From: Guillaume Endignoux <guillaume.endignoux@gmail.com>
Sent: Friday, March 08, 2019 12:46 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: ROUND 2 OFFICIAL COMMENT: BIKE

Dear BIKE submitters,

I had a look [1] at the reference implementation of BIKE and have the following remarks regarding some (minor) differences that I found compared to the specification. I didn't look at any of the optimized implementations.

- The algorithm 5 ("generate pseudorandom bytes of odd weight") doesn't seem to be the same in the specification and in the implementation. On line 2 in the specification, it is specified that to make the weight odd the first bit is flipped, whereas in sampling.c:93 the bit to be flipped is drawn uniformly at random. It's not obvious to me why the implementation is different nor why this change would be necessary, the specification seems simpler and faster.

- The specification of algorithm 6 seems to be missing the "masking" step that occurs in the implementation. In the specification, it draws uniform 32-bit integers (by drawing 4 bytes from the PRF) and then checks if they are smaller than "len" (which doesn't happen often, given that "len" is equal to "r" or "n", which range between 10163 and 72262, much lower than 2^32). On the contrary, the implementation applies a mask to the uniform 32-bit integer, to retain only the lower "ceil(log2(len))" bits, so that the rejection rate is less than 50% for each integer. The implementation is clearly much more efficient.

By the way, it seems to me that the "bit_scan_reverse" function at utilities.h:76 actually returns "floor(log2(len)) + 1" instead of "ceil(log2(len))", although this shouldn't matter as the length is never a power of two for the proposed parameters.

- The approximated formula for the threshold on page 17 for BIKE-1 and BIKE-2, security level 1 is "ceil(13.530 + 0.0069722 |s|)", but the implementation in defs.h:90 uses 0.0069721 for the second constant. Probably not significant given that |s| is a small integer and the result is rounded to an integer, but it would be nice to clarify. I haven't found similar typos for the other security levels.

- The algorithm 3 requires parameters "S" and "delta". It is mentioned that these depend on "r, w, t", but I couldn't find their values in the specification, nor the rationale for them. I found them as "DELTA_BIT_FLIPPING" and "S_BIT_FLIPPING" in the implementation.

- The description of algorithm 7 ("parallel hashing") has some unclear points to me. From what I understood by re-implementing it to double-check, it first splits the input into 8 slices that each contain "alpha" full 128-byte blocks plus a 111-byte remainder (so that after appending the SHA-384 padding this remainder is extended to a full 128-byte block). It then concatenates the SHA-384 hashes of the 8 the slices with the remainder of the input, and applies SHA-384 on it.

My first comment is that in the specification the remainder "Y" is before the hashes "X[7] ... X[0]", whereas in parallel_hash.c:49, the "yx" structure actually contains x before y. From this I understand that the specification of this algorithm denotes bytes in right-to-left order (consistent with the "array" notation), which doesn't seem intuitive.

A second point is that "Y" in the specification should have "lrem" bytes (which depends on the input length "la"), but in parallel_hash.c:54, "y" is instead fixed to "s*hbs" bytes, and all these bytes are hashed on line 104 (using "sizeof(yx)"). So in the implementation, "y" is padded with zero bytes to fill "s*hbs" bytes that are hashed, contrary to what is written in the specification.

Best regards,

[1] The files I looked at (both obtained from https://bikesuite.org/) have the following SHA-256 hashes.
da77f33b0f33d704309fc926a2796638ac671a4b82f470a16cce29f9d192407e BIKE.pdf
eec9201859598c7d0bae505651b19b04ef40322445974ee97c06210a322364d3 Reference_Implementation.2018.06.29.zip

From:	Guillaume Endignoux <guillaume.endignoux@gmail.com>
Sent:	Monday, March 11, 2019 4:34 PM
To:	pqc-comments
Cc:	pqc-forum@list.nist.gov
Subject:	ROUND 2 OFFICIAL COMMENT: BIKE

Dear BIKE submitters,

I had a closer look at the decapsulation method for BIKE, in particular algorithm 3 of the specification ("one-round bit flipping algorithm").

For the loops at lines 7-10 (loop A) and 13-15 (loop B), I considered the reference implementation to derive the "maximum running time", given that the specification doesn't specify this limit. In the implementation (see defs.h:150-151) the criterion is that each of these loops stops after at most "n" iterations (for the "n" parameter defined in table 3).

I tested the decapsulation algorithm on a "malformed" cryptogram: instead of using the result of an encapsulation, I filled the cryptogram with uniformly random bytes (it seems that this yields an invalid cryptogram with high probability). It turns out that on this malformed input, the decoding loops A and B don't finish "naturally" but reach the limit of n iterations.
I tested the decapsulation method of BIKE-1 Level1, and on this "uniformly random bytes" input, the reference implementation took around 40 seconds to complete (and return "decoding failure") on my machine, which is much slower than what it takes to decode a valid cryptogram (see table 6 of the specification).

This looks like this could be turned into a denial-of-service attack against the decoder.

This raises several questions.

1) How to mitigate this for BIKE?

After taking a closer look on what happens on the malformed input, it seems that loop A quickly converges to a slightly improved syndrome but without going below a weight of "S". After that, the following iterations don't change anything (the "check" function at line 9 of the specification doesn't find a bit to flip).
I modified the exit criterion of loop A to break as soon as an iteration doesn't change the syndrome. After this "optimization", the decoding completes in around 2 seconds, an order of magnitude faster than the textbook decoding, but still several orders of magnitude slower than on a valid input.

Regarding loop B, I noticed that on the malformed input the error weight quickly increases beyond "t", whereas a properly encapsulated input must have an error weight of exactly "t". Even though the bit-flipping may "unflip" a bit and thereby decrease the error weight, it seems to me that the decoding could stop early and fail if the error weight reaches "t + some small constant". This would likely avoid running n iterations of loop B.

Also, in practice, valid inputs seem to take much less than "n" iterations to be decoded, so the hard limit could be reduced well below "n".
It would be interesting to see how better bounds could be derived (for a given failure probability).

2) Does NIST consider decapsulation of malicious inputs a valid threat model for the evaluation of post-quantum algorithms?

As far as I'm aware, the KATs only test decapsulation of the results of encapsulation.

3) How to evaluate the performance of algorithms that are slower on invalid inputs?

In particular, does NIST plan to evaluate the average/median runtime on valid inputs, the worst-case runtime on valid inputs, the worst-case runtime on malicious/random inputs?
Or will constant-time algorithms be required?

Similarly, do benchmark frameworks such as SUPERCOP plan to evaluate decapsulation performance on random inputs?

Best regards,
Guillaume Endignoux

From: 'Apon, Daniel C. (Fed)' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Tuesday, March 26, 2019 6:12 PM
To: pqc-forum@list.nist.gov; Guillaume Endignoux
Subject: Re: [pqc-forum] ROUND 2 OFFICIAL COMMENT: BIKE

Hi Guillaume,

"2) Does NIST consider decapsulation of malicious inputs a valid threat model for the evaluation of post-quantum algorithms?"

Yes. Mauling attacks (i.e. attacks based on malformed or malicious ciphertexts) are formally captured by the standard IND-CCA2 security definition.

"3) How to evaluate the performance of algorithms that are slower on invalid inputs?"

As has been previously stated, we **strongly** encourage that all 2nd Round teams provide implementations that are in constant-time.

Hope this helps,
--Daniel Apon, NIST PQC

From: Guillaume Endignoux <guillaume.endignoux@gmail.com>
Sent: Monday, March 11, 2019 4:34:06 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: [pqc-forum] ROUND 2 OFFICIAL COMMENT: BIKE

Dear BIKE submitters,

I had a closer look at the decapsulation method for BIKE, in particular algorithm 3 of the specification ("one-round bit flipping algorithm").

For the loops at lines 7-10 (loop A) and 13-15 (loop B), I considered the reference implementation to derive the "maximum running time", given that the specification doesn't specify this limit. In the implementation (see defs.h:150-151) the criterion is that each of these loops stops after at most "n" iterations (for the "n" parameter defined in table 3).

I tested the decapsulation algorithm on a "malformed" cryptogram: instead of using the result of an encapsulation, I filled the cryptogram with uniformly random bytes (it seems that this yields an invalid cryptogram with high probability). It turns out that on this malformed input, the decoding loops A and B don't finish "naturally" but reach the limit of n iterations.
I tested the decapsulation method of BIKE-1 Level1, and on this "uniformly random bytes" input, the reference implementation took around 40 seconds to complete (and return "decoding failure") on my machine, which is much slower than what it takes to decode a valid cryptogram (see table 6 of the specification).

This looks like this could be turned into a denial-of-service attack against the decoder.

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Saturday, April 06, 2019 1:13 PM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] ROUND 2 OFFICIAL COMMENT: BIKE
Attachments: signature.asc

Guillaume Endignoux writes:
> Similarly, do benchmark frameworks such as SUPERCOP plan to evaluate
> decapsulation performance on random inputs?

There's already a little bit of work in this direction: look for "forgery" in crypto_{box,secretbox,aead}/measure.c. In one case, aead, the forgery-handling speeds are included in the tables posted online.

What makes this difficult to do properly is that the goal isn't really to benchmark random messages; the goal is to benchmark the invalid messages that will take the maximum amount of time. Finding really slow inputs can be easy for an attacker but much harder for automated mechanisms (including fuzzing), and then benchmarking only what's found by the automated mechanisms will give users a very wrong idea of what the attacker is capable of doing. I haven't seen any problematic cases for aead but the public-key situation is much more complicated.

I agree that denial of service can't be ignored---it can be vastly more important than the performance of legitimate messages.

---Dan

From: Mike Hamburg <mike@shiftleft.org>
Sent: Monday, July 1, 2019 6:36 PM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 2 OFFICIAL COMMENT: BIKE

Hello all,

I have a concern with BIKE’s implementation of the Fujisaki-Okamoto transform.

The BIKE specification states that its CCA instances are derived using the “FO^{\not\bot}” (Fujisaki-Okamoto with implicit rejection) transform from Hofheinz-Hövelmanns-Kiltz 2017. This transform recomputes the ciphertext c as c’ := Enc(pk,m’). If c’ != c, it rejects and returns the key H(s,c), where s is a PRF key that’s part of the private key.

However, as shown on pages 13-15 of the BIKE specification, BIKE instead returns H(s,c’). That is, it hashes the re-computed ciphertext instead of the given ciphertext. The reference implementation matches this behavior. The security reduction in HHK’17 doesn’t hold for this variant: indeed it would not be secure with an LPR-like KEM, and I’m not sure it’s secure for BIKE.

The difference is important because (if any function of the returned shared secret leaks to the attacker), it allows the attacker to determine when changes to the ciphertext will result in changes in the decoded message or errors. I’m not sure of the security implications beyond messing up the proof, but here are some first guesses:

BIKE-1: This at least leaks whether the ciphertext was rejected: if it was rejected, then sending (c_0+m’ f_0, c_1+m’ f_1) for random m’ will give the same shared secret, and if it was accepted then it will not. Probably this is not a big deal. More concerning is that if decoding fails, then this exposes the hashed output of the failed decoder state. I haven’t studied the decoder carefully enough to know whether this matters, but it’s at least a bit concerning.

BIKE-2: Again I’d be concerned about exposing the decoder’s state.

BIKE-3: I’m concerned that this is actually vulnerable to CCA attack, since it’s very similar to LPR. In particular, the attacker can probe which messages decode and which don’t by changing e; if the message still decodes then this won’t changed the derived secret.

I suggest that the BIKE authors change to computing H(s,c) instead of H(s,c').

Cheers,
— Mike Hamburg

From: BIKE Team <team@bikesuite.org>
Sent: Monday, July 1, 2019 7:39 PM
To: pqc-forum
Subject: [pqc-forum] ROUND 2 OFFICIAL COMMENT: BIKE

Dear all,

This is to announce that we have made some refinements to the BIKE suite.

After the Round 2 submission, we identified a few things that could be improved. Therefore, we proactively worked on them, and we are now happy to share the results with the community.

We would like to request to NIST to consider the updated version with these improvements for the standardization process. The updated spec (v3.1) and the corresponding reference and optimized implementations have been made available in our website: www.bikesuite.org.

Below, we list the changes. Please feel free to share your comments and/or questions, if any.

Best Regards,
BIKE Team

--

Summary of changes:

- BIKE CCA Flows: In Round 2, we introduced the Backflip decoding algorithm, that allows to reach much lower decoding failure rates (e.g. DFR=2^{-128} for random error vectors). As a result, we were able to introduce CCA-secure BIKE variants based on static keys, in addition to the CPA-secure ones submitted in Round 1 based on ephemeral keys. After Round 2 submission, we identified that our approach for the CCA flows was not optimal. The error vectors used in the schemes are required to be generated at random, but in our previous flows there was no effective way to enforce this. As a result, we refined our CCA flows to require that the error vector is generated by applying a one-way function to a (random) message, or seed. In this way, no one is able to craft error vectors at will, unless she/he is able to defeat the security assumption related to the one-way function. This simple change that has marginal performance impact. We benchmarked these new flows, updated the spec, and re-generated the associated KAT's. Note that this change automatically addresses the official comment recently submitted by Mike Hamburg to the NIST mail list on July 1st, 2019.

- BIKE-3 Optimizations: Half of BIKE-3 public key is a random vector "g". We believe we have not emphasized enough the fact that g does not need to be transmitted. A short 256 bits seed is enough, assuming that a secure PRNG is agreed upon by the communicating parties. Therefore, we explicitly added this flavor of BIKE-3 (based on a seed instead of g) to our reference/optimized implementations. We benchmarked BIKE-3 based on the seed, and updated the spec accordingly. Additionally, for BIKE-3 variants, the error vectors (e0, e1, e) are now derived from a common seed. As a result, in decapsulation time, the noisy-syndrome decoding needs to return the additional error vector (e) as well. This is accomplished in a very simple way: e = transpose(s'), where s' is the final syndrome vector after the decoding algorithm finds (e0, e1).

- Mismatch in threshold computation: We identified a mismatch in the way the threshold for the decoding algorithm is computed in the reference/optimized implementation when compared to spec. We updated the implementations to match the spec. This accelerates decoding on average.

- Memory leak in CCA reference/optimized implementation: We identified a problem in the reference implementation, where some allocated memory is not completely freed. This does not affect the computations, but eventually, long runs would exhaust the available memory and the OS would crash the execution. We fixed that and updated reference/optimized implementations of CCA flows.

- Faster polynomial multiplication using GF2X: we used a build of NTL library with GF2X library support. This led to a speedup for polynomial multiplication, and as a result improved the overall performance of the KEM steps by a factor ~2x to ~5x. The performance numbers have been updated in the spec.

From: BIKE Team <team@bikesuite.org>
Sent: Monday, July 1, 2019 7:47 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: ROUND 2 OFFICIAL COMMENT: BIKE

Dear Mike,

We just published an update to our BIKE suite.

Among other things, the CCA flows have been refined in a way that automatically addresses your comment.

Please let us know if any further questions or comments.

Best Regards,
BIKE Team

On Monday, July 1, 2019 at 6:36:52 PM UTC-4, Mike Hamburg wrote:

> Hello all,
>
> I have a concern with BIKE's implementation of the Fujisaki-Okamoto transform.
>
> The BIKE specification states that its CCA instances are derived using the "FO^{\not\bot}" (Fujisaki-Okamoto with implicit rejection) transform from Hofheinz-Hövelmanns-Kiltz 2017. This transform recomputes the ciphertext c as c' := Enc(pk,m'). If c' != c, it rejects and returns the key H(s,c), where s is a PRF key that's part of the private key.
>
> However, as shown on pages 13-15 of the BIKE specification, BIKE instead returns H(s,c'). That is, it hashes the re-computed ciphertext instead of the given ciphertext. The reference implementation matches this behavior. The security reduction in HHK'17 doesn't hold for this variant: indeed it would not be secure with an LPR-like KEM, and I'm not sure it's secure for BIKE.
>
> The difference is important because (if any function of the returned shared secret leaks to the attacker), it allows the attacker to determine when changes to the ciphertext will result in changes in the decoded message or errors. I'm not sure of the security implications beyond messing up the proof, but here are some first guesses:
>
> BIKE-1: This at least leaks whether the ciphertext was rejected: if it was rejected, then sending (c_0+m' f_0, c_1+m' f_1) for random m' will give the same shared secret, and if it was accepted then it will not. Probably this is not a big deal. More concerning is that if decoding fails, then this exposes the hashed output of the failed decoder state. I haven't studied the decoder carefully enough to know whether this matters, but it's at least a bit concerning.
>
> BIKE-2: Again I'd be concerned about exposing the decoder's state.
>
> BIKE-3: I'm concerned that this is actually vulnerable to CCA attack, since it's very similar to LPR. In particular, the attacker can probe which messages decode and which don't by changing e; if the message still decodes then this won't changed the derived secret.

From: Mike Hamburg <mike@shiftleft.org>
Sent: Monday, July 1, 2019 8:05 PM
To: BIKE Team
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] Re: ROUND 2 OFFICIAL COMMENT: BIKE

Hi BIKE team,

Thanks for the update. Funny timing, I should have checked with you before posting.

I haven't checked the proofs, but the spec changes appear to resolve my concerns about hashing the rederived ciphertexts.

I noticed that you do not hash the public key when deriving the noise vectors. This was a problem for LAC in the first round, due to precomputed multi-target CCA failure attacks. Is it your sense that this doesn't matter for BIKE because, eg, all (m,e_0,e_1,e') tuples have close to the same failure probability?

You have a small but annoying typo on BIKE-3 CCA by the way, where it says |(e'| \neq t/2.

Another curiosity: my browser doesn't like your site's cert. It appears to be issued with a CN of *.web-hosting.com, rather than www.bikesuite.org.

Best regards,
— Mike

> On Jul 1, 2019, at 4:47 PM, BIKE Team <team@bikesuite.org> wrote:
>
> Dear Mike,
>
> We just published an update to our BIKE suite.
>
> Among other things, the CCA flows have been refined in a way that automatically addresses your comment.
>
> Please let us know if any further questions or comments.
>
> Best Regards,
> BIKE Team
>
> On Monday, July 1, 2019 at 6:36:52 PM UTC-4, Mike Hamburg wrote:
>
> > Hello all,
> >
> > I have a concern with BIKE's implementation of the Fujisaki-Okamoto transform.
> >
> > The BIKE specification states that its CCA instances are derived using the "FO^{\not\bot}" (Fujisaki-Okamoto with implicit rejection) transform from Hofheinz-Hövelmanns-Kiltz 2017. This transform recomputes the ciphertext c as c' := Enc(pk,m'). If c' != c, it rejects and returns the key H(s,c), where s is a PRF key that's part of the private key.
> >
> > However, as shown on pages 13-15 of the BIKE specification, BIKE instead returns H(s,c'). That is, it hashes the re-computed ciphertext instead of the given ciphertext. The reference implementation matches this behavior. The security reduction in HHK'17 doesn't hold for this variant: indeed it would not be secure with an LPR-like KEM, and I'm not sure it's secure for BIKE.

From:	pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent:	Friday, February 7, 2020 7:57 AM
To:	pqc-forum
Subject:	[pqc-forum] ROUND 2 OFFICIAL COMMENT: BIKE

Hi Again,

BIKE-1 (CPA) has the same issue as discussed earlier for ROLLO-1; setting the public key (f0,f1) to all zeros will set the shared secret of both parties to an easily decodable value regardless of the secret key. This can be a problem for the ephemeral key exchange. BIKE-2 and BIKE-3 behave in a more complex manner. I'd suggest checking (f0,f1) for trivial values (as is done e.g. in Diffie-Hellman) and/or including the public key in the shared secret computation hash also in this instance.

Tested with Optimized_Implementation-2019.06.30.1.zip

Some other issues:

The current BIKE specification (BIKE-Spec-2019.06.30.1.pdf) defines BIKE-2 public key to be simply h = h1/h0 (Section 2.1.2). However other parts of the specification (such as Table 2) write it as a pair of values (1, h1*h0^-1).

From the KAT files (KAT-2019.06.30.1.zip) one can indeed observe that the BIKE-2 public keys look like this:

pk = 01000000[.. few thousand zeros ...]0000000146302EE5..

(KAT/INDCCA/BIKE2CCA/PQCkemKAT_BIKE2-Level1_8838.rsp)

So numeric constant "1" is always explicitly encoded in the public key with the result that about half of any BIKE-2 public key consists of just zero bytes. I find it quite puzzling why the authors have chosen to do so when the main advantage of BIKE-2 is its reduced bandwidth requirement.

Also, note that the BIKE-2 optimized implementation (Optimized_Implementation-2019.06.30.1.zip) does not match with the KAT test vectors. This is apparently a "feature" of the batch key generation discussed in Section 3.9 of the specification. The batch key generation code in the optimized implementation sidesteps the NIST API and thread-safety rules and generates large static tables for key generation. If the key generation code from the reference implementation is lifted into the optimized implementation, the output is consistent with the test vectors.

Cheers,
- markku

Dr. Markku-Juhani O. Saarinen <mjos@pqshield.com> PQShield, Oxford UK.

ps. It was not too difficult to write the necessary arithmetic operations to remove dependencies on NTL and OpenSSL libraries. This seems to make the optimized code for BIKE-1/2/3 sufficiently small and fast to even run on the reference Cortex M4 embedded target.

However, the sticking point for the publication of the size/speed optimized code is the use of proprietary ParallelizedHash (Algorithm 9 -- patented by BIKE authors in US 8856546B2) whose intellectual property status is still not clarified. It is clearly unnecessary and actually slower than NIST standard hashes on most embedded (non-SIMD) targets. Hashing is not a bottleneck for BIKE anyway. So it would be great if test vectors with some standard hash were published.

From: pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent: Saturday, February 8, 2020 11:35 AM
To: pqc-forum
Subject: [pqc-forum] Re: ROUND 2 OFFICIAL COMMENT: BIKE

Hello,

I can see that there is now a 3.2 version of BIKE available [BIKE]. However checks have not been added and because the CCA transform is also implemented badly, the problem persists for both CPA and CCA versions of BIKE-1. (Yes Dan, it's also a "CCA" problem)

The main problem is the "subspace" attack (analogous to a Diffie-Hellman without group checks) and a secondary problem is that the BIKE1-CCA encapsulation fails to use the message to randomize output as required by the FO transform [HHK].

BIKE1-CCA Encaps (Section 2.2.1, [BIKE]):

1. Generate m <-$- R uniformly at random.
2. Compute (e0, e1) <- H(m*f0, m*f1).
3. Compute c = (c0, c1) <- (m*f0 + e0, m*f1 + e1).
4. Compute K <- K(m*f0, m*f1, c).

Note that if the public key (f0,f1) = (0,0) then the output is exactly as if Bob had chosen m = 0 just by happenstance. The random quantity m is used nowhere without multiplication with either f0 or f1, hence K is actually always the same, as is the ciphertext, thanks to derandomization.

Furthermore the decapsulation and CCA re-encryption both succeed regardless of the secret key, as noted previously for the CPA version.

I've verified this with the new reference implementation.

If one sets (f0,f1) = (0,0) for BIKE1-CCA, the resulting shared secret from encapuslation and decapsulation is always
609765A51A1EA150A48805AF6E409054CC49671F6D8629313AB6FBAF2CB0BED6
or
E23623BC5ACE9F29EEFA6EE352016C214A9D461D3595D9EAB037FC8CE34E3A18
for the new version.

For CPA version the shared secrets are not constant as there is no derandomization, but they match between Alice and Bob and can be easily recovered by an attacker.

All functions return success (as expected).

[HHK] Dennis Hofheinz, Kathrin Hövelmanns, Eike Kiltz, "A Modular Analysis of the Fujisaki-Okamoto Transformation" TCC 2017, https://eprint.iacr.org/2017/604

[BIKE] Rafael Misoczki et al, "BIKE: Bit Flipping Key Encapsulation" https://bikesuite.org/files/round2/spec/BIKE-Spec-2020.02.07.1.pdf

Cheers,
- markku

Dr. Markku-Juhani O. Saarinen <mjos@pqshield.com> PQShield, Oxford UK.

From: pqc-forum@list.nist.gov on behalf of Rafael Misoczki <rafa.misoczki@gmail.com>
Sent: Saturday, February 8, 2020 12:20 PM
To: pqc-forum
Subject: [pqc-forum] Re: ROUND 2 OFFICIAL COMMENT: BIKE

Dear Dr. Sarineen,

We appreciate your continuous interest on BIKE.

The version 3.2 of our spec implements a single change to our submission: replacement of the ParallelHash technique by SHA2, as indicated in our website: https://bikesuite.org/spec.html For more details, please see the message sent to the NIST Forum in the corresponding thread.

Regarding the comments about injecting a zero-value public key, this is trivially addressed by ensuring that public keys are authenticated.

Regarding BIKE-2 public key, the first element is a constant (1) which can be implicitly assumed (and omitted) in a real-world application. We left its explicit representation in the KAT's and associated implementations for the sake of simplicity, thus unifying most of the API's in the functions called by all BIKE-1/2/3.

Best regards,
Rafael

Em sábado, 8 de fevereiro de 2020 08:35:07 UTC-8, Markku-Juhani O. Saarinen escreveu:

> Hello,
>
> I can see that there is now a 3.2 version of BIKE available [BIKE]. However checks have not been added and because the CCA transform is also implemented badly, the problem persists for both CPA and CCA versions of BIKE-1. (Yes Dan, it's also a "CCA" problem)
>
> The main problem is the "subspace" attack (analogous to a Diffie-Hellman without group checks) and a secondary problem is that the BIKE1-CCA encapsulation fails to use the message to randomize output as required by the FO transform [HHK].
>
> BIKE1-CCA Encaps (Section 2.2.1, [BIKE]):
>
> 1. Generate m <-$- R uniformly at random.
> 2. Compute (e0, e1) <- H(m*f0, m*f1).
> 3. Compute c = (c0, c1) <- (m*f0 + e0, m*f1 + e1).
> 4. Compute K <- K(m*f0, m*f1, c).
>
> Note that if the public key (f0,f1) = (0,0) then the output is exactly as if Bob had chosen m = 0 just by happenstance. The random quantity m is used nowhere without multiplication with either f0 or f1, hence K is actually always the same, as is the ciphertext, thanks to derandomization.
>
> Furthermore the decapsulation and CCA re-encryption both succeed regardless of the secret key, as noted previously for the CPA version.

From:	pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent:	Saturday, February 8, 2020 2:24 PM
To:	Rafael Misoczki
Cc:	pqc-forum
Subject:	Re: [pqc-forum] Re: ROUND 2 OFFICIAL COMMENT: BIKE

On Sat, Feb 8, 2020 at 5:19 PM Rafael Misoczki <rafa.misoczki@gmail.com> wrote:

> Dear Dr. Sarineen,
>
> We appreciate your continuous interest on BIKE.
>
> The version 3.2 of our spec implements a single change to our submission: replacement of the ParallelHash technique by SHA2, as indicated in our website: https://bikesuite.org/spec.html For more details, please see the message sent to the NIST Forum in the corresponding thread.

Thanks!

> Regarding the comments about injecting a zero-value public key, this is trivially addressed by ensuring that public keys are authenticated.

Unfortunately, that is not so trivial in all use cases. Classical e-mail encryption applications assume that the mere ability to correctly decrypt a message is an implicit proof that it has been sent using the correct public key. For example, an S/MIME or PGP signature is usually of the message, not of the recipient's public key. The same is true for some messaging protocols.

In BIKE-1 CPA and CCA one can forge completely valid ciphertext messages (for chosen plaintext) even without knowledge of the *public* key, which I find quite wild -- I'm not even sure what is the correct taxonomy for this property. All secret keys can be used to decapsulate any message addressed to a key in that subspace. The decryption/decapsulation process flags nothing unusual about such a message.

What would be trivial is to include the public key in the shared secret hash computation. It is already part of the secret key in the CCA versions, but of course re-encryption step fails to catch this particular problem due to issues discussed earlier.

> Regarding BIKE-2 public key, the first element is a constant (1) which can be implicitly assumed (and omitted) in a real-world application. We left its explicit representation in the KAT's and associated implementations for the sake of simplicity, thus unifying most of the API's in the functions called by all BIKE-1/2/3.

Thanks. I was worried that there was a "compressing BIKE-2 public keys" patent lurking somewhere.

> Best regards,
> Rafael

Cheers,
- markku