPE CPA.PKE scheme. A 32 byte array is chosen as a seed and is expanded into 64 bytes, denoted as z using the SHAKE256 algorithm used as an XOF (Extendable Output) function. The first 32 bytes of z are considered to be the *publicseed* which is used to generate the polynomial $\mathbf{A}$ using the GenA function. The remaining 32 bytes are considered to be then *noiseseed* which are used to generate the polynomials $\mathbf{s}$ and $\mathbf{e}$, but with different nonces. Both the polynomials $\mathbf{s}$ and $\mathbf{e}$ are later converted to the NTT domain using the PolyBitRev function. Further, the RLWE instance is created in the NTT domain as $\hat{\mathbf{b}}$ which is declared to be the public key pk while $\hat{\mathbf{s}}$ is considered to be the secret key sk.

Algorithm 1 NEWHOPE CPA-PKE Key Generation

```
1: procedure NEWHOPE.CPAPKE.GEN()
2:     $seed \leftarrow \{0, \ldots, 255\}^{32}$
3:     $z \leftarrow \mathsf{SHAKE256}(64, seed)$
4:     $publicseed \leftarrow z[0:31]$
5:     $noiseseed \leftarrow z[32:63]$
6:     $\hat{\mathbf{A}} \leftarrow \mathsf{GenA}(publicseed)$
7:     $\mathbf{s} \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(noiseseed, 0))$
8:     $\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$
9:     $\mathbf{e} \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(noiseseed, 1))$
10:    $\hat{\mathbf{e}} \leftarrow \mathsf{NTT}(\mathbf{e})$
11:    $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{A}} \cdot \hat{\mathbf{s}} + \hat{\mathbf{e}}$
12:    Return $(pk = \mathsf{EncodePK}(\hat{\mathbf{b}}, publicseed), sk = \mathsf{EncodePolynomial}(\mathbf{s}))$
```

2.3 KYBER KEM scheme

KYBER (KYBER.CCAKEM) is a MLWE (Module LWE) based IND-CCA2 (Indistinguishability under Chosen Ciphertext Attack) secure key-encapsulation mechanism (KEM). The KYBER.CCAKEM scheme is built in two stages - The KYBER.KEM scheme is based on an IND-CPA secure public key encryption scheme (KYBER.CPAPKE) which is later converted to a CCA-secure KEM scheme based on a slightly tweaked Fujisaki-Okamoto (FO) transform [19]. KYBER.CPAPKE is essentially the LPR encryption scheme introduced by Lyubashevsky, Peikert and Regev in [25], but with extension to module lattices. Thus, the hardness of KYBER is derived from the hardness of solving the MLWE problem [23], while the classical LPR encryption scheme is based on the hardness of solving the RLWE problem. Apart from this, there are other modifications to the scheme like the modified technique for generation of the public module $\mathbf{A}$ as in [4], compressed public key and ciphertexts through "Bit-dropping" using the Learning-with-rounding (LWR) problem [6]. Computation is done over $R_q^k = \mathbb{Z}_q^k[X]/(X^n + 1)$ with $k > 1$ which correlates to module lattices with rank k.

Refer Alg.2 for the key generation procedure of the KYBER.CPAPKE scheme. A 256 bit seed is expanded into two more 256 bit seeds ρ and σ from which ρ is used to generate the public module $\mathbf{A}$ of dimension $k \times k$ wherein each element is present in the ring R_q while σ is used to generate the secret module $\mathbf{s}$ and the error module $\mathbf{e}$ in dimension k and l respectively. Both the secret module $\mathbf{s}$ and error module $\mathbf{e}$ are generated by sampling from a Centered Binomial Distribution denoted as CBD_η where η denotes that each sampled element is present in the range $[-\eta, \eta]$ where $\eta << q$. Randomness required for sampling from CBD_η is derived from a Pseudo Random function (PRF implemented using SHAKE256 as XOF) whose inputs are σ and N where N is the nonce incremented with each sampled polynomial of the module. Module $\mathbf{t}$ is computed as an MLWE instance in the normal domain and is further input to the $\mathsf{Compress}_q$ function to generate the public key pk. The $\mathsf{Compress}_q$ function acts upon every coefficient of $\mathbf{t}$ and rounds it to a lower modulus 2^{d_t}. This Compress$_q$ function is used to reduce the size of the public key and ciphertexts but also provides additional security as it forms an MLWR (Module based Learning With Rounding) instance. A corresponding Decompress$_q$ function is used further in both the encryption and decryption procedures to recover $\mathbf{t}$, but every coefficient of the recovered $\mathbf{t}$ is only a perturbed version of the original $\mathbf{t}$ generated during the key-generation procedure KYBER.CPAPKE.GEN(). Thus, the public key can be assumed to be built based on the hardness of both the MLWE and MLWR problem.

Algorithm 2 KYBER CPA-PKE Key Generation

```
1: procedure KYBER.CPAPKE.GEN()
2:     $d \leftarrow \{0,1\}^{256}$
3:     $(\rho, \sigma) := G(d)$
4:     $N := 0$
5:     for $i$ from 0 to $k-1$ do
6:         for j from 0 to $k-1$ do
7:             $\mathbf{A}[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho||j||i))$
8:     for $i$ from 0 to $k-1$ do
9:         $\mathbf{s}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
10:        $N := N + 1$
11:    for $i$ from 0 to $k-1$ do
12:        $\mathbf{e}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
13:        $N := N + 1$
14:    $\hat{\mathbf{s}} := \mathsf{NTT}(\mathbf{s})$
15:    $\mathbf{t} := \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \cdot \hat{\mathbf{s}}) + \mathbf{e}$
16:    $pk := (\mathsf{Encode}_{d_t}(\mathsf{Compress}_q(\mathbf{t}, d_t))||\rho)$
17:    $sk := \mathsf{Encode}_{13}(\hat{\mathbf{s}}\ mod^+ q)$
18:    Return $(pk, sk)$
```

2.4 FRODO KEM scheme

The FRODO lattice based KEM scheme unlike many lattice schemes bases its hardness on standard lattice assumptions. The FRODO KEM scheme is a CCA secure scheme that has in its core a public key encryption scheme denoted FRODO.CPAPKE, which is a CPA secure scheme whose security is tightly related to the hardness of a corresponding LWE problem. The FRODO.CPAPKE scheme is actually based on the implementation of the Lindner-Peikert scheme [24] with some modifications like pseudo-random generation of the public matrix $\mathbf{A}$ using a small seed, sampling from a near Gaussian distribution and new LWE parameters. The FRODO KEM scheme achieves IND-CCA security by transforming the IND-CPA secure FRODO.CPAPKE scheme using the Fujisaki-Okamoto (FO) transform [19] in the random oracle model. The FRODO.CPAPKE scheme has a very simple structure allowing for easy implementation to also reduce the potential for errors. The modulus q is chosen to be a power of 2 that enables easy reduction through bit masking. The secret and error components are sampled using the inversion sampling technique, otherwise known as the CDT (Cumulative Distribution Table) technique. Operation in the FRODO KEM scheme is done over matrices and vectors and hence has moderately larger running times and bandwidth requirements compared to schemes like NEWHOPE and KYBER which are based on algebraically structured LWE variants, but consists of a simpler design with much better hardness guarantees.

Refer Alg.3 for the key generation procedure of the FRODO.CPAPKE scheme. The public constant matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ matrix is generated from a seed $seed_{\mathbf{A}}$ of length len_A using the procedure Frodo.Gen. Subsequently, both the secret component $\mathbf{S} \in \mathbb{Z}_q^{n \times n}$ and the error component $\mathbf{E} \in \mathbb{Z}_q^{n \times n}$ are generated using the same seed $seed_{\mathbf{E}}$ of length len_E but with different nonces 1 and 2 respectively. The LWE instance generated using these components is declared as the public key of the scheme, while $\mathbf{S}$ remains the secret key.

Algorithm 3 FRODO CPA-PKE Key Generation

```
1: procedure FRODO.CPAPKE.GEN()
2:     $seed_{\mathbf{A}} \leftarrow U(\{0,1\}^{len_A})$
3:     $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(seed_{\mathbf{A}}) \in \mathbb{Z}_q^{n \times n}$
4:     $seed_{\mathbf{E}} \leftarrow U(\{0,1\}^{len_E})$
5:     $\mathbf{S} \leftarrow \mathsf{Frodo.SampleMatrix}(seed_{\mathbf{E}}, n, \bar{n}, T_\chi, 1)$
6:     $\mathbf{E} \leftarrow \mathsf{Frodo.SampleMatrix}(seed_{\mathbf{E}}, n, \bar{n}, T_\chi, 2)$
7:     $\mathbf{B} = \mathbf{A} \times \mathbf{S} + \mathbf{E}$
8:     Return public key $pk \leftarrow (seed_{\mathbf{A}}, \mathbf{B})$ and secret key $sk \leftarrow \mathbf{S}$
```

2.5 DILITHIUM Digital Signature scheme

The DILITHIUM signature scheme, similar to the KYBER KEM scheme is based on the hardness guarantees of MLWE [23] problem. The DILITHIUM signature scheme operates over the same environment (as in R_q^k) as `KYBER` except for different values of n and q. The key generation procedure of `DILITHIUM`, `DILITHIUM.KeyGen()` uses the SHAKE256 from SHA3 as an XOF to generate the public constant $\mathbf{A} \in R_q^{k \times l}$ from a random 256 bit seed ρ. The secret keys $\mathbf{s}_1 \in R_q^{\ell}$ and $\mathbf{s}_2 \in R_q^k$ are generated by expanding another random seed $\rho' \in \{0,1\}^{256}$. While the same ρ' is used for both $\mathbf{s}_1$ and $\mathbf{s}_2$, a nonce value starting from zero and incremented for every sampled polynomial is used as input to the Sample function such that all coefficients of the sampled polynomials are uniformly distributed in the range $[-\eta, \eta]$. The value $\mathbf{t} = \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2$ is computed and is input to the $\mathsf{Power2Round}_q(\mathbf{t}, d)$ function which partitions each coefficient of $\mathbf{t}$ into higher and lower order bits $\mathbf{t}_1$ and $\mathbf{t}_0$ respectively. This function is computed coefficient wise such that $\mathbf{t}_1$ corresponds to the $\lceil log_2(q) \rceil - d$ higher order bits of all the coefficients of $\mathbf{t}$, while $\mathbf{t}_0$ correspond to the d lower order bits of the corresponding coefficients of $\mathbf{t}$. The secret-key sk is $(\rho', \mathbf{s}_1, \mathbf{s}_2, \mathbf{t})$ while the public-key pk is $(\rho, \mathbf{A}, \mathbf{t}_1)$.

Algorithm 4 `DILITHIUM` Key Generation

```
1: procedure DILITHIUM.KeyGen()
2:     $\rho, \rho' \leftarrow \{0,1\}^{256}$
3:     $K \leftarrow \{0,1\}^{256}$
4:     $N := 0$
5:     for $i$ from 0 to $\ell - 1$ do
6:         $\mathbf{s}_1[i] := Sample(PRF(\rho', N))$
7:         $N := N + 1$
8:     for $i$ from 0 to $k - 1$ do
9:         $\mathbf{s}_2[i] := Sample(PRF(\rho', N))$
10:        $N := N + 1$
11:    $\mathbf{A} \sim R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$
12:    Compute $\mathbf{t} = \mathbf{A} \times \mathbf{s}_1 + \mathbf{s}_2$
13:    Compute $\mathbf{t}_1 := \mathsf{Power2Round}_q(\mathbf{t}, d)$
14:    $tr \in \{0,1\}^{384} := \mathsf{CRH}(\rho || \mathbf{t}_1)$
15:    Return $pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
```

2.6 Insecure Instantiations of LWE

The error component in the LWE plays a very crucial part in ensuring its hardness guarantees. There are some instantiations of LWE that are very trivial to solve. For example, if the error component is an all zero vector, then the LWE instance is converted into a set of linear equations with equal number of equations and unknowns. This instance can be solved using straight forward Gaussian elimination. If the error component only has values in a fixed interval $[z + \frac{1}{2}, z - \frac{1}{2}]$, then one can just "round away" the non-integral part and subtract z to remove the error from every sample [29]. There are also other easy instances of LWE, For eg. From a given set of n LWE instances, if k of the n error components add up to zero, then one can simply add the corresponding samples to cancel the error and obtain an error free sample. It is also possible to solve an LWE instance in roughly n^d time and space using n^d samples if the error in the samples lie in a known set of size d [5]. For a very small d, this yields a very practical attack.

In this work, we fault the nonce utilized to generate the secret and error components in such a way, so as to obtain an error vector that is equal to the secret vector in the LWE instance. By doing so, one reduces the LWE instance into an exactly defined set of linear equations (equal number of equations and unknowns) which can be trivially solved by Gaussian elimination. Thus, if the nonce is not suitably protected from implementation attacks, faulting the nonce serves as a single point of failure to trivially retrieve the hidden secret component in the LWE instance.

3 Faulting the nonce

In the key generation procedures of all the four schemes `KYBER`, `NEWHOPE`, `FRODO` and `DILITHIUM`, nonces have been used in the generation of the secret and error components of the LWE instance. They have been used to reduce the amount of randomness required for generating the required components. We will illustrate, in this section, the use of nonces to generate both $\mathbf{s}$ and $\mathbf{e}$ in each of the aforementioned schemes and analyse the effects of faulting the nonce to certain values to generate an easy LWE instance. The code snippets described in this section are all taken from the reference implementations of the corresponding proposals submitted to the NIST standardization process.

3.1 Fault Model

We notice that the aforementioned schemes use a fixed set of nonces. The set of nonces are usually derived from a fixed value like 0 or 1, in order to reduce randomness requirement to generate secret and error components of the LWE instance and associated overheads. Starting from a fixed nonce value, the following nonces are derived by simply an increment operator for every new polynomial generated. This specific choice of nonces, eases the effort of attacker in fault injection. As shown later, we require in the proposed attack that both the secret and error component are maliciously generated with the same nonce.

By using a simple bit flip model, which is widely studied and practically demonstrated on a range of devices [1, 8] with high repeatability, we are able to satisfy this attack requirement. For typical parameters, the attacker only needs to perform a very few number of bit-flips (1-3) to fault the nonce to a desired value. Recently, He et al. [21] realize sophisticated flip-flop bit flips (1-4 bit flips) even on modern Virtex-5 FPGA device through fine-grained laser perturbations. In the following, we assume this model to demonstrate a vulnerability and corresponding key recovery attack on lattice based crypto-systems.

3.2 Vulnerability in NEWHOPE

Please refer to code snippet in Fig.1 that describes the function calls used to sample **s** and **e** that are further used to create the LWE instance for the `NEWHOPE` scheme. While a nonce of 0 is used to generate **s**, a nonce of 1 is used to generate **e** with the *noiseseed* used for generating both the components remaining the same. Thus, if an attacker can perform a single bit flip of the nonce such that both the calls to the function `poly_sample` use the same seed, then it yields $\mathbf{s} = \mathbf{e}$. An LWE instance generated with both **s** and **e** the same, can be very trivially solved using Gaussian elimination. The fault can either be injected before the first call to the function `poly_sample` to yield a nonce of 1 for both the calls or a fault can be injected after the first call to the `poly_sample` function so that the nonce used for generation of **e** is also 0 thus yielding $\mathbf{s} = \mathbf{e}$ in both the cases.

```
/* Sample short vectors shat and ehat */
  poly_sample(&shat, noiseseed, 0);
  poly_ntt(&shat);
  poly_sample(&ehat, noiseseed, 1);
  poly_ntt(&ehat);
```

Fig. 1: Code Snippet showing sampling of **s** and **e** for the NEWHOPE scheme

3.3 Vulnerability in FRODO

Please refer to code snippet in Fig.2 that describes the function calls used to sample **S** and **E** that are further used to create the LWE instance for the `FRODO` scheme. We can see that the nonces are handled in a similar way as in the case of `NEWHOPE`, but only have a different value (i.e 0 and 1 in `NEWHOPE` as opposed to 1 and 2 in `FRODO`). But, both the values of nonces used in FRODO differ by two bits, thus an attacker can attempt to insert faults using multiple ways so as to ensure that the function calls to generate both **S** and **E** use the same value of nonce. This creates an LWE instance which can be trivially solved using Gaussian elimination. The attacker can either inject two faults (i.e) flip the second LSB bit when nonce = 1 and flip the first LSB bit when nonce = 2, thus yielding the same value of nonce = 3 for generation of both **S** and **E**. The attacker can vice versa target the first LSB bit when nonce = 1 and second LSB bit when nonce = 2 to yield a nonce of 0 for generation of both **S** and **E**. Thus, ultimately the attacker has to inject a minimum of just two bit flips in order to create a very easy LWE instance that results in direct retrieval of the secret key **S** through Gaussian elimination.

```
    // Generate S and E from procedure frodo_sample_n
    frodo_sample_n(S, PARAMS_N*PARAMS_NBAR,
                   randomness + CRYPTO_BYTES, CRYPTO_BYTES, 1);
    frodo_sample_n(E, PARAMS_N*PARAMS_NBAR,
                   randomness + CRYPTO_BYTES, CRYPTO_BYTES, 2);
```

Fig. 2: Code Snippet showing sampling of $\mathbf{s}_1$ and $\mathbf{s}_2$ for the KYBER scheme

3.4 Vulnerability in DILITHIUM

Please refer to code snippet in Fig.3 that describes the function calls used to sample $\mathbf{s}_1$ and $\mathbf{s}_2$ that are further used in the MLWE instance for the DILITHIUM signature scheme. One can see that the nonce, starting with the value of 0 is incremented by 1 for every newly generated polynomial, thus ℓ polynomials are sampled for $\mathbf{s}_1$ and k polynomials are sampled for $\mathbf{s}_2$. If one manages to fault the value of nonce to zero after the generation of $\mathbf{s}_1$, then the same set of nonces are used to generate both $\mathbf{s}_1$ and $\mathbf{s}_2$. It is important to note that the number of polynomials in $\mathbf{s}_1$ (ℓ) is less than the number of polynomials in $\mathbf{s}_2$. One can still form $n \times \ell$ equations with $n \times \ell$ unknowns to solve for the unknown coefficients of $\mathbf{s}_1$. But, this attack is only possible if the whole of $\mathbf{t} = \mathbf{A} \times \mathbf{s}_1 + \mathbf{s}_2$ is known to the attacker. But only the higher order bits of $\mathbf{t}$ (i.e) $\mathbf{t}_1$ is known to the attacker, hence it might not be possible to directly solve for $\mathbf{s}_1$ through Gaussian elimination. But, it is also not clear if the lower order bits of $\mathbf{t}$ (i.e) $\mathbf{t}_0$ are leaked through signature outputs of the `DILITHIUM` signature scheme. Hence, if the whole of $\mathbf{t}$ can be reconstructed by observing a sufficient amount of signatures for the same $\mathbf{s}_1$ and $\mathbf{s}_2$, then one can easily retrieve $\mathbf{s}_1$ through Gaussian elimination.

```
/* Sample short vectors s1 and s2 */
  unsigned char nonce=0;
  for(i = 0; i < l; ++i)
    poly_uniform_eta(&s1.vec[i], rhoprime, nonce++);
  for(i = 0; i < k; ++i)
    poly_uniform_eta(&s2.vec[i], rhoprime, nonce++);
```

Fig. 3: Code Snippet showing sampling of $\mathbf{s}_1$ and $\mathbf{s}_2$ for the DILITHIUM scheme

3.5 Vulnerability in KYBER

Please refer to code snippet in Fig.4 that describes the function calls used to sample $\mathbf{s}_1$ and $\mathbf{s}_2$ that are further used to create the MLWE instance during key generation in the `KYBER` CPA-PKE scheme. A similar fault vulnerability as in `DILITHIUM` exists in the key generation procedure of `KYBER` that can ensure that

$\mathbf{s}_1 = \mathbf{s}_2$. In KYBER, the dimensions of both $\mathbf{s}_1$ and $\mathbf{s}_2$ are the same and equal k. But, the generated MLWE instance $\mathbf{t} = \mathbf{a} \times \mathbf{s}_1 + \mathbf{s}_2$ is further protected by the hardness of the MLWR problem through the Compress_q function and hence the public key is formed by a combination of MLWE and MLWR instances. Thus, an attacker cannot deduce the output after the creation of the MLWE instance since, only the output of the Compress_q function is revealed as the public key, which is a combination of both the LWE and LWR instance. Thus, by inducing the same fault in the nonce as in the DILITHIUM signature scheme, the attacker can ensure that both the secret and error components are same, thus removing the hardness derived from the MLWE problem. But, the scheme is still protected by the LWR instance. Hence, we can conclude that though the induced fault poses as a vulnerability for KYBER, it does not result in complete breakdown of the scheme or in key recovery, but only weakens the scheme considerably.

```
/* Sample short vectors s and e */
  unsigned char nonce=0;
  for(i=0;i<k;i++)
    poly_getnoise(skpv.vec+i, noiseseed, nonce++);
  polyvec_ntt(&skpv);
  for(i=0;i<k;i++)
    poly_getnoise(e.vec+i, noiseseed, nonce++);
```

Fig. 4: Code Snippet showing sampling of $\mathbf{s}_1$ and $\mathbf{s}_2$ for the KYBER scheme

4 Countermeasures

In all the aforementioned key generation procedures of the corresponding schemes, we have seen that the nonce is more or less hard-coded and is incremented, starting from 0 for every polynomial that is generated. Since the nonce is deterministically generated, one can use an error correction scheme to check if the correct value of nonce is being used for the generation of polynomials. This will ensure that the same nonce is not used to generate the secret and error components thus thwarting the generation of a straightforward LWE instance, whose secret can be solved for using Gaussian elimination. The motivation to use a nonce for generation of polynomials is mainly to reduce the randomness requirement and use the same seed to generate multiple polynomials. Thus, one can simply forego the use of the nonce and generate a new seed for the generation of every polynomial generated, though this might yield a considerable performance overhead for the key generation procedure. This is especially true in the case of schemes like KYBER, DILITHIUM which work with module lattices as multiple polynomials have to be generated for a single component, thus requiring generation of multiple seeds.

5 Conclusion

In this short paper, we point out to some fault vulnerabilities that exist in various lattice based schemes like `NEWHOPE`, `KYBER` and `FRODO` CCA2 secure KEM schemes and the `DILITHIUM` signature scheme. Nonces are used to reduce the randomness requirement to generate the public and secret keys during key generation. The fixed nonces that are used are easily vulnerable to faults and we have shown that a single to very few bit flips on the nonce will result in complete key recovery in the `NEWHOPE` and `FRODO` scheme. The nonces are faulted to ensure that the same nonces are used to generate both the secret and error components of the LWE instance and this can lead to trivial key recovery through Gaussian elimination. The same attack with suitable modifications can also be applied to the `DILITHIUM` signature scheme given that the complete output of the generated LWE instance ($\mathbf{t} \in R_q^k$) can be reconstructed through observation of a sufficient number of signatures. But, the same attack on `KYBER` cannot result in complete key recovery as the public key is formed as a combination of both the LWE and LWR instance, but we believe that our fault removes the hardness derived from the LWE problem. The aforementioned fault vulnerabilities associated with the nonce only occur due to the implementation strategies used and hence can we believe can be easily corrected albeit with a small performance overhead.

References

1. Agoyan, M., Dutertre, J.M., Mirbaha, A.P., Naccache, D., Ribotta, A.L., Tria, A.: How to flip a bit? In: On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International. pp. 235–239. IEEE (2010)
2. Albrecht, M., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions. In: Annual Cryptology Conference. pp. 153–178. Springer (2016)
3. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Newhope without reconciliation. IACR ePrint 1157, 2016 (2016)
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-Quantum Key Exchange-A New Hope. In: USENIX Security Symposium. pp. 327–343 (2016)
5. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: International Colloquium on Automata, Languages, and Programming. pp. 403–415. Springer (2011)
6. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. Advances in Cryptology–EUROCRYPT 2012 pp. 719–737 (2012)
7. Barends, R., Kelly, J., Megrant, A., Veitia, A., Sank, D., Jeffrey, E., White, T.C., Mutus, J., Fowler, A.G., Campbell, B., et al.: Superconducting quantum circuits at the surface code threshold for fault tolerance. Nature 508(7497), 500–503 (2014)
8. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pellicioli, M., Pelosi, G.: Fault attack on aes with single-bit induced faults. In: Information Assurance and Security (IAS), 2010 Sixth International Conference on. pp. 167–172. IEEE (2010)
9. Bos, J., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1006–1018. ACM (2016)

10. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Tech. rep. (2017)
11. Bruinderink, L.G., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload–a cache attack on the bliss lattice-based signature scheme. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 323–345. Springer (2016)
12. Campbell, P., Groves, M., Shepherd, D.: Soliloquy: A cautionary tale. In: ETSI 2nd Quantum-Safe Crypto Workshop. pp. 1–9 (2014)
13. Chen, H., Lauter, K., Stange, K.E.: Attacks on search RLWE. https://www.microsoft.com/en-us/research/publication/attacks-on-search-rlwe/ (2015)
14. Cramer, R., Ducas, L., Peikert, C., Regev, O.: Recovering short generators of principal ideals in cyclotomic rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 559–585. Springer (2016)
15. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehle, D.: Crystals–dilithium: Digital signatures from module lattices
16. Elias, Y., Lauter, K.E., Ozman, E., Stange, K.E.: Provably weak instances of Ring-LWE. In: Annual Cryptology Conference. pp. 63–92. Springer (2015)
17. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Loop abort faults on lattice-based fiat-shamir & hash'n sign signatures. IACR Cryptology ePrint Archive 2016, 449 (2016)
18. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Side-channel attacks on bliss lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1857–1874. ACM (2017)
19. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual International Cryptology Conference. pp. 537–554. Springer (1999)
20. Harty, T., Allcock, D., Ballance, C.J., Guidoni, L., Janacek, H., Linke, N., Stacey, D., Lucas, D.: High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit. Physical review letters 113(22), 220501 (2014)
21. He, W., Breier, J., Bhasin, S., Jap, D., Ong, H.G., Gan, C.L.: Comprehensive laser sensitivity profiling and data register bit-flips for cryptographic fault attacks in 65 nm fpga. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 47–65. Springer (2016)
22. Ishiguro, T., Kiyomoto, S., Miyake, Y., Takagi, T.: Parallel Gauss sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In: International Workshop on Public Key Cryptography. pp. 411–428. Springer (2014)
23. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography 75(3), 565–599 (2015)
24. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: CT-RSA. pp. 319–339 (2011)
25. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM 60(6), 43 (2013)
26. NIST: Post-quantum crypto project. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/` (2016), accessed: 23.11.2017
27. NIST: Post quantum cryptography - round 1 submissions. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions/` (2017)
28. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation

29. Peikert, C.: How (not) to instantiate ring-lwe. In: International Conference on Security and Cryptography for Networks. pp. 411–430. Springer (2016)
30. Pessl, P.: Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In: Progress in Cryptology–INDOCRYPT 2016: 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings 17. pp. 153–170. Springer (2016)
31. Pessl, P., Bruinderink, L.G., Yarom, Y.: To bliss-b or not to be: Attacking strongswan's implementation of post-quantum signatures. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1843–1855. ACM (2017)
32. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 353–370. Springer (2014)
33. Preskill, J.: Reliable quantum computers. In: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences. vol. 454, pp. 385–410. The Royal Society (1998)