Secure Sampling of Constant Weight Words – Application to BIKE

Nicolas Sendrier

Inria
nicolas.sendrier@inria.fr

Abstract. The pseudorandom sampling of constant weight words, as it is currently implemented in cryptographic schemes like BIKE or HQC, is prone to the leakage of information on the seed being used for the pseudorandom number generation. 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 instances of such an attack were presented in an earlier version of this work [Sen21] and independently in [GHJ+22].
The timing attack stems from the variation of the amount of pseudorandom data to draw and process for sampling uniformly a constant weight word. We give here the exact distribution of this amount for BIKE. This will allow us to estimate precisely the cost of the natural countermeasure which consists in drawing always the same (large enough) amount of randomness for the sampler to terminate with probability overwhelmingly close to one.
The main contribution of this work is to suggest a new approach for fixing the issue. 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 the 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.

Keywords: Constant weight words, BIKE, constant-time implementation

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 constant weight words (CWW) depends on the seed used to initialized the pseudorandom 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 as shown independently in [Sen21] then in [GHJ+22].

Prior Art. The process used in BIKE for the pseudorandom sampling of CWW features a timing variation similar to the one observed in [HLS21] for HQC. If key reuse is allowed, it is possible to recover either a message or the secret key. Such attacks are suggested in [Sen21,GHJ+22]. In [GHJ+22], the attack is fully implemented and is successful against

known implementations of BIKE. The standard countermeasure consists in modifying the sampling to make sure it runs in constant time. In particular the pseudorandom data to generate must be fixed to an amount large enough to fulfill the sampler's need with overwhelming probability. This will result in a significant overhead for the CWW sampler of BIKE.

Contributions.

- We first give the exact distribution of the amount of pseudorandom data needed for sampling one CWW in BIKE. This will allow us to measure precisely the cost of the standard countermeasure.
- 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 believed to be a requirement for the IND-CCA security proof. 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 new variant of the Fisher-Yates shuffle which is particularly well suited to secure implementation. This new variant can be implemented with a predetermined sequence of operations and memory access, making it oblivious to its internal randomness. It samples constant weight words from uniformly distributed random bits with a distribution that is provably close enough to uniform to have a negligible impact on security.

Organization of the Paper. In §2 we recall how the constant weight word (CWW) sampler of BIKE works, how it can be attacked, and what would be the overhead of standard countermeasures. In §3 we present a variant for the Fisher-Yates shuffle, which is well suited to secure implementation of CWW sampling. Finally, in §4 we prove that BIKE's IND-CCA security does not decrease significantly if constant weight words are sampled non uniformly, but close enough to uniform. In addition, it is shown that the Fisher-Yates variant of §3 can be implemented securely as BIKE's CWW sampler with a distribution close enough to uniform.

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 $\{0, 1, \ldots, n-1\}$. 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 which sets the bits corresponding to the selected positions in a binary word of length n. The function $\mathbf{rand}(n, \mathsf{prng})$ produces a random integer uniformly distributed in $\{0, 1, \ldots, n-1\}$ from uniformly distributed integers in $\{0, 1, \ldots, 2^B - 1\}$ produced by $\mathbf{randbits}(B, \mathsf{prng})$. The pseudorandom number generator prng is initialized with seed. In BIKE, $\mathtt{mask} = 2^u - 1$ 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

Algorithm 1 Current BIKE Constant Weight Word Sampling

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

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

specification, it was first suggested to use AES in counter mode then SHAKE 256 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, not upper bounded, but ranging in practice 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 $\texttt{decoder}(e_0 h_0 + e_1 h_1, h_0, h_1) = (e_0, e_1)$ 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 be adapted to BIKE, as briefly mentioned in the conclusion of [HLS21]. The exact mechanism differs slightly, but the principle is the same. It was first sketched in a preliminary version of this work [Sen21], then, independently, described and implemented in [GHJ+22]. Below we describe it as in [Sen21].

The attacks presented here do not apply to the ephemeral key setting but only to scenarii where secret keys are 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.

Notation	
$\mathbb{F}_2$:	Binary finite field
$\mathcal{R}$:	Cyclic polynomial ring $\mathbb{F}_2[X]/(X^r-1)$, equivalent to $\mathbb{F}_2^r$
$\mathcal{H}_w$:	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 $g\in\mathcal{R}$
$u\xleftarrow{\$}U$:	Variable u is sampled uniformly at random from the set U
$u\xleftarrow{\mathcal{D}}U$:	Variable u is sampled from the set U according to distribution $\mathcal{D}$

KeyGen : $() \mapsto (h_0,h_1,\sigma),h$

Output: $(h_0,h_1,\sigma)\in\mathcal{H}_w\times\mathcal{M}$, $h\in\mathcal{R}$

1: $(h_0,h_1)\xleftarrow{\$}\mathcal{H}_w$
2: $h\leftarrow h_1h_0^{-1}$
3: $\sigma\xleftarrow{\$}\mathcal{M}$

Encaps : $h\mapsto K,c$

Input: $h\in\mathcal{R}$

Output: $K\in\mathcal{K}$, $c\in\mathcal{R}\times\mathcal{M}$

1: $m\xleftarrow{\$}\mathcal{M}$
2: $(e_0,e_1)\leftarrow\mathbf{H}(m)$
3: $c\leftarrow(e_0+e_1h, m\oplus\mathbf{L}(e_0,e_1))$
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: $e'\leftarrow\texttt{decoder}(c_0h_0,h_0,h_1)$ ▷ $e'\in\mathcal{R}^2\cup\{\perp\}$
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

In BIKE, a call to **Decaps** on a ciphertext (c_0,c_1) 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, with Algorithm 1, 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 choosing (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 mentioned above. To thwart this attack, it is required that $\mathbf{H}()$ runs in time independent of its input, in particular it must always generate the same amount of randomness.

Using a Constant Amount of Randomness. Since the running time variation comes in a large part 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**() with a modulo, as in HQC (Table 2). It has a smaller rejection rate, in the order of 10^{-6} if $B = 32$ and 10^{-3} if $B = 24$, and leads to a smaller overhead.

rand(n, prng) :
1: **repeat**
2: $\quad x \leftarrow$ **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* **randbits**() *calls 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.

Overhead for Constant-Time Implementation. The generating function of Proposition 1 provides an efficient and convenient way[1] 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 with $B = 32$), 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). An execution of Algorithm 1 with no rejection at all would require t calls to **randbits**(), thus the overhead is $N_0 - t$ and the relative overhead is $(N_0 - t)/t$. The

			rand() with `mask`			**rand**() with mod		
n	t	λ	π	$N_0 - t$	$\frac{N_0-t}{t}$	π	$N_0 - t$	$\frac{N_0-t}{t}$
24 646	134	128	0.248	193	144%	$1.7\,10^{-6}$	26	19%
49 318	199	192	0.247	289	145%	$5.1\,10^{-6}$	37	19%
81 194	264	256	0.381	600	227%	$1.1\,10^{-5}$	48	18%

Table 3: Overheads for a Constant-Time Implementation of Algorithm 1 ($B = 32$)

countermeasures proposed in [HLS21] and in [GHJ+22] are essentially of the same nature and lead to similar overheads.

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 still depend on the instance, making it vulnerable to cache attacks and requiring a careful implementation, possibly involving another overhead, *e.g.* [GHJ+22].

Cost for Producing a Distinguishable Message. Proposition 1 can also help to determine what could be the cost for producing 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 2), 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 is enough to make the timing difference measurable, and thus the attack effective, will depend on the adversary's capabilities, 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 IND-CCA security reduction given in the BIKE specification document, it is required that constant weight words are produced with a uniform distribution. However, we show later in this paper, in §4.2, that if instead the distribution is only close to uniform, the impact on security is negligible. Meanwhile, we show in §3 that a variant

[1] Using a suitable computer algebra system, *e.g.* Maple

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 (left-hand side), it returns a sequence of t distinct elements of $\{0, \ldots, n-1\}$ 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.

Algorithm 2 Fisher-Yates Algorithm

Input: n, t	***Abstract version***
Output: t distinct elements of $\{0, \ldots, n-1\}$	▷ $(i\ \ell)$ *the transposition of* i *and* ℓ
1: pos $\leftarrow [0, 1, \ldots, n-1]$	*1:* $\sigma \leftarrow \mathrm{id}$ ▷ *identity permutation*
2: **for** $i = 0$ **to** $t-1$ **do**	*2:* ***for*** $i = 0$ **to** $t-1$ ***do***
3: $\quad \ell \xleftarrow{\$} \{i, \ldots, n-1\}$	*3:* $\quad \ell \xleftarrow{\$} \{i, \ldots, n-1\}$
4: $\quad$ swap pos$[i]$ and pos$[\ell]$	*4:* $\quad \sigma \leftarrow \sigma \circ (i\ \ell)$
5: **return** pos$[0], \ldots,$ pos$[t-1]$	*5:* ***return*** $\sigma(0), \ldots, \sigma(t-1)$

Mathematical Abstraction. The right-hand side of Algorithm 2 is a mathematical abstraction of the algorithm, step by step equivalent, where the representation of the permutation is not specified. It is easy to check that at any point of the algorithm pos $= [\sigma(0), \ldots, \sigma(n-1)]$. 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 $\mathrm{pos}[i] = \sigma(i)$, $0 \le i < t$, there is no need for a table of size n nor for data-dependent access to memory. Note that $\mu_j(i) = i$ if $j > i$, so to compute $\sigma(i)$ we may ignore the transpositions μ_j for $j > i$. At the end of execution, we have for all i, $0 \le i < t$,

$$\mathrm{pos}[i] = \sigma(i) = \mu_0 \circ \cdots \circ \mu_{i-1} \circ \mu_i(i) = \mu_0 \circ \cdots \circ \mu_{i-1}(s_i).$$

Which leads to the variant proposed in the next section.

3.2 A Variant of Fisher-Yates Shuffle

The variant described in Algorithm 3 will iterate on the randomly sampled transpositions $(\mu_i)_{0 \le i < t}$ to update the images $\mathrm{pos}[j], 0 \le j < t$ of $0, \ldots, t-1$. Note that the

Algorithm 3 Fisher-Yates Algorithm – Variant

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

transpositions need to be applied in reverse order, μ_{t-1} first and μ_0 last. At the i-th iteration, Algorithm 3 applies the transposition μ_i to update the relevant entries of table pos[]. At the end of the i-th iteration we have $\mathrm{pos}[j] = \sigma_i(j)$ for all j, $i \le j < t$, where $\sigma_i = \mu_i \circ \cdots \circ \mu_{t-1}$.

Secure Implementation. A key feature of Algorithm 3 is that the pattern of access to the table pos[] is independent of the data, which allows implementations that easily reach immunity 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.

4 Alternative Implementation for BIKE

The above variants of Fisher-Yates, Algorithms 2 and 3, return the same object if they use the same randomness (the randomness is in reverse order in Algorithm 3, because the loop order is reversed). They generate a random permutation σ as a product of t transpositions, and they return the sequence of integers $\sigma(0), \sigma(1), \ldots, \sigma(t-1)$ in this order. When $t = n$, the output is a permutation. To sample uniformly a (binary) constant weight word, one uses the t returned values as the indices of the non-zero positions of a

word of length n. Those indices could be returned in any order and still produce the same word. This allows yet another variant given in Algorithm 4, which enjoys a marginally simpler memory writing pattern.

Algorithm 4 Constant Weight Word Sampling

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

Proposition 2. *If they use the same randomness (instruction 2), Algorithm 3 and Algorithm 4 return the same list of integers up to the order.*

Proof. We assume that in both algorithms the initial values drawn for $\text{pos}[i]$ are identical, we will denote s_i this integer, note that $s_i \geq i$. Let us prove by induction that, in both algorithms, at the end of every main loop (instructions 1-4) the *sets* $\{\text{pos}[j], i \leq j < t\}$ are identical. It is true for $i = t-1$, the set is the singleton $\{s_{t-1}\}$. At the i-th iteration (remember i goes backwards), the element s_i comes into play. The following holds for both algorithms:

- if $s_i \notin \{\text{pos}[j], i < j < t\}$ then $\{\text{pos}[j], i \leq j < t\} = \{\text{pos}[j], i < j < t\} \cup \{s_i\}$,
- if $s_i \in \{\text{pos}[j], i < j < t\}$ then $\{\text{pos}[j], i \leq j < t\} = \{\text{pos}[j], i < j < t\} \cup \{i\}$.

At the end of the i-th iteration, the sets $\{\text{pos}[j], i \leq j < t\}$ are thus identical in both algorithms. When the induction reaches $i = 0$, we get the statement. □

In fact, Algorithm 4 does not return all sequences of t distinct positions uniformly. For instance, sequences with $\text{pos}[t-1] < t-1$ are never obtained. In particular Algorithm 4 cannot be used with $t = n$ to sample permutations uniformly. However, it produces subsets of $\{0, \ldots, n-1\}$ of cardinality t, and thus binary words of length n and Hamming weight t, uniformly.

4.1 Relaxing the Distribution

Finally, we suggest to replace Algorithm 1, the CWW sampler of BIKE, by Algorithm 5. In this algorithm, which derives from Algorithm 4, the random integers at the beginning of each loop are sampled with a bias. However, it can easily be implemented with a protection against timing and cache attacks. An additional step is required to produce the words in their final form, *e.g.* transform a list of t indices into a binary word of Hamming weight t. This step also needs to be protected, as for instance in the available state-of-the-art implementations [DGK] of BIKE.

Algorithm 5 Constant Weight Word Sampling – Alternative for BIKE

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

rand(n, prng) :
1: $x \leftarrow$ **randbits**(B, prng)
2: **return** $x \mod n$

Proposition 3. *Let $\mathcal{D}$ be the distribution over $\mathcal{E}_t = \{e \in \mathbb{F}_2^n \mid |e| = t\}$ stemming from Algorithm 5, when $x \leftarrow$ **randbits**(B, prng) behaves as a random oracle which yields uniformly distributed integers, $0 \le x < 2^B$. For any integer $B > 0$, we have*

$$\prod_{i=0}^{t-1}\left(1 - \frac{r_i}{2^B}\right) = \tau_{\min} \le \frac{\Pr\left[e \mid e \xleftarrow{\mathcal{D}} \mathcal{E}_t\right]}{\Pr\left[e \mid e \xleftarrow{\$} \mathcal{E}_t\right]} \le \tau_{\max} = \prod_{i=0}^{t-1}\left(1 + \frac{(n-i) - r_i}{2^B}\right), \tag{1}$$

where $r_i = 2^B \mod (n-i)$ for all i, $0 \le i < t$.

Proof. Let s_i denote the random integer, $i \le s_i < n$, obtained at the i-th iteration of Algorithms 2, 3, 4, and 5. There are $\prod_{i=0}^{t-1}(n-i)$ different sequences $(s_i)_{0 \le i < t}$ which constitute the randomness of those algorithms. Each randomness sequence produces each possible output sequence of Algorithms 2 and 3 exactly once. Thus, each subset of t elements of $\{0, \dots, n-1\}$ is obtained exactly $t!$ times in Algorithms 2 and 3, as well as in Algorithm 4 because of Proposition 2. With the same randomness Algorithm 5 is identical to Algorithm 4, thus each individual element $e \in \mathcal{E}_t$ is obtained as an output of Algorithm 5 for exactly $t!$ distinct randomness sequences $s = (s_i)_{0 \le i < t}$. We denote $\mathcal{S}(e)$ the set of those $t!$ sequences. We have

$$\Pr\left[e \mid e \xleftarrow{\mathcal{D}} \mathcal{E}_t\right] = \sum_{s \in \mathcal{S}(e)} \Pr\left[s \mid s_i \leftarrow i + \mathbf{rand}(n-i, \mathsf{prng}), 0 \le i < t\right] \tag{2}$$

$$= \sum_{s \in \mathcal{S}(e)} \prod_{i=0}^{t-1} \Pr\left[s_i \mid s_i \leftarrow i + \mathbf{rand}(n-i, \mathsf{prng})\right] \tag{3}$$

Where (3) derives from (2) because prng is modelled as a random oracle and so the s_i are independent from one another. Now, for all i, $0 \le i < t$, we denote $r_i = 2^B \mod (n-i)$. The call $\mathbf{rand}(n-i, \mathsf{prng})$ will return biased integers, as the values $< r_i$ are slightly more likely than those $\ge r_i$. We have

$$\Pr\left[s_i \mid s_i \leftarrow i + \mathbf{rand}(n-i, \mathsf{prng})\right] = \begin{cases} \frac{1}{n-i}\left(1 + \frac{(n-i)-r_i}{2^B}\right) & \text{if } i \le s_i < i + r_i \\ \frac{1}{n-i}\left(1 - \frac{r_i}{2^B}\right) & \text{if } i + r_i \le s_i < n \end{cases} \tag{4}$$

and thus from (4) and (3), and because $|\mathcal{S}(e)| = t!$, for all $e \in \mathcal{E}_t$ we have

$$t! \cdot \prod_{i=0}^{t-1} \frac{1}{n-i}\left(1 - \frac{r_i}{2^B}\right) \le \Pr\left[e \mid e \xleftarrow{\mathcal{D}} \mathcal{E}_t\right] \le t! \cdot \prod_{i=0}^{t-1} \frac{1}{n-i}\left(1 + \frac{(n-i) - r_i}{2^B}\right).$$

Finally, for the uniform distribution over $\mathcal{E}_t$, we have

$$\Pr\left[e \mid e \xleftarrow{\$} \mathcal{E}_t\right] = \frac{1}{\binom{n}{t}} = t! \cdot \prod_{i=0}^{t-1} \frac{1}{n-i}$$

and we easily conclude the proof. □

For BIKE parameters, the ratios $\tau_{\min}$ and $\tau_{\max}$ are very close to 1 (see Table 4). As

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

$B = 24$			
n	t	$\tau_{\min}$	$\tau_{\max}$
24 646	134	0.91	1.11
49 319	199	0.76	1.37
81 194	264	0.51	1.85

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

shown next, this is close enough to the uniform distribution to have a negligible impact on security.

4.2 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 error patterns 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

$$\sum_{e \in \mathcal{E}_t} \Pr\left[e \mid e \xleftarrow{\mathcal{D}} \mathcal{E}_t\right] V(e) \leq \tau_{\max} \cdot \sum_{e \in \mathcal{E}_t} \Pr\left[e \mid e \xleftarrow{\$} \mathcal{E}_t\right] V(e). \tag{5}$$

It follows that the two terms mentioned above, and thus the advantage of any adversary when the distribution of the error $e \in \mathcal{E}_t$ changes from uniform to $\mathcal{D}$, cannot increase by a factor larger than $\tau_{\max}$, as defined in (1). And, from Table 4, this factor $\tau_{\max}$ is small enough to ignore its impact on security.

Note that using a biased internal distribution in a cryptosystem can have a stronger impact on security than what we observe here. The security proof must be checked, as we do in §B, and if the adversarial model allows multiple access to the biased distribution, more stringent arguments and more accurate metrics may be required. This is the case for instance in lattice-based crypto with the use of Rényi divergence [BLL+15,Pre17].

5 Conclusion

There exists a vulnerability stemming from variable time sampling of constant weight words (CWW) in HQC [HLS21] and in BIKE [Sen21,GHJ+22]. We analyze precisely this timing variation and deduce the minimal implementation overhead for the CWW sampler of BIKE.

We have proposed another approach to avoid the timing attacks targeting BIKE's CWW sampling. It is based on the Fisher-Yates shuffle and is original in two respects. First, and contrary to common belief about Fisher-Yates shuffle, our variant is very well suited for implementations secure 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 provably have no effective impact on the global security of the BIKE scheme.

Our new proposed algorithm for constant weight word sampling does not have a significantly higher algorithmic complexity and, though its output is not uniform, it does not reduce the IND-CCA security of BIKE. We believe that for secure implementations, it offers an advantageous trade-off.

Finally, note that it is possible to sample a word of length n and constant weight word t by sampling a permutation, and permutations can be sampled in constant time by sorting an array of n random numbers, see [WSN18] for instance. This is not advantageous when the weight $t = O(\sqrt{n})$ as the cost of sorting will always exceed the cost t^2 of the sampler proposed here.

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`.

BLL+15. Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015*, volume 9452 of *LNCS*, pages 3–24. Springer, 2015.

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, pages 537–554. Springer, 1999.

GHJ+22. Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. Cryptology ePrint Archive, Report 2021/1485, version 3, January 2022. `https://ia.cr/2021/1485`.

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, version 2, November 2021. `https://eprint.iacr.org/2021/1485/20211121:100918`.

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

Pre17. Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017*, volume 10624 of *LNCS*, pages 347–374. Springer, 2017.

Sen21. Nicolas Sendrier. Secure sampling of constant-weight words – application to bike. Cryptology ePrint Archive, Report 2021/1631, version 1, December 2021. `https://eprint.iacr.org/2021/1631/20211217:142141`.

WSN18. Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based Niederreiter cryptosystem using binary Goppa codes. In Tanja Lange and Rainer Steinwandt, editors, *PQCrypto*, volume 10786 of *LNCS*, pages 77–98. Springer, 2018.

A Proof of Proposition 1

Let's consider a Bernoulli 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

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

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, BIKE follows the $\texttt{KEM}^{\not\perp}$ transformation of [HHK17] which derives itself from the Fujisaki-Okamoto transformation [FO99]. The

$\texttt{PKE}_0$:

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$
	$(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
$\mathbf{Encrypt}_0$	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: $e \in \mathcal{E}_t \cup \{\perp\}$
	$e \leftarrow \mathrm{decoder}(sh_0, h_0, h_1)$

$\texttt{PKE}$:

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

$\texttt{PKE}_1$:

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$
	$(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
$\mathbf{Encrypt}_1$	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: $m \in \mathcal{M} \cup \{\perp\}$
	$m \leftarrow \mathbf{Decrypt}((h_0, h_1), c)$
	if $m \neq \perp$ **and** $c \neq \mathbf{Encrypt}_1(h, m)$ **then** $m \leftarrow \perp$

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

initial public encryption $\texttt{PKE}_0$ is transformed into $\texttt{PKE}$ then $\texttt{PKE}_1$ 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 $\texttt{PKE}_1$ 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 $\texttt{PKE}_1$ running in about the same time such that

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

where q_K is the number of calls to the key derivation function (viewed as a random 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 adversary B' against $\mathtt{PKE}_1$, there exists an IND-CPA adversary A against $\mathtt{PKE}$ running in about the same time such that

$$\mathsf{Adv}^{\text{OW-PCA}}_{\mathtt{PKE}_1}(B') \leq q_H \cdot \delta + \frac{2 \cdot q_H + 1}{|\mathcal{M}|} + 3 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathtt{PKE}}(A) \tag{6}$$

where q_H is the number of calls to the CWW sampler (viewed as a random oracle) made by the adversary. This part of the proof must be re-engineered.

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

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

Table 6: Modified $\mathtt{PKE}$ with Non Uniform Error Pattern

1. The first term $q_H \cdot \delta$ of the right-hand side of (6) 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

$$\Pr[(e_0, e_1) \neq \mathtt{decoder}(e_0 h_0 + e_1 h_1, h_0, h_1) \,|\, (h_0, h_1) \xleftarrow{\$} \mathcal{H}_w, (e_0, e_1) \xleftarrow{\mathcal{D}} \mathcal{E}_t] \leq \delta,$$

   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 (5) 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 output distribution of $\mathbf{H}()$ is irrelevant to bound this term. This part is unchanged.
3. The final term must be modified because the derandomization of $\mathtt{PKE}$ 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 (6) 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 (5), 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.