HILA5 Pindakaas:[†] On the CCA Security of Lattice-Based Encryption with Error Correction

Daniel J. Bernstein[1](✉), Leon Groot Bruinderink[2](✉), Tanja Lange[2](✉), and Lorenz Panny[2](✉)

[1] Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7045, USA
`djb@cr.yp.to`

[2] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
`l.groot.bruinderink@tue.nl`, `tanja@hyperelliptic.org`, `lorenz@yx7.cc`

Abstract. We show that the NISTPQC submission HILA5 is not secure against chosen-ciphertext attacks. Specifically, we demonstrate a key-recovery attack on HILA5 using an active attack on reused keys. The attack works around the error correction in HILA5. The attack applies to the HILA5 key-encapsulation mechanism (KEM), and also to the public-key encryption mechanism (PKE) obtained by NIST's procedure for combining the KEM with authenticated encryption. This contradicts the most natural interpretation of the IND-CCA security claim for HILA5.

Keywords: Post-quantum cryptography · KEM · RLWE
Reaction attack

1 Introduction

HILA5 [13] is a public-key scheme designed by Saarinen and published at SAC 2017. HILA5 was submitted as a "Key Encapsulation Mechanism and Public Key Encryption Algorithm" [12] to NIST's call [10] for post-quantum proposals. HILA5's design is based on Ring Learning With Errors (RLWE) over NTRU NTT rings. HILA5 takes the same ring parameters as New Hope [2] and changes the reconciliation method by which Alice and Bob achieve the same key to get a much lower chance of decryption failures.

The HILA5 submission [12] states

[†] "Helaas pindakaas" is a Dutch expression meaning "Oh well, too bad".

[*] Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported in part by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO), project number 643161 (ECRYPT-NET), and project number 645421 (ECRYPT-CSA); and by the U.S. National Science Foundation under grant 1314919. Date of this document: 2018.02.27.

This design also provides IND-CCA secure KEM-DEM [CS03] public key encryption if used in conjunction with an appropriate AEAD [Rog02] such as NIST approved AES256-GCM [FIP01, Dwo07].

In this paper we show that HILA5 is not CCA secure: We compute Alice's secret key by sending her multiple encapsulation messages and using her answers to determine whether her decapsulated shared secret matches a certain guess or not. Our attack works independently of whether an AEAD is used or not and despite the error correcting code introduced in HILA5.

We have fully implemented our attack and experimentally verified that it works with high probability. We use the HILA5 reference implementation for Alice's part and also to verify that the retrieved secret key works for decryption. We use a slightly modified version of the same software for computations on the attacker's side; of course the attacker need not follow the computations an honest party would.

Acknowledgement. We thank Christine van Vredendaal for helpful discussions.

1.1 Related Work

Ajtai–Dwork [1] and NTRU [7] are the oldest lattice-based encryption systems. In 1999 Hall, Goldberg, and Schneier [6] developed a reaction attack which recovers the Ajtai–Dwork private key by observing decryption failures for suitably crafted encryptions to the public key. They wrote "We feel that the existance of these attacks effectively limits these ciphers to theoretical considerations only. That is, any implementation of the ciphers will be subject to the attacks we present and hence not safe."

Hoffstein and Silverman [8] adapted the attack to NTRU. As a defense, they suggested modifying NTRU to use the Fujisaki–Okamoto transform [5]. For a system without decryption failures, this transform turns a CPA-secure system into a CCA-secure one. At the same time this complicates and slows down the cryptosystem. For NTRU, the transform turns out to still allow attacks that exploit occasional decryption failures induced by *valid* ciphertexts; see [9].

New Hope [2] is a key-encapsulation mechanism (KEM), presented as a key-exchange protocol. It allows occasional decryption failures for valid ciphertexts, and explicitly avoids the "changes" that would be required for the Fujisaki–Okamoto transform. To prevent reaction attacks and other chosen-ciphertext attacks by a malicious Bob, New Hope requires using ephemeral keys, meaning keys that change with every execution of the protocol. The New Hope paper warns that reusing a public key in multiple protocol runs ("key caching") would be "disastrous for security", although it does not describe an attack.

Fluhrer [4] showed the details of how to attack key reuse in a similar key-exchange protocol. Followup work [3] extended the attack to more key-exchange protocols.

HILA5 is similar to New Hope, and still does not use the Fujisaki–Okamoto transform. HILA5 includes an error-correction method that practically eliminates decryption failures for valid ciphertexts. HILA5 does not warn against key caching: on the contrary, the most natural interpretation of the HILA5 security claims is that HILA5 is secure against chosen-ciphertext attacks. See Sect. 5. We published our results in December 2017; as of February 2018, the designer of HILA5 has not proposed an alternative interpretation of the security claims.

2 Data Flow in the Attack

A KEM is defined by three algorithms. Key generation produces a secret key and a public key. Encapsulation produces a ciphertext and a session key, given a public key. Decapsulation produces a session key or failure, given a ciphertext and a secret key. The HILA5 submission document [12] gives details and reference code for a particular KEM, the "HILA5 KEM".

Our attack is a key-recovery attack against the HILA5 KEM: the attacker, evil Bob, ends up computing the secret key of a target Alice. This secret key gives the attacker the ability to run the decapsulation algorithm using Alice's secret key, and thus the ability to immediately decrypt legitimate ciphertexts sent by other users to Alice.

Our attack is a chosen-ciphertext attack: evil Bob chooses ciphertexts to provide to Alice (different from the legitimate ciphertexts), and learns something from observing the outputs of Alice decapsulating those ciphertexts. Formally, the attack shows that the HILA5 KEM does not provide IND-CCA2 security.

There are two important ways that the attack does not need the full power of a CCA2 decapsulation oracle. First, the attack is what is called a "reaction attack" in [6] or a "sloppy Alice attack" in [14]: evil Bob has a guess for the output of each decapsulation, and learns whether Alice's actual decapsulation output matches this guess. Evil Bob does not need any further information.

Second, evil Bob chooses all of his ciphertexts, and learns the secret key from Alice's reactions, before seeing the legitimate ciphertexts to decrypt. Formally, the attack shows not only that the HILA5 KEM does not provide IND-CCA2 security, but also that it does not provide IND-CCA1 security.

2.1 Hashing the Secret Key Does Not Stop the Attack

One can easily stop key-recovery attacks by defining HILA5Hash as follows. HILA5Hash key generation computes a uniform random 32-byte string s, and then runs HILA5 key generation to obtain a public key, hashing s to generate all randomness used in HILA5 key generation. The HILA5Hash secret key is s. HILA5Hash encapsulation is the same as HILA5 encapsulation. HILA5Hash decapsulation reconstructs the HILA5 secret key from s (again running the HILA5 key-generation algorithm; alternatively, the HILA5 secret key can be cached), and then runs the HILA5 decapsulation algorithm.

Unless the hash function is easy to invert, a key-recovery attack against HILA5 does not produce a key-recovery attack against HILA5Hash. However, this hashing does not prevent the attacker from decrypting legitimate ciphertexts sent by other users to Alice.

2.2 AEAD Does Not Stop the Attack

A PKE is defined by three algorithms. Key generation produces a secret key and a public key, as in a KEM. Encryption produces a ciphertext, given a plaintext and a public key. Decryption produces a plaintext or failure, given a ciphertext and a secret key.

The subtitle of the HILA5 submission is "Key Encapsulation Mechanism (KEM) and Public Key Encryption Algorithm". The submission document does not include a definition of a PKE, but NIST had already stated before submission that it would automatically convert each submitted KEM to a PKE using the following "standard conversion technique": "appending to the KEM ciphertext, an AES-GCM ciphertext of the plaintext message" where the AES-GCM key is "the symmetric key output by the encapsulate function". This is the standard Cramer–Shoup "KEM-DEM" construction, using AES-GCM as the DEM. We write "HILA5 PKE" for the PKE that NIST will automatically produce in this way from the HILA5 KEM.[1]

Breaking the IND-CCA2 security of a KEM does not necessarily imply breaking the IND-CCA2 security of a PKE obtained in this way. IND-CCA2 attacks against the KEM can see session keys produced by decapsulation, whereas IND-CCA2 attacks against the PKE are merely able to see the result of AES-GCM decryption using those keys.

However, our attack against the HILA5 KEM is also a key-recovery attack against the HILA5 PKE. It is important here that the attack is a reaction attack: what evil Bob needs to know is merely whether a guessed session key is correct. Starting from this guessed session key, evil Bob produces a valid AES-GCM ciphertext using this guess as an AES key. If decapsulation in fact produces this session key then AES-GCM decryption succeeds and produces the plaintext that evil Bob started with. If decapsulation produces a different session key then AES-GCM decryption is practically guaranteed to fail (anything else would be a surprising security flaw in AES-GCM), so evil Bob sees a decryption failure from the PKE.

To summarize, evil Bob sees decryption failures from the PKE, and learns from this which guesses were correct, which is the same information that evil Bob obtains from the KEM. Evil Bob then computes the secret key from this information. Consequently, the HILA5 PKE does not provide IND-CCA2 security, and does not even provide IND-CCA1 security.

2.3 Black Holes Would Stop the Attack

Like other chosen-ciphertext attacks, our attack is inapplicable to scenarios where the results of decapsulation and decryption are hidden from the attacker.

[1] NIST actually deviates slightly from the KEM-DEM construction: it specifies a "randomly generated IV" for AES-GCM, while Cramer and Shoup use a deterministic DEM. For consistency with the ciphertext sizes mentioned in [12], we actually define "HILA5 PKE" to be the Cramer–Shoup construction using AES-GCM with an all-zero IV. Switching to NIST's construction would expand ciphertext sizes by 12 bytes using the default IV sizes for AES-GCM, and would not affect our attack.

For example, if ciphertexts are sent to NSA's public key, and if NSA hides the results of applying its secret key to those ciphertexts, then an attacker outside NSA cannot use our attack to compute NSA's secret key. However, if NSA reacts to those results in a way that leaks to the attacker which ciphertexts were valid, then the attacker can compute NSA's secret key.

2.4 The Fujisaki–Okamoto Transform Would Stop the Attack

We briefly outline a more radical change to HILA5, which we call "HILA5FO". HILA5FO ciphertexts are slightly larger than HILA5 ciphertexts, decapsulation is more complicated, and decapsulation is extrapolated (from reported HILA5 benchmarks) to be several times slower, but HILA5FO would stop our attack.

The idea of the HILA5FO KEM is to reapply the encapsulation algorithm as part of decapsulation, and check whether the resulting ciphertext is identical to the received ciphertext. This is not a new idea: it is used in many other submissions to NIST (with various differences in details), typically with credit to Fujisaki and Okamoto [5].

HILA5 does not provide any easy way to reconstruct the randomness used in encapsulation (most importantly Bob's b), so the HILA5FO KEM computes this randomness as a hash of a plaintext recovered as part of decapsulation. The HILA5 KEM does not transmit a plaintext, so the HILA5FO KEM is instead built from the HILA5 PKE.

Encapsulation in the HILA5FO KEM thus chooses a random plaintext, and encrypts this plaintext using the HILA5 PKE (the HILA5 KEM producing a session key for AES-GCM) using a hash of the plaintext to compute all randomness used inside the PKE. Decapsulation applies HILA5 PKE decryption (HILA5 KEM decapsulation producing a session key for AES-GCM decryption), and checks that the resulting plaintext produces the same ciphertext.

Deriving a PKE from the HILA5FO KEM would involve two layers of AES-GCM, which can be compressed to one layer as follows: place 32 bytes of randomness at the beginning of the user-supplied plaintext, and then encrypt this plaintext using the HILA5 PKE, again using a hash of the plaintext to compute all randomness used inside the PKE. The overall ciphertext size is the original plaintext size, plus 32 bytes (the randomness), plus the HILA5 KEM ciphertext size, plus 16 bytes (the AES-GCM authenticator), i.e., 32 bytes more than the HILA5 PKE. The main cost in HILA5FO decryption (for short messages) is reapplying HILA5 KEM encapsulation, which according to [12, Table 1] is five times slower than HILA5 KEM decapsulation.

3 Preliminaries

This section describes the HILA5 scheme and Fluhrer's attack on RLWE schemes.

3.1 The HILA5 Scheme

We describe the scheme as given in [12, Sect. 4.9] but leave out formatting and NTT conversions. These are used in the attack implementation to interface with the reference implementation but do not contribute to the security and hamper readability.

The major computations take place in the ring $R = \mathbb{Z}_q[x]/(x^n + 1)$, where $n = 1024$ and $q = 12289$. Alice's secret key is a small, random polynomial $a \in R$, where small (here and in the following) means that the coefficients are chosen from a narrow distribution around zero, more precisely the discrete binomial distribution Ψ_{16} which has integer values in $[-16, 16]$. To compute the public key she picks another small random polynomial $e \in R$ and a random $g \in R$ and computes $A = ga + e$. She publishes (g, A) and keeps a as her secret.

An honest Bob picks two random small polynomials $b, e' \in R$ and computes $B = gb + e'$ and $y = Ab$. Bob sends B to Alice. The second value

$$y = Ab = (ga + e)b = gab + eb \approx gab$$

is very close to what Alice can compute using her secret:

$$x = aB = a(gb + e') = gab + e'a \approx gab,$$

because a, b, e, e' are all small.

A simple rounding operation to achieve a shared secret, such as taking the top bits of each coefficient, will induce differences between Alice's and Bob's version with too high probability. For example, Bob could take $k[i] = \lfloor 2\,y[i]/q \rceil$ and Alice could take $k'[i] = \lfloor 2\,x[i]/q \rceil$, where we use $t[i]$ to denote the ith coefficient of polynomial or vector t, but for indices with $(gab)[i] \approx 0$ (or $q/2$) the error-terms can cause the values to flip to a different bit, i.e., $k[i] \neq k'[i]$. For this rounding operation, we call elements of $\{0, q/2\}$ the "edges", as these are the values for which it is probable that errors occur.

This is why Bob sends a second vector, a binary reconciliation vector c, to help Alice recover the same k as Bob. Basically, this means that the scheme uses two pairs of edges. If $y[i]$ was close to one edge of a certain pair, Bob will choose the other pair of edges, so that Alice can still successfully recover the shared secret. In previous work [11], the reconciliation vector achieves a successful shared secret with high probability, as long as $|x[i] - y[i]| < q/8$.

HILA5 differs in how these reconciliation bits are computed. For each coefficient $y[i]$ of y Bob computes $k[i] = \lfloor 2\,y[i]/q \rfloor$, $c[i] \equiv \lfloor 4\,y[i]/q \rfloor \bmod 2$, and

$$d[i] = \begin{cases} 1 & \text{if } |(y[i] \bmod \lfloor q/4 \rceil) - \lfloor q/8 \rfloor| \leq \beta \\ 0 & \text{otherwise,} \end{cases}$$

where $\beta = 799$. He then selects the first 496 positions i for which $d[i] = 1$ and restarts with fresh b and e' if there are fewer. Positions with $d[i] = 1$ are those for which it is likely that Alice and Bob recover the same value. In other words, for these indices the value $(gab)[i]$ is likely to be far away from an edge, thus further reducing the probability of errors in the shared secret. (Note that the description suggests to discard some positions if there are more than 496 such positions while the code deterministically discards the later ones by setting $d[j] = 0$ for them.)

The encapsulation consists of B, d, c, and an extra part r described below; here d covers the full n positions while c can be compressed to those positions i where $d[i] = 1$.

Alice recovers the $k[i]$ at the selected 496 positions by computing

$$k'[i] = \lfloor 2\,(x[i] - c[i] \cdot \lfloor q/4 \rceil + \lfloor q/8 \rceil \bmod q)/q \rfloor.$$

The HILA5 submission shows that $k'[i] = k[i]$ with probability $1 - 2^{-36}$. Let k (resp. k') be the 496-bit string given by the concatenation of the $k[i]$ (resp. $k'[i]$).

The role of r is not well described but the HILA5 design overview says that is an encrypted encoding of a part of k. It is computed by splitting k as $k = m \| z$, where m gets the first 256 bits and z the remaining 240 bits. HILA5 uses a custom-designed error-correcting code XE5 that corrects at least 5 errors to compute a 240-bit checksum s of m and then computes $r = s \oplus z$, where $\oplus$ denotes bitwise addition (XOR).

Alice computes $k' = m' \| z'$, the checksum s' on m', and applies the XE5 error correction to m', s', z' and r to correct m' to m.

3.2 Fluhrer's Attack

The chosen-ciphertext attack on HILA5 that we are going to present is a variant of the following attack against key reuse in RLWE-based key exchange protocols presented by Fluhrer in 2016 [4]. This section assumes that Bob computes the $c[i]$ and $k[i]$ in a way similar to the previous section. The $d[i]$ were added in HILA5 and will be considered in the next section.

Recall that Alice's version of the shared secret key is

$$gab + e'a,$$

where g is some large public generator element, a and b are Alice's and Bob's small private keys, and e' is a small noise vector chosen by Bob. This version of the shared secret differs from Bob's by some small error, hence they need to employ a reconciliation mechanism to arrive at the same secret bit string.

The general strategy of an evil Bob is to artificially force one (say, the first) coefficient of gab to be close to the edge M between the intervals that are mapped to bits 0 and 1 during reconciliation. An honest user would set the reconciliation bit $c[0]$ in that case, so Alice would use another mapping that is less likely to produce an error; but evil Bob does not. Since evil Bob proceeds honestly except for the first bit, he knows two possibilities for Alice's key, hence he can query Alice with one of these guesses and distinguish between 0 and 1 based on her reaction. If we assume for the moment that evil Bob can choose, hence knows, $(gab)[0]$, this tells him that $(e'a)[0]$ lies in a certain interval.

After a few queries using binary search with varying values for $(gab)[0]$, evil Bob knows the exact distance of $(e'a)[0]$ from the edge, and if he sets $e' = 1$, this distance is nothing but the first coefficient of Alice's secret key a. Note that in Fluhrer's setting the edge M is at zero and he uses b with $(gab)[0] = 1$, hence evil Bob can just multiply that b by small distances to obtain a prescribed $(gab)[0]$ when searching for $(e'a)[0]$. In our adaptation of the attack to HILA5, this step is more involved; see Sect. 4.2.

One could apply this method individually to each coefficient to extract Alice's full secret key. However, being able to recover the coefficient at one position is enough: due to the structure of the underlying ring, evil Bob can shift the ith coefficient of a into the constant term of $e'a$ by setting e' to $-x^{n-i}$, i.e., a vector with one entry of -1 and 0 elsewhere.

We now come back to the assumption made above. Notice that evil Bob does not a priori know a vector $b \in R$ such that $(gab)[0] = 1$, but he can still reasonably guess one: Alice's public key is $ga+e$ for small vectors a and e, hence if b is a small low-weight vector such that $(b \cdot (ga + e))[0]$ is close to 1, there is a good chance that in fact $(gab)[0] = 1$. Thus, while evil Bob does not have a deterministic method to find an "evil" b, he can still just make educated guesses based on Alice's public key until he finds one that works. Finding $b \in R$ with $(b \cdot (ga + e))[0]$ close to 1 is an offline computation using only Alice's public key; testing for $(gab)[0] = 1$ requires interaction with Alice.

There are several follow-ups to Fluhrer's paper, e.g. the recently posted [3], but a small and new generalization of Fluhrer's attack is sufficient to attack HILA5.

4 Chosen-Ciphertext Attack on HILA5

In this section, we describe how we circumvent the error-correction code and how to adapt Fluhrer's attack to the HILA5 case.

4.1 Working Around Error Correction

The HILA5 construction includes XE5 as an error-correcting code that is applied to the shared secret after decapsulation. Both Alice and Bob compute their version of a redundancy check, which will help Alice to correct up to 5 errors in the shared secret. The redundancy part r is divided into ten subcodewords $r = r_0, \ldots, r_9$ of variable sizes. For the purpose of the attack, these sizes do not matter, but we use the same notation L_i for the size, as in the HILA5 paper. This means we can index each $r_i = r_{(i,0)} \ldots r_{(i,L_i-1)}$ for $i \in \{0, \ldots, 9\}$.

Bob first computes his part of the HILA5 encapsulation, i.e., he computes his version of the shared secret, selects the indices that are safe to use by Alice and computes the reconciliation vector. The last 240 bits of Bob's shared secret are used in XE5 error-correction. From these bits, Bob constructs his redundancy check r', and sends this as part of the ciphertext.

Upon receiving Bob's ciphertext, Alice first computes her part of the HILA5 decapsulation, i.e., she computes her version of the shared secret. Then she computes her own redundancy check r and computes the distance r^{Δ} with Bob's r' from the ciphertext:

$$r^{\Delta} = r' \oplus r$$

To determine which bits in the shared secret are erroneous, Alice determines a weight $w_k^{\Delta} \in [0, 10]$ for each of the 256 bits by the following formula:

$$w_k^{\Delta} = r_{0,\lfloor k/16 \rfloor}^{\Delta} + \sum_{j=1}^{9} r_{j,k \bmod L_j}^{\Delta}$$

Now, if a single bit k of Alice's shared secret is flipped, it means $w_k^{\Delta} = 10$ [12, Lemma 2], and it is therefore detectable and correctable by Alice. Moreover, it is shown that XE5 corrects bit k as long as $w_k^{\Delta} \geq 6$ [12, Theorem 1], which means XE5 can correct at least 5 bits in the shared secret. This means that applying Fluhrer's original attack directly to HILA5 will not work, as Fluhrer's original attack depends crucially on the attacker's ability to detect single-bit errors in Alice's version of the shared secret. Thus, to apply Fluhrer's attack, we have to work around these error-correction abilities.

In the attack described in the next section, we focus on inducing errors only in the first bit $k = 0$ of the shared secret. This means the attacker evil Bob needs to force w_0^{Δ} to be less than 6, as this means XE5 is no longer capable of correcting the first bit. However, evil Bob needs to leave the remaining error-correction in place, otherwise he still does not know if the first bit was the only flipped bit. In order to do that, evil Bob needs to change his redundancy check r' to do exactly that. As w_0^{Δ} is obtained by summing up the first bits of the subcodeword distances r_i^{Δ}, he can flip any 5 of the bits labeled $r'_{(0,0)}$ through $r'_{(9,0)}$ to force $w_0^{\Delta} < 6$. Our attack flips the first 5 of these bits. This means in the following section we consider the issue of error-correction solved and can directly apply a modification of Fluhrer's attack.

4.2 Details of the Attack

This section elaborates evil Bob's approach to recover Alice's secret key. As mentioned before, the general procedure mimics Fluhrer's attack (Sect. 3.2). The major steps are:

1. Guess a small low-weight secret b_0 such that $(gab_0)[0]$ is at the edge M.
2. For each $\delta \in \{-16, \ldots, 16\}$, compute b_δ such that $(gab_\delta)[0] = M + \delta$.
3. For each target coefficient of Alice's secret:
   (a) Choose e' such that $(e'a)[0]$ is the target coefficient.
   (b) Perform a binary search using the b_δ to recover the target coefficient. (Alice's coefficient $(gab_\delta + e'a)[0]$ maps to a 1 bit iff $(-e'a)[0] > \delta$.)
4. If the results look "bad" after recovering a few coefficients in this way, the guess for b_0 was probably wrong and evil Bob should start over at step 1.

Note that for each oracle query, i.e., for every interaction with Alice, Bob proceeds honestly except for using specially crafted b_δ and e', setting $d_0 = c_0 = 1$, and flipping a few bits in the error correction as described in Sect. 4.1. We now explain and analyze the steps above in more detail.

Forcing Coefficients Near the Edge. In HILA5's reconciliation mechanism, there is no edge at zero for any choice of reconciliation bit, hence Fluhrer's attack does not apply without modifications. We chose to set the reconciliation bit c_0 to 1 and attack the edge at

$$M = \lfloor q/8 \rceil = 1536.$$

To perform the binary search for Alice's secret coefficients in the attack, we need to find small low-weight vectors b_δ such that

$$(gab_\delta)[0] = M + \delta$$

for all δ with $|\delta| \leq 16$. (As mentioned in Sect. 3.2, Fluhrer's evil Bob attacked $M = 0$, thus he could guess b_1 based on Alice's public key and set $b_\delta = \delta \cdot b_1$.) One could of course try to guess each b_δ individually based on Alice's public key, but as we want to get all b_δ right at the same time, this has exponentially low success probability. Instead, we make use of a special property of the M used in HILA5: The inverse

$$M^{-1} \bmod q = -8$$

is small.[2] Hence, as soon as evil Bob successfully guessed b_0, he may simply set

$$b_\delta = (1 + \delta M^{-1} \bmod q) \cdot b_0.$$

In our case, we choose b_0 with only two non-zero coefficients from $\{\pm 1\}$, thus b_δ will have only two non-zero coefficients bounded by $1 + 8\delta$. This property is necessary to make sure evil Bob can actually know what Alice's version of the shared secret will be (except for the target bit that leaks information): If the coefficients of b_δ are too large, the error $eb - e'a$ between Alice's and Bob's shared secrets becomes too large to recover from and their secrets will mismatch no matter what the value of the attacked bit is. In theory, with these parameters we still expect a tiny possibility of unintended errors, but this happens so rarely that it is not an issue in practice. If it ever does occur, Bob can detect that his recovered secret key is wrong and simply start over with a new b_0.

When evil Bob chooses a random b_0 with two non-zero coefficients in $\{\pm 1\}$ and with $(Ab_0)[0] = M$, the probability that in fact $(gab_0)[0] = M$ holds is just the probability that two Ψ_{16}-distributed values sum to zero:

$$\sum_{i=0}^{32} \binom{32}{i}^2 / 2^{64} \approx 9.9\%,$$

hence he can expect to find a good b_0 after about 10 tries. Since A can be approximated by a uniformly distributed sequence over $\mathbb{Z}_q$, the expected number of ± 1-combinations of two coefficients of A which equal M is

$$\binom{1024}{2} \cdot 4/q \approx 170.$$

[2] Note that this also holds for some other "natural" choices of M as rounded fractions of q, but it is not automatically true for any conceivable M.

Hence, the probability that evil Bob exhausts this pool of choices without finding a good b_0 is roughly 2^{-25}.

(If this ever happens, then evil Bob can still try a larger interval, i.e., search for b_0 with $|(Ab_0)[0] - M| \leq K$ for some small K. This would in theory work for a wider range of keys, but the expected number of wrong guesses grows slightly. One could also choose three non-zero coefficients in b_0, although this increases the chance of unintended errors in Alice's shared secret. We have not had any problems with $K = 0$ in practice.)

Detecting Bad Guesses. After choosing b_0 based on Alice's public key as described above, evil Bob may just go ahead and try to recover Alice's secret key using that b_0. If it is correct, he will of course find a sequence that looks like it was sampled from the Ψ_{16} distribution. If b_0 is bad, say, $(gab_0)[0] = M + \gamma$ for some small $\gamma \neq 0$, then

$$(gab_\delta)[0] = M + \delta + \gamma - 8\delta\gamma,$$

hence typically $(gab_\delta)[0]$ is considerably smaller than M if $\delta > 0$ and considerably larger if $\delta < 0$; in both cases Alice's secret $(e'a)[0]$ is dominated by $\delta + \gamma - 8\delta\gamma$, which means the oracle output does not depend on the secret. This implies the binary search will always converge to 0 or -1 when b_0 is bad. (For $\delta = 0$, the behavior *does* depend on $(e'a)[0]$ since γ is small, so both cases really occur.) Evil Bob can detect this failure mode by determining a few coefficients and checking whether all of them are in $\{0, -1\}$. If this is the case, evil Bob simply starts over with a new b_0. The probability that an actual secret key starts with a sequence of k coefficients from $\{0, -1\}$ is about 0.27^k, hence setting $k = 8$ reduces the probability of a false negative to roughly 2^{-15}. There is a small probability of false positives if evil Bob uses only this heuristic (e.g., when $|\gamma| = 1$), but this can easily can be detected using statistical methods (the recovered sequence will not be Ψ_{16}-distributed) or by simply testing the obtained secret key in the end and running the attack again if it failed. In practice the heuristic works fine.

The Number of Queries. Assuming we already have a good b_0, the binary search needs an expected $5 + \varepsilon$ queries to the oracle to recover one coefficient.[3] Since evil Bob decides whether he has a good b_0 based on the first few coefficients that he obtains using that b_0, he usually wastes a few hundred queries on guesses for b_0 that turn out to be useless: If he looks at the first 8 coefficients obtained from each b_0 as suggested above, this adds expected ≈ 400 queries to the 5120 needed to recover all the coefficients. In summary, evil Bob will with overwhelming probability recover Alice's secret key in less than 6000 queries.

Evil Bob can trade computation for a smaller number of queries: retrieve some coefficients, and reduce the original lattice problem to low enough dimension to solve by computation.

[3] The ε arises from the fact that Ψ_{16} samples from $33 > 2^5$ distinct values, but the extremal values occur so rarely that $\varepsilon \approx 2^{-27}$.

4.3 Implementation

We implemented a proof of concept of the attack in Python, reusing portions of the HILA5 reference implementation via the `ctypes` library. The only modifications we made to the reference implementation were making some functions non-`static` to be able to call them from within Python, and adding extra parameters to the encapsulation function (not used by Alice) such that evil Bob can override his private values b and e'. The complete attack script can be found at https://helaas.org/hila5-20171218.tar.gz. As expected, we have never observed the attack script failing to recover Alice's key. The empirical number of queries matches the theoretical prediction made above.

5 HILA5 Security Claims

In this section, we discuss our interpretation of security claims made by both the paper and NIST submission of HILA5, which motivated this paper.

NIST does not require IND-CCA2 security for KEM and PKE submissions. Instead it requires submissions to say whether they are aiming for IND-CCA2 security or merely for IND-CPA security.

IND-CPA security is adequate in the context of key exchange in TLS, if a new public key is generated for each TLS session. For example, New Hope [2] appears to be safe for use in TLS. New Hope does not aim for IND-CCA2 security, and specifically warns against using a key more than once: "No key caching ... it is crucial that both parties use fresh secrets for each instantiation".

We emphasize that our attack does not break the IND-CPA security of HILA5. If HILA5 were clearly labeled as aiming merely for IND-CPA security then our attack would merely be a cautionary note, showing the importance of not reusing keys.

However, HILA5 went beyond claiming IND-CPA security. There are some undefined words in the HILA5 security claims, but the most natural interpretation of the security claims is that the HILA5 PKE provides IND-CCA2 security. There is certainly a high risk of the claims being interpreted in this way by potential users. Our attack shows that the HILA5 PKE does not provide IND-CCA2 security.

There is even a risk of users thinking that the HILA5 KEM is being claimed to provide IND-CCA2 security.[4] The HILA5 submission document does not say that the HILA5 KEM security target is merely IND-CPA. Our attack shows that the HILA5 KEM does not provide IND-CCA2 security.

We give four quotes from [12] to explain why the HILA5 security claims are most naturally interpreted as claiming IND-CCA2 security for the HILA5 PKE. We have not found anything in [12] or [13] indicating a different interpretation.

[4] Adam Langley posted an online table of speeds for announced KEMs submitted to NIST. He wrote "I only want to list CCA-secure KEMs here". He listed HILA5, and accepted a correction from the HILA5 author regarding the speed of HILA5. After the correction, HILA5 had the fastest decapsulation in the entire table.

> [12, Section 1]: *The HILA5 KEM can be adopted for public key encryption in straightforward fashion. We recommend using the AES-256-GCM AEAD [FIP01, Dwo07] in conjunction with the KEM when public key encryption functionality is desired.*

The details of this "conjunction" are not formally defined. The most natural interpretation is that this is the HILA5 PKE, using the session key produced by the HILA5 KEM as the AES-GCM key.

> [12, Section 4.1]: *NIST requires at least IND-CPA [BDPR98] security from a KEM scheme (Section 1.6). ... The design also provides IND-CCA secure KEM-DEM [CS03] public key encryption if used in conjunction with an appropriate AEAD [Rog02] such as NIST approved AES256-GCM [FIP01, Dwo07]. These properties are derived from [Pei14].*

This is a claim of IND-CCA security for a PKE. "IND-CCA" in the literature usually means IND-CCA2, although sometimes it means merely IND-CCA1. The PKE is not formally defined, but again the most natural interpretation is simply that the session key produced by the HILA5 KEM is the AES-GCM key used to encrypt a user-supplied plaintext. Our attack shows that this PKE does not even provide IND-CCA1 security, let alone IND-CCA2 security.

Our attack does not work against what we call the HILA5FO PKE (see Sect. 2.4), a more complicated PKE using the Fujisaki–Okamoto transformation. This transformation is also mentioned in "[Pei14]" as a way to achieve IND-CCA security. It is conceivable that the HILA5 submission was alluding to a PKE of this type. However, this interpretation does not appear to be compatible with the statement "Ciphertext size: 2012 Byte expansion (KEM) + payload + MAC" in [12, Sect. 6]; the HILA5FO ciphertext size is 32 bytes larger than this.

> [12, Section 4.9]: *For active security we suggest that* K *is used as keying material for an AEAD (Authenticated Encryption with Associated Data) [Rog02] scheme such as AES256-GCM [Dwo07, FIP01] or Keyak [BDP$^+$16] in order to protect message integrity.*

Here "K" is defined as the session key produced by the HILA5 KEM. In the context of KEMs and PKEs, "active security" is normally interpreted as IND-CCA2 security, although it might have other interpretations. The authentication in AES-GCM prevents modifications to the message encrypted by AES-GCM, but this is not enough to stop active attacks, since it does not protect the underlying KEM.

> [12, Section 6.1]: *HILA5 is essentially drop-in compatible with current public key encryption applications. There are no practical usage restrictions.*

Security against chosen-ciphertext attacks is essential for a wide range of current PKE applications, so this would appear to include a claim of CCA security for the HILA5 PKE. However, our attack retrieves the secret key from the HILA5 PKE.

References

1. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: STOC, pp. 284–293. ACM (1997)
2. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: USENIX Security Symposium, pp. 327–343. USENIX Association (2016)
3. Ding, J., Alsayigh, S., Saraswathy, R.V., Fluhrer, S.R., Lin, X.: Leakage of signal function with reused keys in RLWE key exchange. In: ICC, pp. 1–6. IEEE (2017)
4. Fluhrer, S.R.: Cryptanalysis of ring-LWE based key exchange with key share reuse. IACR Cryptology ePrint Archive 2016/085 (2016). https://ia.cr/2016/085
5. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_34
6. Hall, C., Goldberg, I., Schneier, B.: Reaction attacks against several public-key cryptosystem. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 2–12. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-540-47942-0_2
7. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054868
8. Hoffstein, J., Silverman, J.H.: Reaction attacks against the NTRU public key cryptosystem. NTRU Cryptosystems Technical report 015, version 2 (2000). https://web.archive.org/web/20000914041434/http://www.ntru.com:80/NTRUFTPDocsFolder/NTRUTech015.pdf
9. Howgrave-Graham, N., Nguyen, P.Q., Pointcheval, D., Proos, J., Silverman, J.H., Singer, A., Whyte, W.: The impact of decryption failures on the security of NTRU encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 226–246. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_14
10. National Institute of Standards and Technology: Announcing request for nominations for public-key post-quantum cryptographic algorithms (2016). https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms
11. Peikert, C.: Lattice cryptography for the internet. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 197–219. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_12
12. Saarinen, M.-J.O.: HILA5: key encapsulation mechanism (KEM) and public key encryption algorithm (2017). Submission to NIST: https://github.com/mjosaarinen/hila5/blob/master/Supporting_Documentation/hila5spec.pdf
13. Saarinen, M.-J.O.: HILA5: on reliability, reconciliation, and error correction for ring-LWE encryption. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 192–212. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72565-9_10
14. Verheul, E.R., Doumen, J.M., van Tilborg, H.C.A.: Sloppy Alice attacks! Adaptive chosen ciphertext attacks on the McEliece public-key cryptosystem. In: Blaum, M., Farrell, P.G., van Tilborg, H.C.A. (eds.) Information, Coding and Mathematics. ECS(CIT), vol. 687, pp. 99–119. Springer, Boston (2002). https://doi.org/10.1007/978-1-4757-3585-7_7