Secure Sampling of Constant-Weight Words – Application to BIKE

Nicolas Sendrier

Inria
nicolas.sendrier@inria.fr

Abstract. The pseudo-random sampling of constant weight word, as it is currently implemented in schemes like BIKE or HQC, is prone to the leakage of information on the seed being used. This creates a vulnerability when the semantic security conversion requires a deterministic re-encryption. This observation was first made in [HLS21] about HQC, and a timing attack was presented to recover the secret key. As suggested in [HLS21] a similar attack applies to BIKE and an instance of such an attack is presented here, as well as countermeasures similar to those suggested in [HLS21] for HQC.
A new approach for fixing the issue is also proposed. It is first remarked that, contrary to what is done currently, the sampling of constant weight words doesn't need to produce a uniformly distributed output. If the distribution is close to uniform in the appropriate metric, the impact on security is negligible. Also, a new variant of Fisher-Yates shuffle is proposed which is (1) very well suited for secure implementations against timing and cache attacks, and (2) produces constant weight words with a distribution close enough to uniform.

1 Introduction

In a recent work [HLS21], a timing attack on an implementation of HQC [AMAB$^+$21] is described. This attack exploits the fact that the timing for sampling words of constant weight depends on the seed used to initialized the pseudo-random generator. In conjunction, the Fujisaki-Okamoto transformation, which is used to provide semantic security, uses secret data for this seed. This can be used to mount a successful key recovery attack on HQC. As suggested in [HLS21], the same vulnerability applies to BIKE. We give hereafter a possible instance of attack using this same idea.

We first review the process used in BIKE for pseudo-random sampling constant weight words and we observe that its timing variation induces a vulnerability which could potentially be exploited as in [HLS21], if key reuse is allowed, to recover either a message or the secret key. The standard countermeasure consists in modifying the sampling to make sure it runs in constant time. We give an estimate of the corresponding overhead for BIKE.

The main contribution of this paper consists in exploring a new possibility. Part of the difficulty of the sampling comes from the fact that uniform distribution is a requirement. We will show it is not the case, and sampling constant weight words with a distribution close enough to uniform (in a metric that we will specify) has no impact on security. We will exhibit a variant of the Fisher-Yates shuffle, which we believe to be new and which is particularly well suited to secure implementation. This new variant can be implemented in constant time and with a fixed memory access pattern to sample constant weight words from uniformly distributed random bits, with a distribution which is close enough to uniform to have a negligible impact on security.

Algorithm 1 Current BIKE Constant Weight Word Sampling

```
Input: $n$, $t$, seed
Output: $t$ distinct elements of $\{0,\dots,n-1\}$
1: prng $\leftarrow$ prng_init(seed)
2: $i \leftarrow 0$
3: while $i < t$ do
4:     $j \leftarrow$ rand($n$, prng)
5:     if $j \notin$ pos then
6:         pos[$i$] $\leftarrow j$
7:         $i \leftarrow i + 1$
8: return pos
```

```
rand($n$, prng) :
1: repeat
2:     $x \leftarrow$ randbits($B$, prng)
3:     $x \leftarrow x$ & mask
4: until $x < n$
5: return $x$
```

2 Sampling Constant-Weight Words in BIKE

In its current specification, BIKE uses (essentially) Algorithm 1 to sample a table of t distinct indices in $\{0,1,\dots,n-1\}$. Further treatment to change that into a constant-weight word (sorting or producing a binary word or just keeping a table of indices) is not considered here. In practice, the state-of-the-art implementation of BIKE [DGK] uses a constant-time routine, embedded in Algorithm 1, which sets the bit corresponding to each accepted position in a binary word of length n. The function **rand**(n, prng) produces a random integer uniformly distributed in $\{0,1,\dots,n-1\}$ from uniformly distributed integers in $\{0,1,\dots,2^B-1\}$ produced by **randbits**(B, prng). The pseudorandom number generator prng is initialized with seed. In BIKE, mask $= 2^u - 1$ where u is the smallest integer such that $n \le 2^u$, and $B = 32$. Those quantities are hard-coded for each given parameter set. In BIKE specification, it is suggested to use AES in counter mode as prng. Whatever is used for prng, it is assumed that **randbits**(B, prng) is constant-time for a given B and behaves as a random oracle producing uniformly distributed bits. The situation is very similar to what is observed in [HLS21], and, as for HQC, the sampling process has a variable running time for two reasons:

1. In **rand**(), the index x is rejected with a probability that can reach 50%. For BIKE parameters the rejection probability varies from 25% to 38%. The number of iterations is a random number ranging from 1 to a few units.
2. In the main algorithm, any duplicate position is rejected (instruction 5) and causes an additional execution of the while loop.

Overall, the timing for the constant-weight word sampling will depend on the total number of calls to **randbits**() which will vary with seed.

2.1 BIKE Specification

The specification of BIKE is given in Table 1. The parameters are the block length r (the code length is $n = 2r$), the row weight w, the error weight t (with $w \approx t \approx \sqrt{n}$), and the message size ℓ. In addition, a mapping $\texttt{decoder} : \mathcal{R} \times \mathcal{H}_w \to \mathcal{R}^2 \cup \{\perp\}$ must be defined such that $\texttt{decoder}(e_0h_0 + e_1h_1, h_0, h_1) = (e_0, e_1)$ with high probability when $(e_0, e_1) \in \mathcal{E}_t$ and $(h_0, h_1) \in \mathcal{H}_w$ are drawn uniformly at random.

2.2 Timing Attacks on BIKE

The attack of [HLS21] on HQC can easily be adapted for BIKE. The exact mechanism and the way the timing information is used may differ, but the principle is the same.

Notation	
$\mathbb{F}_2$:	Binary finite field
$\mathcal{R}$:	Cyclic polynomial ring $\mathbb{F}_2[X]/(X^r - 1)$
$\mathcal{H}_w$:	Private key space $\{(h_0, h_1) \in \mathcal{R}^2 \mid \lvert h_0\rvert = \lvert h_1\rvert = w/2\}$
$\mathcal{E}_t$:	Error space $\{(e_0, e_1) \in \mathcal{R}^2 \mid \lvert e_0\rvert + \lvert e_1\rvert = t\}$
$\mathcal{M}$:	Message space $\{0,1\}^\ell$
$\mathcal{K}$:	Session key space $\{0,1\}^\ell$
$\mathbf{H}$:	Constant weight word (CWW) sampler $\mathbf{H} : \mathcal{M} \to \mathcal{E}_t$
$\mathbf{L}$:	Hash function $\mathbf{L} : \mathcal{R}^2 \to \mathcal{M}$
$\mathbf{K}$:	Key derivation function $\mathbf{K} : \mathcal{M} \times \mathcal{R} \times \mathcal{M} \to \mathcal{K}$
$\perp$:	Decoding failure
$\lvert g\rvert$:	Hamming weight of a binary polynomial $g \in \mathcal{R}$
$u \overset{\$}{\leftarrow} U$:	Variable u is sampled uniformly at random from the set U
$u \overset{\mathcal{D}}{\leftarrow} U$:	Variable u is sampled from the set U according to distribution $\mathcal{D}$

```
KeyGen : () ↦ (h_0, h_1, σ), h
Output: $(h_0, h_1, \sigma) \in \mathcal{H}_w \times \mathcal{M}$, $h \in \mathcal{R}$
 1: $(h_0, h_1) \overset{\$}{\leftarrow} \mathcal{H}_w$
 2: $h \leftarrow h_1 h_0^{-1}$
 3: $\sigma \overset{\$}{\leftarrow} \mathcal{M}$
```

```
Encaps : $h \mapsto K, c$
Input: $h \in \mathcal{R}$
Output: $K \in \mathcal{K}$, $c \in \mathcal{R} \times \mathcal{M}$
 1: $m \overset{\$}{\leftarrow} \mathcal{M}$
 2: $(e_0, e_1) \leftarrow \mathbf{H}(m)$
 3: $c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
 4: $K \leftarrow \mathbf{K}(m, c)$
```

```
Decaps : $(h_0, h_1, \sigma), c \mapsto K$
Input: $((h_0, h_1), \sigma) \in \mathcal{H}_w \times \mathcal{M}$, $c = (c_0, c_1) \in \mathcal{R} \times \mathcal{M}$
Output: $K \in \mathcal{K}$
 1: $e' \leftarrow \texttt{decoder}(c_0 h_0, h_0, h_1)$          ▷ $e' \in \mathcal{R}^2 \cup \{\perp\}$
 2: $m' \leftarrow c_1 \oplus \mathbf{L}(e')$          ▷ with the convention $\perp = (0, 0)$
 3: if $e' = \mathbf{H}(m')$ then $K \leftarrow \mathbf{K}(m', c)$ else $K \leftarrow \mathbf{K}(\sigma, c)$
```

Table 1: The BIKE Key Encapsulation Mechanism

The attacks presented here do not apply to the ephemeral key setting but only to scenarii where secret keys can be reused. The adversary is allowed to make multiple request, with the same secret key, to a decapsulation oracle. The attacks were not implemented nor fully analyzed, the purpose of this paper is to state that the vulnerabilities presented in [HLS21] affect BIKE to some extend, and to propose countermeasures.

In BIKE, each call to **Decaps** will issue a call to $\mathbf{H}(m')$ (using the notation in BIKE specification, Table 1), where $m' = c_1 \oplus \mathbf{L}(e')$ and e' is the result of the decoding of c_0. The call to $\mathbf{H}(m')$ will sample a constant-weight word using m' as seed.

The adversary first seeks a message $\tilde{m} \in \{0,1\}^\ell$ such that the call $\mathbf{H}(\tilde{m})$ produces a remarkable timing. Typically such that the number of calls to **randbits**() is very high, so that any call to $\mathbf{H}(\tilde{m})$ can be detected from the timing. Because of the random oracle assumption, this can only be done by brute force. We assume that the adversary was able to perform this task and knows such a message $\tilde{m}$.

In the next step the adversary produces a fake encapsulation (c_0, c_1) with $c_1 = \tilde{m} \oplus \mathbf{L}(\perp)$ and any choice of c_0 (it could be $c_0 = e_0 + he_1$ for some chosen error $e = (e_0, e_1)$ or anything else). This encapsulation is submitted to **Decaps** and the timing of the execution of $\mathbf{H}()$ is measured. From the timing, the adversary can distinguish the case of a decoding failure, where $\mathbf{H}(c_1 \oplus \mathbf{L}(\perp)) = \mathbf{H}(\tilde{m})$ is called, from the case of a successful decoding, where $\mathbf{H}(c_1 \oplus \mathbf{L}(e))$ is called.

Note that this decapsulation almost certainly fails, but the adversary can nevertheless measure the execution time.

Several attacks are possible from there:

1. *A message recovery attack.* The adversary wants to decode some $c_0 = e_0 + e_1 h$ for an unknown error (e_0, e_1). Several small modifications of c_0 are submitted and by observing which decoding fail and which succeed, the adversary gains information on (e_0, e_1). This leads to a variant of the reaction attack [KI00].
2. *A key recovery attack.* The adversary can mount a GJS-like attack by chosing (e_0, e_1) with a specific distance spectrum (see [GJS16] for details).

2.3 Countermeasures

Those attacks both use the timing of $\mathbf{H}()$ to guess whether or not the decoder failed. Being able to do that enables the various forms of reaction attacks mentionned above. To completely avoid that, the simplest (only?) way is to make sure that $\mathbf{H}()$ runs in time independent of its input.

Constant Time Sampling. Since the running time variation comes from the number N of calls to **randbits**(), the solution would be to estimate the distribution of N when seed varies, choose a value N_0 such that only a negligible proportion of the calls to $\mathbf{H}()$ require more than N_0 calls to **randbits**(), and finally implement the sampler so that it makes exactly N_0 calls to **randbits**(). This would lead to a relatively high computational overhead because the rejection probability is high and so is the standard deviation of N. An alternative is to use **rand**() as in HQC (Table 2). It leads to a smaller overhead.

```
rand(n, prng) :
1: repeat
2:     $x \leftarrow$ randbits($B$, prng)
3: until $x < \lfloor 2^B/n \rfloor n$
4: return $x \mod n$
```

Table 2: Uniform Sampling in $\{0, ..., n-1\}$ with Lower Rejection Rate

The following proposition gives the generating function of the distribution of the number of calls to **randbits**() for sampling one constant-weight word:

Proposition 1. *Let N be the random variable for the number of calls to* **randbits**() *in Algorithm 1. We have the following series identity*

$$S(X) = \sum_{\ell=1}^{\infty} \Pr[N = \ell] X^{\ell} = \prod_{i=0}^{t-1} \frac{(n-i)(1-\pi)X}{n - ((n-i)\pi + i)X}$$

where π is the probability to reject a call to **randbits**() *in* **rand**().

A proof is given in appendix.

		rand() with `mask`			**rand**() with mod		
n	t	π	$N_0 - t$	$\frac{N_0-t}{t}$	π	$N_0 - t$	$\frac{N_0-t}{t}$
24 646	134	0.248	193	144%	$1.7\,10^{-6}$	26	19%
49 318	199	0.247	289	145%	$5.1\,10^{-6}$	37	19%
81 194	264	0.381	600	227%	$1.1\,10^{-5}$	48	18%

Table 3: Overheads for a Constant-Time Implementation of Algorithm 1

Overhead for Constant-Time. The generating function of Proposition 1 provides a convenient way to estimate the probabilities $\Pr[N = \ell]$. In Table 3 we give, for BIKE parameters, the overheads corresponding to the two variants of **rand**() (π is the rejection probability in both situations), the value of N_0 is chosen such that the probability to make more than N_0 calls to **randbits**() is smaller than $2^{-\lambda}$, where λ is the security parameter (respectively 128, 192, and 256 for the three rows Table 3). The countermeasures proposed in [HLS21] are essentially of the same nature and lead to similar overheads in the case HQC.

Note that the cost for sampling constant-weight words is far from dominant, and compared to the full cost of **Encaps** or **Decaps**, this overhead would have a very limited impact on the global efficiency. Still, the patched version of Algorithm 1 would be probabilistic, and moreover the access to the table pos[] would depend on the instance, making it vulnerable to cache attacks and requiring a careful implementation, possibly involving another overhead.

Cost for Producing a Distinguishable Message. Proposition 1 can also help to determine what we could reasonably expect for the distinguishable message $\tilde{m}$ in the attack. For instance, for the first set of parameters, with Algorithm 1, the average number of calls to **randbits**() is $\bar{N} = 178$ while obtaining a message $\tilde{m}$ such that $N = 237$ (33% overhead) would require about 2^{64} calls to **H**(). If **rand**() is as in HQC (Table 3), most of the time there is no additional call to **randbits**(), and one message out of 2^{64} will be such that $N = t + 15$ (11% overhead). Other parameter sets give numbers of similar magnitude. Whether or not this enough to make the timing difference measurable, and thus the attack effective, will depend on the adversary's capabalities, but, nevertheless, the threat is significant enough to justify the need for a constant-time sampling.

Relaxing the Uniformity Condition. There is another possible approach. To comply with the security reduction given in the BIKE specification document, it is required that constant weight words are produced with a uniform distribution. However, we show in the next section that if instead the distribution is only close to uniform, the impact on security is negligible. We will see that a variant of the Fisher-Yates algorithm which is suitable for constant time implementation can be devised to produce constant-weight words with a distribution close enough to uniform.

3 Towards Constant-Time Variants of the Fisher-Yates Shuffle

An advantage of the Fisher-Yates algorithm is that it doesn't need to reject duplicate indices. This cancels one source of running time variation. The other source of variation, the rejection sampling for uniform random integers is still there, but we will see that removing this rejection creates a bias whose impact on security is negligible.

Using the Fisher-Yates shuffle for cryptographic implementations has sometimes been discarded because of its data-dependent memory access. We give here a variant with a fixed memory access pattern. Moreover, because the space complexity is proportional to t rather than n, this variant is well adapted to the sampling of words of fixed weight t much smaller than the block length n.

3.1 Fisher-Yates Shuffle

In its usual form, the Fisher-Yates algorithm inputs, n and t, are equal and a uniformly distributed random permutation of n elements is returned. In the version given in Algorithm 2, it returns a

Algorithm 2 Fisher-Yates Algorithm

```
Input: $n$, $t$
Output: $t$ distinct elements of $\{0, \ldots, n-1\}$
1: pos $\leftarrow [0, 1, \ldots, n-1]$
2: **for** $i = 0$ **to** $t-1$ **do**
3:     $\ell \overset{\$}{\leftarrow} \{i, \ldots, n-1\}$
4:     swap pos$[i]$ and pos$[\ell]$
5: **return** pos$[0], \ldots,$ pos$[t-1]$
```

table of t distinct elements of $\{0, \ldots, n-1\}$, that is a constant weight word, also with a uniform distribution. The algorithm is ill-suited to the case where t is small compared to n and to secure implementation in the context of cryptographic implementation, mostly because of the of need to make data-dependent access in a large table.

Mathematical Abstraction. If we denote S_i the random number drawn at the i-th iteration (instruction 3 of Algorithm 2), the Fisher-Yates algorithm draws t random transpositions $\mu_i = (i\ S_i)$ with $i \leq S_i < n$ and returns the images of $0, 1, \ldots, t-1$ by the permutation

$$\sigma = \mu_0 \circ \mu_1 \circ \cdots \circ \mu_{t-1} \text{ (where } \mu_i = (i\ S_i))$$

obtained as the product of those t transpositions. In fact, to obtain the t indices $\mathrm{pos}[i] = \sigma(i)$, $0 \leq i < t$, there is no need for a table of size n nor for data-dependent access to a data structure. Note that $\mu_i(j) = j$ if $j < i$, so to compute $\sigma(i)$ we may ignore the transpositions μ_j for $j > i$. We have:

$$\begin{aligned}
\mathrm{pos}[0] &= \sigma(0) = \mu_0(0) = S_0 \\
\mathrm{pos}[1] &= \sigma(1) = \mu_0 \circ \mu_1(1) = \mu_0(S_1) \\
&\vdots \qquad \vdots \\
\mathrm{pos}[i] &= \sigma(i) = \mu_0 \circ \cdots \circ \mu_{i-1} \circ \mu_i(i) = \mu_0 \circ \cdots \circ \mu_{i-1}(S_i) \\
&\vdots \qquad \vdots
\end{aligned}$$

The above formulas can be evaluated column-wise or row-wise and lead to the variants proposed in the next section.

Two Variants of Fisher-Yates Shuffle. The variants described in Algorithm 3 and Algorithm 4 will sample the transpositions, and then either iterate on the transpositions, starting from the end, updating all the images $\mathrm{pos}[j], 0 \leq j < t$ of $0, \ldots, t-1$, or iterate on the elements $i \in \{0, \ldots, t-1\}$, applying all transpositions, starting from the end, to each i to obtain $\mathrm{pos}[i]$. Note that most loops are in reverse order. This is needed, either to apply the transposition in

Algorithm 3 Fisher-Yates Algorithm – Variant

```
Input: $n$, $t$
Output: $t$ distinct elements of $\{0, \ldots, n-1\}$
1: for $i = 0$ to $t-1$ do
2:     $\mathrm{pos}[i] \xleftarrow{\$} \{i, \ldots, n-1\}$
3: for $i = t-1$ downto $0$ do
4:     for $j = i+1$ to $t-1$ do
5:         $\mathrm{pos}[j] \leftarrow (\mathrm{pos}[j] = \mathrm{pos}[i])\ ?\ i\ :\ \mathrm{pos}[j]$
6: return $\mathrm{pos}[0], \ldots, \mathrm{pos}[t-1]$
```

Algorithm 4 Fisher-Yates Algorithm – Variant

```
Input: $n$, $t$
Output: $t$ distinct elements of $\{0, \ldots, n-1\}$
1: for $i = 0$ to $t-1$ do
2:     $\mathrm{pos}[i] \xleftarrow{\$} \{i, \ldots, n-1\}$
3: for $i = t-1$ downto $0$ do
4:     for $j = i-1$ downto $0$ do
5:         $\mathrm{pos}[i] \leftarrow (\mathrm{pos}[i] = \mathrm{pos}[j])\ ?\ j\ :\ \mathrm{pos}[i]$
6: return $\mathrm{pos}[0], \ldots, \mathrm{pos}[t-1]$
```

the correct order (outer loop in Algorithm 3 and inner loop in Algorithm 4), or to allow the computation in place (outer loop of Algorithm 4). At the i-th iteration, Algorithm 3 applies the transposition μ_i to update the whole table pos[]. Algorithm 4 has a slightly different logic with a simpler pattern for writing: at iteration i, only the i-th entry of the table is modified. Also, if they use the same randomness, Algorithms 2, 3, and 4 return exactly the same value. So it is possible to choose the most suitable variant depending on the platform and/or security requirement while keeping full interoperability.

Secure Implementation. A key feature of Algorithm 3 and Algorithm 4 is that the pattern of access to the table pos[] is independent of the data, which brings a natural imunity to timing and cache attacks. On the other hand, the complexity is quadratic (in t), but in practice, as far as secure implementation is concerned, it is also the case of Algorithm 1 because of the collision check (instruction 5), and of Algorithm 2 because a secure implementation of the swap (instruction 4) will require an overhead of at least the same magnitude.

3.2 Relaxing the Distribution

In Algorithm 5 the rejection sampling is removed from **rand**(), so it runs in constant time. On the other hand, the distribution of its output is no longer uniform. This algorithm derives from Algorithm 3 but Algorithm 4 could be used as well. With Algorithm 3 (or Algorithm 4), each possible output, an array of t distinct integers in $\{0, \ldots, n-1\}$, is obtained with the same probability

$$\pi' = \prod_{i=0}^{t-1} \frac{1}{n-i}.$$

With the integers drawn as in Algorithm 5, the maximal probability over all possible outputs is

$$\pi_{\max} = \prod_{i=0}^{t-1} \frac{1}{n-i} \left(1 + \frac{(n-i) - (2^B \mod (n-i))}{2^B}\right).$$

Algorithm 5 Fisher-Yates Algorithm – Non Uniform

```
Input: $n$, $t$, seed
Output: $t$ distinct elements of $\{0,\dots,n-1\}$
 1: prng ← prng_init(seed)
 2: for $i = 0$ to $t-1$ do
 3:     $\mathrm{pos}[i] \leftarrow i + \mathbf{rand}(n-i, \mathsf{prng})$
 4: for $i = t-1$ downto 0 do
 5:     for $j = i+1$ to $t-1$ do
 6:         $\mathrm{pos}[j] \leftarrow (\mathrm{pos}[j] = \mathrm{pos}[i])\ ?\ i\ :\ \mathrm{pos}[j]$
 7: return $\mathrm{pos}[0],\dots,\mathrm{pos}[t-1]$
```

```
rand($n$, prng) :
 1: $x \leftarrow \mathbf{randbits}(B, \mathsf{prng})$
 2: return $x \mod n$
```

and the minimal probability over all possible outputs is

$$\pi_{\min} = \prod_{i=0}^{t-1} \frac{1}{n-i}\left(1 - \frac{2^B \mod (n-i)}{2^B}\right).$$

For BIKE parameters, the ratios $\tau_{\min} = \pi_{\min}/\pi'$ and $\tau_{\max} = \pi_{\max}/\pi'$ are very close to 1 (see Table 4). As shown next, this is close enough to the uniform distribution to have a negligible

$B=32$				$B=24$			
n	t	$\tau_{\min}$	$\tau_{\max}$	n	t	$\tau_{\min}$	$\tau_{\max}$
24 646	134	0.99962	1.00038	24 646	134	0.91	1.11
49 319	199	0.9989	1.0011	49 319	199	0.76	1.37
81 194	264	0.9975	1.0025	81 194	264	0.51	1.85

Table 4: Bias Between the Uniform Distribution and the Output of Algorithm 5

impact on security.

Finally, remark that each constant weight word can be obtained from $t!$ distinct and independent outputs of the algorithm, leading to an unsurprising probability $t!\pi' = 1/\binom{n}{t}$ in the uniform case. In the non uniform case, the minimal probability is at least $t!\pi_{\min}$ and the maximal probability is at most $t!\pi_{\max}$. If we denote $\mathcal{D}$ the distribution over the sample space $\mathcal{E}_t$ which stems from Algorithm 5

$$\tau_{\min} \le \frac{\Pr\left[e \mid e \xleftarrow{\mathcal{D}} \mathcal{E}_t\right]}{\Pr\left[e \mid e \xleftarrow{\$} \mathcal{E}_t\right]} \le \tau_{\max} \tag{1}$$

3.3 Security Reduction

This section relates to the IND-CCA proof of BIKE, available in [AAB+21, §C] and deriving from [HHK17]. We give below a sketch of why the security is not reduced when the output distribution $\mathcal{D}$ of $\mathbf{H}()$ is close to uniform instead of uniform. More details are given in §B. The security proof of BIKE cares about the distribution of the errors on two occasions: (1) for the computational hardness of decoding, and (2) for the so-called correctness error, that is the decoding failure rate. In both cases the corresponding terms in the reduction (see [AAB+21, Theorem 3,§C.3] and [HHK17, Theorem 3.2 and 3.4]) are averaged over all error patterns $e \in \mathcal{E}_t$. Now, for any real-valued random variable $V : \mathcal{E}_t \to \mathbb{R}$, we have

$$\sum_{e \in \mathcal{E}_t} \Pr\left[e \mid e \overset{\mathcal{D}}{\leftarrow} \mathcal{E}_t\right] V(e) \leq \tau_{\max} \cdot \sum_{e \in \mathcal{E}_t} \Pr\left[e \mid e \overset{\$}{\leftarrow} \mathcal{E}_t\right] V(e). \tag{2}$$

It follows that the two terms mentionned above, and thus the advantage of any adversary when the error distribution changes from uniform to $\mathcal{D}$, cannot increase by a factor larger than $\tau_{\max}$, as defined in (1).

4 Conclusion

We have shown here that the vulnerabilty of constant weight word sampling presented in [HLS21] for HQC, indeed applies to BIKE when key reuse is allowed. The timing variation can be analyzed and the overhead for constant time implementation estimated.

We have proposed another approach for BIKE's constant weight word sampling based on the Fisher-Yates shuffle and which is original in two respects. First, and contrary to common belief about Fisher-Yates shuffle, our variant is very well suited for secure implementation against timing and cache attacks. Second, we allow our sampler to produce its output with a non uniform distribution, but close enough to uniform to have no effective impact on the global security of the BIKE scheme.

Even though our new proposed algorithm for constant weight word sampling has a higher algorithmic complexity, we believe that for secure implementation, it offers an interesting, possibly even advantageous, trade-off.

References

AAB$^+$21. Carlos Aguilar Melchor, Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Güneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, and Gilles Zémor. BIKE. Round 3 Submission to the NIST Post-Quantum Cryptography Call, v. 4.2, September 2021.

AMAB$^+$21. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and Jurjen Bos. Optimized implementation of HQC, June 2021. `https://pqc-hqc.org/download.php?file=hqc-optimized-implementation_2021-06-06.zip`.

DGK. Nir Drucker, Shay Gueron, and Dusan Kostic. Optimized constant-time implementation of BIKE. `https://github.com/awslabs/bike-kem`.

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO*, volume 1666 of *LNCS*, pages 537–554. Springer, 1999.

GJS16. Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 789–815, 2016.

HHK17. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017.

HLS21. Clemens Hlauschek, Norman Lahr, and Robin Leander Schröder. On the timing leakage of the deterministic re-encryption in HQC KEM. Cryptology ePrint Archive, Report 2021/1485, 2021. `https://ia.cr/2021/1485`.

KI00. Kazukuni Kobara and Hideki Imai. Countermeasure against reaction attacks (in japanese). In *The 2000 Symposium on Cryptography and Information Security : A12*, January 2000.

A Proof of Proposition 1

Let's consider a Bernouili trial of probability $1-p$, and let N be the random variable for the index of the first success. The following generating function gives the distribution of N

$$S_p(X) = \sum_{\ell=1}^{\infty} \Pr\left[N=\ell\right] X^\ell = \sum_{\ell=1}^{\infty} p^{\ell-1}(1-p)X^\ell = \frac{(1-p)X}{1-pX}.$$

The i-th loop of Algorithm 1 is a Bernoulli trial of probability $1-p_i$ with $p_i = i/n$. The successive iterations are independent, thus the distribution of number of calls to **rand**() is given by the series

$$S'(X) = \prod_{i=0}^{t-1} S_{p_i}(X) = \prod_{i=0}^{t-1} \frac{(n-i)X}{n-iX}.$$

The number of calls to **randbits**() in one call of **rand**() is itself a Bernoulli trial of probability $1-\pi$ and thus the generating series for the total number of calls to **randbits**() is

$$S(X) = S'(S_\pi(X)) = \prod_{i=0}^{t-1} \frac{(n-i)(1-\pi)X}{n-((n-i)\pi+i)X}.$$

B More on the Security Reduction

We consider an instance of BIKE's KEM where the Constant Weight Word (CWW) sampler $\mathbf{H}()$ produces elements of $\mathcal{E}_t$ with a non uniform distribution $\mathcal{D}$. We denote

$$\tau_{\max} = \max_{e\in\mathcal{E}_t} \frac{\Pr\left[e \mid e \xleftarrow{\mathcal{D}} \mathcal{E}_t\right]}{\Pr\left[e \mid e \xleftarrow{\$} \mathcal{E}_t\right]}.$$

Modifying the distribution of the CWW sampling from uniform to $\mathcal{D}$ close to uniform will not affect significantly the computational hardness of decoding nor the decoding failure rate. However the IND-CCA security model is more involved, in particular by allowing multiple call to the CWW sampler. We revisit below the IND-CCA proof of BIKE to check that the impact of this biased distribution on the IND-CCA security is indeed negligible.

In order to achieve IND-CCA security, the key encapsulation mechanism BIKE follows the $\mathtt{KEM}^{\not\perp}$ transformation of [HHK17] which derives itself from the Fujisaki-Okamoto transformation [FO99]. The initial public encryption $\mathtt{PKE}_0$ is transformed into $\mathtt{PKE}$ then $\mathtt{PKE}_1$ and finally into the KEM of Table 1. The final step of the proof in the HHK framework relates the security of the KEM to the security of $\mathtt{PKE}_1$ with [HHK17, Theorem 3.4] which states that for all IND-CCA adversary B against the KEM, there exists an OW-PCA adversary B' against $\mathtt{PKE}_1$ running in about the same time such that

$$\mathsf{Adv}_{\mathtt{KEM}^{\not\perp}}^{\text{IND-CCA}}(B) \le \frac{q_K}{|\mathcal{M}|} + \mathsf{Adv}_{\mathtt{PKE}_1}^{\text{OW-PCA}}(B')$$

where q_K is the number of calls to the key derivation function (viewed as an oracle). This inequality holds regardless of the CWW sampler $\mathbf{H}()$. No change here. The security of $\mathtt{PKE}_1$ relates to the security of $\mathtt{PKE}$ with [HHK17, Theorem 3.2] which states that for all IND-PCA

$\mathtt{PKE}_0$:

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$ $(h_0, h_1) \overset{\$}{\leftarrow} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
$\mathbf{Encrypt}_0$	Input: $h \in \mathcal{R}$, $(e_0, e_1) \in \mathcal{E}_t$ Output: $s \in \mathcal{R}$ $s \leftarrow e_0 + e_1 h$
$\mathbf{Decrypt}_0$	Input: $(h_0, h_1) \in \mathcal{H}_w$, $s \in \mathcal{R}$ Output: $e \in \mathcal{E}_t \cup \{\bot\}$ $e \leftarrow \text{decoder}(s h_0, h_0, h_1)$

$\mathtt{PKE}$:

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$ $(h_0, h_1) \overset{\$}{\leftarrow} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
Encrypt	Input: $h \in \mathcal{R}$, $m \in \mathcal{M}$ Output: $c \in \mathcal{R} \times \mathcal{M}$ $(e_0, e_1) \overset{\$}{\leftarrow} \mathcal{E}_t$; $c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
Decrypt	Input: $(h_0, h_1) \in \mathcal{H}_w$, (c_0, c_1) Output: $m \in \mathcal{M} \cup \{\bot\}$ $e \leftarrow \text{decoder}(s h_0, h_0, h_1)$ **if** $e = \bot$ **then** $m \leftarrow \bot$ **else** $m \leftarrow c_1 \oplus \mathbf{L}(e)$

$\mathtt{PKE}_1$:

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$ $(h_0, h_1) \overset{\$}{\leftarrow} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
$\mathbf{Encrypt}_1$	Input: $h \in \mathcal{R}$, $m \in \mathcal{M}$ Output: $c \in \mathcal{R} \times \mathcal{M}$ $(e_0, e_1) \leftarrow \mathbf{H}(m)$; $c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
$\mathbf{Decrypt}_1$	Input: $(h_0, h_1) \in \mathcal{H}_w$, $c \in \mathcal{R} \times \mathcal{M}$, $h \in \mathcal{R}$ Output: $m \in \mathcal{M} \cup \{\bot\}$ $m \leftarrow \mathbf{Decrypt}((h_0, h_1), c)$ **if** $m \neq \bot$ **and** $c \neq \mathbf{Encrypt}_1(h, m)$ **then** $m \leftarrow \bot$

Table 5: BIKE Encryption: $\mathtt{PKE}$ is randomized from $\mathtt{PKE}_0$ and $\mathtt{PKE}_1$ is derandomized from $\mathtt{PKE}$

adversary B' against $\mathtt{PKE}_1$, there exists an IND-CPA adversary A against $\mathtt{PKE}$ running in about the same time such that

$$\mathsf{Adv}^{\text{OW-PCA}}_{\mathtt{PKE}_1}(B') \leq q_H \cdot \delta + \frac{2 \cdot q_H + 1}{|\mathcal{M}|} + 3 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathtt{PKE}}(A) \tag{3}$$

where q_H is the number of calls to the CWW sampler (viewed as an oracle) made by the adversary. This part of the proof must be reengineered.

1. The first term $q_H \cdot \delta$ of the right-hand side of (3) relates to the correctness δ. It is an upper bound for the probability that at least one of the error pattern sampled by $\mathbf{H}()$ will produce a decoding failure. The value of δ must be such that
$$\Pr[(e_0, e_1) \neq \mathtt{decoder}(e_0 h_0 + e_1 h_1, h_0, h_1) \mid (h_0, h_1) \overset{\$}{\leftarrow} \mathcal{H}_w, (e_0, e_1) \overset{\mathcal{D}}{\leftarrow} \mathcal{E}_t] \leq \delta,$$
that is the DFR (Decoding Failure Rate) when e is sampled according to $\mathcal{D}$ (rather than uniformly) in $\mathcal{E}_t$. It follows from (2) that, compared to the proof for the original scheme, the DFR, that is the left-hand side of the above inequality, increases by a factor at most $\tau_{\max}$.
2. The second term appears in the proof of [HHK17, Theorem 3.2] in relation with the probability of the event that $\mathbf{H}()$ is called with a specific input. The ouput distribution of $\mathbf{H}()$ is irrelevant to bound this term. This part is unchanged.

$\mathtt{PKE}^{\mathcal{D}}$:

KeyGen$_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$ $(h_0, h_1) \overset{\$}{\leftarrow} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
Encrypt	Input: $h \in \mathcal{R}$, $m \in \mathcal{M}$ Output: $c \in \mathcal{R} \times \mathcal{M}$ $(e_0, e_1) \overset{\mathcal{D}}{\leftarrow} \mathcal{E}_t$; $c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
Decrypt	Input: $(h_0, h_1) \in \mathcal{H}_w$, (c_0, c_1) Output: $m \in \mathcal{M} \cup \{\perp\}$ $e \leftarrow \text{decoder}(sh_0, h_0, h_1)$ **if** $e = \perp$ **then** $m \leftarrow \perp$ **else** $m \leftarrow c_1 \oplus \mathbf{L}(e)$

Table 6: Modified PKE with Non Uniform Error Pattern

3. The final term must be modified because the derandomization of PKE would not lead to $\mathtt{PKE}_1$. Instead, we have to consider $\mathtt{PKE}^{\mathcal{D}}$, modified as in Table 6 with the error e sampled according to $\mathcal{D}$. The advantage in the rightmost term of (3) becomes $\mathsf{Adv}^{\text{IND-CPA}}_{\mathtt{PKE}^{\mathcal{D}}}(A)$ which can be viewed as a real-valued random variable over $\mathcal{E}_t$ equipped with the distribution $\mathcal{D}$. Again from (2), this modified advantage cannot exceed the original one by a factor more than $\tau_{\max}$.

So finally, for any adversary against BIKE's KEM, its advantage in the IND-CCA game cannot increase by more than a factor $\tau_{\max}$ when we replace the uniform CWW sampler by a biased one. To state things crudely, because $\tau_{\max}$ is a constant close to 1, any adversary able to break the scheme with a biased CWW sampler would be able to break the original scheme with about the same computational effort.