From: Peter Schwabe <peter@cryptojedi.org> via pqc-forum@list.nist.gov
To: pqc-forum <pqc-forum@list.nist.gov>
CC: authors@pq-crystals.org
Subject: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Saturday, December 03, 2022 08:04:13 PM ET

Dear all,

This e-mail is a follow-up to our presentation at the 4th NIST PQC standarization workshop and one of two mails meant to continue and eventually conclude discussions towards the standardization of Kyber. Several questions, possible tweaks, and ideas have been proposed by members of the team, by researchers and future users from the community, and by NIST. The discussion about the standardization of Kyber-512 has already been covered in the mail by Dustin from November 30. The remaining discussions fall roughly into two categories, hence two e-mail threads. Part 1 (this e-mail) is about the choice of symmetric primitives in Kyber.

As a reminder, round-3 Kyber uses multiple algorithms from the Keccak family (FIPS202). Domain separation is achieved partially by using different functions (SHAKE-128, SHAKE-256, SHA3-256, and SHA3-512) and partially by input length. Performance of software implementations of Kyber is currently bottlenecked by Keccak permutations; in order to showcase the possible performance of Kyber with hardware support for symmetric primitives, we also described a "90s" variant based on AES and SHA2.

We have received several questions along the lines of "What about Kyber with X instead of Keccak?" (typically with X taking values from the 90s variant or possibly allowing users of Kyber to choose any symmetric crypto they fancy). The team feels that having multiple incompatible versions of Kyber is not desirable and the obvious choice is to stick to Keccak as the sole underlying symmetric primitive. However, we continue to be interested in hearing opinions and feedback about this. Also, even when fixing Keccak as underlying symmetric primitives, there are still two open questions:

1.) Should Kyber continue to use different functions from the FIPS202 standard and rely on the internal domain separation of those functions or use just cSHAKE or KMAC from NIST SP-800-185 with explicit domain separation? The advantage of such a change is that fewer Keccak-based functions would be used and that analysis of domain separation would be easier. The disadvantage is that one either needs additional Keccak permutations to process domain separation or needs to store pre-computed Keccak states (after absorbing domain separation), one per domain-separated function.

2.) Should the generation of the public matrix A use a 12-round version of Keccak ("TurboSHAKE") instead of the standard 24-round version. This was proposed by the Keccak team and speeds up one of the most costly subroutines of Kyber by a factor of 2. All properties one would expect from a hash function are achieved by Keccak with 12 rounds (by a comfortable margin!). Also, the requirements on the output of this function are rather weak; informally it should look uniformly random and not interact in weird ways with the lattice problems. The main disadvantage of moving to a 12-round version of Keccak is that it requires phrasing the function in terms of lower-level functions of FIPS202 instead of simply using one of the SHA3/SHAKE functions.

We're looking forward to hearing what everybody thinks!

All the best,

The Kyber team

From: John Mattsson <john.mattsson@ericsson.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Peter Schwabe <peter@cryptojedi.org>, pqc-forum <pqc-forum@list.nist.gov>
CC: authors@pq-crystals.org
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Sunday, December 04, 2022 08:18:54 AM ET

Hi,

1) Using cSHAKE/KMAC seems nice, but I don't know the details of how it affects implementation complexity/performance. I trust NIST/the authors to make right choices.

2) Yes. I think it is a good idea to use a 12-round version of Keccak ("TurboSHAKE"). As stated, TurboSHAKE seems to be a cryptographically secure hash function with a good margin. I don't see any problems using it for the Kyber XOF which have lower requirements. I don't think phrasing Kyber in terms of Keccak is a disadvantage. I think is it essential that implementations support lower-level functions like Keccac-p and the AES round function. FIPS 202 clearly states that other KECCAK-p permutations may be specified in the future. Any implementation not adhering to this have themself to blame. Kyber will likely be with us for many decades, optimize for the future, not HW already on the market.

Cheers,

John

From: pqc-forum@list.nist.gov <pqc-forum@list.nist.gov> on behalf of Peter Schwabe <peter@cryptojedi.org>
Date: Sunday, 4 December 2022 at 02:04
To: pqc-forum <pqc-forum@list.nist.gov>
Cc: authors@pq-crystals.org <authors@pq-crystals.org>
Subject: [pqc-forum] Kyber decisions, part 1: Symmetric crypto

Dear all,

This e-mail is a follow-up to our presentation at the 4th NIST PQC standarization workshop and one of two mails meant to continue and eventually conclude discussions towards the standardization of Kyber. Several questions, possible tweaks, and ideas have been proposed by members of the team, by researchers and future users from the community,

and by NIST. The discussion about the standardization of Kyber-512 has already been covered in the mail by Dustin from November 30. The remaining discussions fall roughly into two categories, hence two e-mail threads. Part 1 (this e-mail) is about the choice of symmetric primitives in Kyber.

As a reminder, round-3 Kyber uses multiple algorithms from the Keccak family (FIPS202). Domain separation is achieved partially by using different functions (SHAKE-128, SHAKE-256, SHA3-256, and SHA3-512) and partially by input length. Performance of software implementations of Kyber is currently bottlenecked by Keccak permutations; in order to showcase the possible performance of Kyber with hardware support for symmetric primitives, we also described a "90s" variant based on AES and SHA2.

We have received several questions along the lines of "What about Kyber with X instead of Keccak?" (typically with X taking values from the 90s variant or possibly allowing users of Kyber to choose any symmetric crypto they fancy). The team feels that having multiple incompatible versions of Kyber is not desirable and the obvious choice is to stick to Keccak as the sole underlying symmetric primitive. However, we continue to be interested in hearing opinions and feedback about this. Also, even when fixing Keccak as underlying symmetric primitives, there are still two open questions:

1.) Should Kyber continue to use different functions from the FIPS202 standard and rely on the internal domain separation of those functions or use just cSHAKE or KMAC from NIST SP-800-185 with explicit domain separation? The advantage of such a change is that fewer Keccak-based functions would be used and that analysis of domain separation would be easier. The disadvantage is that one either needs additional Keccak permutations to process domain separation or needs to store pre-computed Keccak states (after absorbing domain separation), one per domain-separated function.

2.) Should the generation of the public matrix A use a 12-round version of Keccak ("TurboSHAKE") instead of the standard 24-round version.

This was proposed by the Keccak team and speeds up one of the most costly subroutines of Kyber by a factor of 2. All properties one would expect from a hash function are achieved by Keccak with 12 rounds (by a comfortable margin!). Also, the requirements on the output of this function are rather weak; informally it should look uniformly random and not interact in weird ways with the lattice problems. The main disadvantage of moving to a 12-round version of Keccak is that it requires phrasing the function in terms of lower-level functions of FIPS202 instead of simply using one of the SHA3/SHAKE functions.

We're looking forward to hearing what everybody thinks!

All the best,

The Kyber team

--

From: Blumenthal, Uri - 0553 - MITLL <uri@ll.mit.edu> via pqc-forum@list.nist.gov
To: John Mattsson <john.mattsson@ericsson.com>, Peter Schwabe <peter@cryptojedi.org>, pqc-forum <pqc-forum@list.nist.gov>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Sunday, December 04, 2022 09:46:57 AM ET
Attachments: smime.p7m

Both of these proposals make sense. However, I'd like to underscore the importance of:

1. Kyber uses one PRF, not a bunch (perhaps, with one exception for matrix expansion); and
2. Whatever PRF it uses – is a NIST standard.

I'd love to see TurboSHAKE there – assuming NIST makes it a standard.

Thanks!

P.S. If that NIST-blessed PRF turns out to be SHA2-based, not SHA-3, I'll welcome that, as it seems that the world-wide PKI is stuck with SHA-2 for foreseeable future – thus all the PK-related crypto will have to support it no matter what.

P.P.S. If the world decides that one standard secure hash family is enough, and deprecates either one of the two (I don't particularly care which one), it would be great. Of course, I doubt I'd see it, but one is allowed to hope… ;-)

--

V/R,

Uri

There are two ways to design a system. One is to make it so simple there are obviously no deficiencies.

The other is to make it so complex there are no obvious deficiencies.

- C. A. R. Hoare

> **From:** 'John Mattsson' via pqc-forum
> **Reply-To:** John Mattsson
> **Date:** Sunday, December 4, 2022 at 08:18
> **To:** Peter Schwabe , pqc-forum

Cc: "authors@pq-crystals.org"
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto

Hi,

1) Using cSHAKE/KMAC seems nice, but I don't know the details of how it affects implementation complexity/performance. I trust NIST/the authors to make right choices.

2) Yes. I think it is a good idea to use a 12-round version of Keccak ("TurboSHAKE") . As stated, TurboSHAKE seems to be a cryptographically secure hash function with a good margin. I don't see any problems using it for the Kyber XOF which have lower requirements. I don't think phrasing Kyber in terms of Keccak is a disadvantage. I think is it essential that implementations support lower-level functions like Keccac-p and the AES round function. FIPS 202 clearly states that other KECCAK-p permutations may be specified in the future. Any implementation not adhering to this have themself to blame. Kyber will likely be with us for many decades, optimize for the future, not HW already on the market.

Cheers,

John

From: pqc-forum@list.nist.gov on behalf of Peter Schwabe
Date: Sunday, 4 December 2022 at 02:04
To: pqc-forum
Cc: authors@pq-crystals.org
Subject: [pqc-forum] Kyber decisions, part 1: Symmetric crypto

Dear all,

This e-mail is a follow-up to our presentation at the 4th NIST PQC standarization workshop and one of two mails meant to continue and eventually conclude discussions towards the standardization of Kyber. Several questions, possible tweaks, and ideas have been proposed by members of the team, by researchers and future users from the community, and by NIST. The discussion about the standardization of Kyber-512 has already been covered in the mail by Dustin from November 30. The remaining discussions fall roughly into two categories, hence two e-mail threads. Part 1 (this e-mail) is about the choice of symmetric primitives in Kyber.

As a reminder, round-3 Kyber uses multiple algorithms from the Keccak family (FIPS202). Domain separation is achieved partially by using different functions (SHAKE-128, SHAKE-256, SHA3-256, and SHA3-512) and partially by input length. Performance of software implementations of Kyber is currently bottlenecked by Keccak permutations; in order to showcase the possible performance of Kyber with hardware support for symmetric primitives, we also described a "90s" variant based on AES and SHA2.

We have received several questions along the lines of "What about Kyber with X instead of Keccak?" (typically with X taking values from the 90s variant or possibly allowing users of Kyber to choose any symmetric crypto they fancy). The team feels that having multiple incompatible versions of Kyber is not desirable and the obvious choice is to stick to Keccak as the sole underlying symmetric primitive. However, we continue to be interested in hearing opinions and feedback about this. Also, even when fixing Keccak as underlying symmetric primitives, there are still two open questions:

1.) Should Kyber continue to use different functions from the FIPS202 standard and rely on the internal domain separation of those functions or use just cSHAKE or KMAC from NIST SP-800-185 with explicit domain separation? The advantage of such a change is that fewer Keccak-based functions would be used and that analysis of domain separation would be easier. The disadvantage is that one either needs additional Keccak permutations to process domain separation or needs to store pre-computed Keccak states (after absorbing domain separation), one per domain-separated function.

2.) Should the generation of the public matrix A use a 12-round version of Keccak ("TurboSHAKE") instead of the standard 24-round version. This was proposed by the Keccak team and speeds up one of the most costly subroutines of Kyber by a factor of 2. All properties one would expect from a hash function are achieved by Keccak with 12 rounds (by a comfortable margin!). Also, the requirements on the output of this function are rather weak; informally it should look

uniformly random and not interact in weird ways with the lattice problems. The main disadvantage of moving to a 12-round version of Keccak is that it requires phrasing the function in terms of lower-level functions of FIPS202 instead of simply using one of the SHA3/SHAKE functions.

We're looking forward to hearing what everybody thinks!

All the best,

The Kyber team

forum+unsubscribe@list.nist.gov.
To view this discussion on the web visit https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/E0B13AA4-C547-4969-A8C5-0220FF948E9B%40ll.mit.edu.

From: Simon Hoerder <simon@hoerder.net> via pqc-forum@list.nist.gov
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Sunday, December 04, 2022 01:14:01 PM ET

Hi,

regarding TurboSHAKE I see a messaging problem:

* Allowing TurboSHAKE for PQC but not for hashing is inconsistent. Message hashing tends to process far more data than is needed for PQC and is supposed to use the slow method?
* Anyone who deployed SHA-3 / SHAKE so far was counting that as "PQC preparedness". In SW / SW+ISE implementations it shouldn't be a big issue to change to round reduced variants but in HW co-processors it is not a common thing to have a configurable number of rounds. Do you want to tell those people that they can't use their HW accelerators for PQC even if there are no security issues? (Not sure how many there are but I very much suspect the number is > 0 in infrastructure markets.)

The solution would be to update SHA-3 / SHAKE with additional round-reduced parameter sets and to specify additional PQC parameter sets that use SHA-3 / SHAKE as it is now. For Kyber that could be Kyber-512, Kyber-768 and Kyber-1024 with round reduced SHAKE and at least Kyber-1024-c with classic SHA-3/SHAKE.

I would also like to point out that all HW vendors aiming for CNSA 2.0 transition timelines will be working at least on SHA-3/SHAKE today. Getting clarity regarding (Turbo- and C-)SHAKE versions asap is a priority in this context.

Finally, this is not just an issue for Kyber, it is an issue for all PQC algorithms that rely on SHA-3/SHAKE.

Best,
Simon
(speaking only for myself)

Op 04/12/2022 om 02:03 schreef Peter Schwabe:
> Dear all,
>
> This e-mail is a follow-up to our presentation at the 4th NIST PQC
> standarization workshop and one of two mails meant to continue and
> eventually conclude discussions towards the standardization of Kyber.
> Several questions, possible tweaks, and ideas have been proposed by
> members of the team, by researchers and future users from the community,
> and by NIST. The discussion about the standardization of Kyber-512 has
> already been covered in the mail by Dustin from November 30. The
> remaining discussions fall roughly into two categories, hence two e-mail
> threads. Part 1 (this e-mail) is about the choice of symmetric
> primitives in Kyber.
>
> As a reminder, round-3 Kyber uses multiple algorithms from the Keccak
> family (FIPS202). Domain separation is achieved partially by using
> different functions (SHAKE-128, SHAKE-256, SHA3-256, and SHA3-512) and
> partially by input length. Performance of software implementations of
> Kyber is currently bottlenecked by Keccak permutations; in order to
> showcase the possible performance of Kyber with hardware support for
> symmetric primitives, we also described a "90s" variant based on AES and
> SHA2.
>
> We have received several questions along the lines of "What about Kyber
> with X instead of Keccak?" (typically with X taking values from the 90s
> variant or possibly allowing users of Kyber to choose any symmetric
> crypto they fancy). The team feels that having multiple incompatible
> versions of Kyber is not desirable and the obvious choice is to stick to
> Keccak as the sole underlying symmetric primitive. However, we continue
> to be interested in hearing opinions and feedback about this. Also, even
> when fixing Keccak as underlying symmetric primitives, there are still
> two open questions:
>
> 1.) Should Kyber continue to use different functions from the FIPS202
>   standard and rely on the internal domain separation of those
>   functions or use just cSHAKE or KMAC from NIST SP-800-185 with

> explicit domain separation? The advantage of such a change is that
> fewer Keccak-based functions would be used and that analysis of
> domain separation would be easier. The disadvantage is that one
> either needs additional Keccak permutations to process domain
> separation or needs to store pre-computed Keccak states (after
> absorbing domain separation), one per domain-separated function.
>
> 2.) Should the generation of the public matrix A use a 12-round version
> of Keccak ("TurboSHAKE") instead of the standard 24-round version.
> This was proposed by the Keccak team and speeds up one of the most
> costly subroutines of Kyber by a factor of 2. All properties one
> would expect from a hash function are achieved by Keccak with 12
> rounds (by a comfortable margin!). Also, the requirements on the
> output of this function are rather weak; informally it should look
> uniformly random and not interact in weird ways with the lattice
> problems. The main disadvantage of moving to a 12-round version of
> Keccak is that it requires phrasing the function in terms of
> lower-level functions of FIPS202 instead of simply using one of the
> SHA3/SHAKE functions.
>
> We're looking forward to hearing what everybody thinks!
>
> All the best,
>
> The Kyber team
>

From: Peter Schwabe <peter@cryptojedi.org> via pqc-forum@list.nist.gov
To: Blumenthal, Uri - 0553 - MITLL <uri@ll.mit.edu>
CC: John Mattsson <john.mattsson@ericsson.com>, Peter Schwabe <peter@cryptojedi.org>, pqc-forum <pqc-forum@list.nist.gov>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Sunday, December 04, 2022 08:39:18 PM ET

"Blumenthal, Uri - 0553 - MITLL" <uri@ll.mit.edu> wrote:

Dear Uri, dear all,

> Both of these proposals make sense. However, I'd like to underscore
> the importance of:
>
>
> Kyber uses one PRF, not a bunch (perhaps, with one exception for
> matrix expansion); and Whatever PRF it uses – is a NIST standard.

This may be nitpicking, but I just want to clarify so that everybody is on the same page: Just one PRF is not sufficient as symmetric building block for Kyber; we *do* need hash functions and we do need a XOF.

I would like to understand the reasons for preferring the cSHAKE solution. At least for software implementations, this will not significantly reduce code complexity. For any of the proposed solutions, the costly core routine is the Keccak permutation. For hardware implementations it depends on what exactly is implemented in hardware and what the interfaces look like.

All the best,

Peter

From: Markku-Juhani O. Saarinen <mjos.crypto@gmail.com> via pqc-forum@list.nist.gov
To: pqc-forum <pqc-forum@list.nist.gov>
CC: Peter Schwabe <peter@cryptojedi.org>, John Mattsson <john.mattsson@ericsson.com>, pqc-forum <pqc-forum@list.nist.gov>, u...@ll.mit.edu <uri@ll.mit.edu>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Sunday, December 04, 2022 10:04:45 PM ET

Hi All,

I'd think that as long as the hardware module can handle various rate/capacity values, differences between SHA3-256/512, SHAKE128, and SHAKE256 are indeed trivial (effectively the contents of one padding byte.) As Peter noted, it is purely a matter of domain separation.

- Side-channel secure implementations complicate this a bit. Using the function names of the Kyber spec, the "G", "PRF", and "KDF" instances need a masked Keccak, while the "XOF" or "H" doesn't need to be (if I recall correctly!). If there is no direct hardware support for it, masked Keccak permutations are much more expensive than regular ones. So having the minimum required "rate" parameter to the security level, there is preferable (assuming that the input is long enough to affect the number of permutations; for short inputs, it makes no difference.)

- I would prefer domain separation based on input encoding rather than using cSHAKE: Most hardware architects probably agree that zeroing the state at initialization is preferable to having a (muxed) 1600-bit holding register -- as would be required to skip the first permutation in cSHAKE. cSHAKE may also force additional masking refreshes on "secret hashes" if some particularly unfortunate mode selection is made.

- To me, a 12-round Keccak, especially as an XOF for the expansion of A matrix (where security requirements are low -- the input and output are always public), makes total sense. Dilithium would equivalently benefit from this. TurboSHAKE is very natural as long as the round constants are selected as they have "always" been selected (e.g. in KangarooTwelve - "K12.")

- The TurboSHAKE talk discussed features that go a little bit beyond K12; I interpreted "multi-string" as input domain separation in the style of TupleHash (which is in SP 800-185). Additionally, any design decision that increases the potential for parallelism of permutations would be preferable. The theory of permutation-based cryptography facilitates a bit more of this than is currently used in NIST standard Keccak modes (yep, I know that the "A" extension XOFs in Kyber and Dilithium are already running in a de facto counter mode.)

Cheers,

- markku

Dr. Markku-Juhani O. Saarinen <mjos@iki.fi>

On Monday, December 5, 2022 at 2:39:14 AM UTC+1 Peter Schwabe wrote:

> "Blumenthal, Uri - 0553 - MITLL" <u...@ll.mit.edu> wrote:
>
> Dear Uri, dear all,
>
> > Both of these proposals make sense. However, I'd like to underscore
> > the importance of:
> >
> >
> > Kyber uses one PRF, not a bunch (perhaps, with one exception for
> > matrix expansion); and Whatever PRF it uses – is a NIST standard.
>
> This may be nitpicking, but I just want to clarify so that everybody is
> on the same page: Just one PRF is not sufficient as symmetric building
> block for Kyber; we *do* need hash functions and we do need a XOF.
>
> I would like to understand the reasons for preferring the cSHAKE
> solution. At least for software implementations, this will not
> significantly reduce code complexity. For any of the proposed solutions,
> the costly core routine is the Keccak permutation. For hardware
> implementations it depends on what exactly is implemented in hardware
> and what the interfaces look like.
>
> All the best,
>
> Peter

--

forum+unsubscribe@list.nist.gov.
To view this discussion on the web visit https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/a14b5849-c402-4af8-94c1-7ed69b18335cn%40list.nist.gov.

From: Mike Hamburg <mike@shiftleft.org> via pqc-forum@list.nist.gov
To: pqc-forum <pqc-forum@list.nist.gov>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Monday, December 05, 2022 08:00:07 AM ET

[Attempting to re-send, since the list server is flaking out. Apologies if you received this more than once.]

Hi all,
I mostly agree with Markku here.

I think it's better to domain separate based on input encoding instead of based on cSHAKE. It's not as broadly supported as SHAKE, and if you have several separate instances then you either need an extra permutation call or else several precomputed 200-byte states, which is significant in hardware or perhaps in embedded software. For ThreeBears I used cSHAKE(parameters || purpose || data) with a fixed personalization string "ThreeBears", but raw SHAKE would have been about as good — using cSHAKE and personalization was probably overkill. Here the `parameters` are different between different KEM instances (in your case eg Kyber512 vs Kyber768 vs Kyber1024). The goal is that if someone accidentally reuses a seed string for different key sizes, they won't get closely related keys. The `purpose` is what you're using it for (hash, keygen, generate A, encrypt, PRF etc). The whole (parameters || purpose) block has a fixed 8-byte aligned size, in case clients are optimized for that. You could also put the counter in the purpose block, but I put it at the end to simplify use of parallel hashing APIs.

I agree that it would be cryptographically fine to generate A using TurboSHAKE, and whether to do this is mostly a matter of support (in hardware, ISEs, parallel or serial libraries etc) vs speed. The counterpoint is that Kyber is basically fast enough already, and that introducing a different function causes more disruption than it's worth even pre-standardization. I would not want to go beyond the existing mode (sponge construction in counter mode) to avoid causing problems for hardware. In particular I would not want to use a Farfalle construction. But since expanding A is called on fixed-size objects, it's possible that it would not go beyond sponge mode even if TurboSHAKE can do this in general.

To help weigh this, do you have a profile for what fraction of Kyber's (or Dilithium's) runtime is specifically the generation of A?

Regards,

— Mike

On Dec 5, 2022, at 4:04 AM, Markku-Juhani O. Saarinen <mjos.crypto@gmail.com> wrote:

Hi All,
I'd think that as long as the hardware module can handle various rate/capacity values, differences between SHA3-256/512, SHAKE128, and SHAKE256 are indeed trivial (effectively the contents of one padding byte.) As Peter noted, it is purely a matter of domain separation.

- Side-channel secure implementations complicate this a bit. Using the function names of the Kyber spec, the "G", "PRF", and "KDF" instances need a masked Keccak, while the "XOF" or "H" doesn't need to be (if I recall correctly!). If there is no direct hardware support for it, masked Keccak permutations are much more expensive than regular ones. So having the minimum required "rate" parameter to the security level, there is preferable (assuming that the input is long enough to affect the number of permutations; for short inputs, it makes no difference.)

- I would prefer domain separation based on input encoding rather than using cSHAKE: Most hardware architects probably agree that zeroing the state at initialization is preferable to having a (muxed) 1600-bit holding register -- as would be required to skip the first permutation in cSHAKE. cSHAKE may also force additional masking refreshes on "secret hashes" if some particularly unfortunate mode selection is made.

- To me, a 12-round Keccak, especially as an XOF for the expansion of A matrix (where security requirements are low -- the input and output are always public), makes total sense. Dilithium would equivalently benefit from this. TurboSHAKE is very natural as long as the round constants are selected as they have "always" been selected (e.g. in KangarooTwelve - "K12.")

- The TurboSHAKE talk discussed features that go a little bit beyond K12; I interpreted "multi-string" as input domain separation in the style of TupleHash (which is in SP 800-185). Additionally, any design decision that increases the potential for parallelism of permutations would be preferable. The theory of permutation-based cryptography facilitates a bit more of this than is currently used

in NIST standard Keccak modes (yep, I know that the "A" extension XOFs in Kyber and Dilithium are already running in a de facto counter mode.)

Cheers,

- markku

Dr. Markku-Juhani O. Saarinen <mjos@iki.fi>

On Monday, December 5, 2022 at 2:39:14 AM UTC+1 Peter Schwabe wrote:

> "Blumenthal, Uri - 0553 - MITLL" <u...@ll.mit.edu> wrote:
>
> Dear Uri, dear all,
>
> > Both of these proposals make sense. However, I'd like to underscore
> > the importance of:
> >
> >
> > Kyber uses one PRF, not a bunch (perhaps, with one exception for
> > matrix expansion); and Whatever PRF it uses – is a NIST standard.
>
> This may be nitpicking, but I just want to clarify so that everybody is
> on the same page: Just one PRF is not sufficient as symmetric building
> block for Kyber; we *do* need hash functions and we do need a XOF.
>
> I would like to understand the reasons for preferring the cSHAKE
> solution. At least for software implementations, this will not
> significantly reduce code complexity. For any of the proposed solutions,
> the costly core routine is the Keccak permutation. For hardware
> implementations it depends on what exactly is implemented in hardware
> and what the interfaces look like.
>
> All the best,
>
> Peter

From: peter...@gmail.com <peter.pessl@gmail.com> via pqc-forum@list.nist.gov
To: pqc-forum <pqc-forum@list.nist.gov>
CC: mi...@shiftleft.org <mike@shiftleft.org>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Monday, December 05, 2022 12:57:08 PM ET

Hi all,
I just want to add that the runtime benefit of TurboSHAKE is a likely lot higher for Dilithium (compared to Kyber), especially for memory-constrained devices. While the larger matrix dimensions are partly to blame, the more important difference is that storing A in full (between 12 and 42kB) is simply not an option on many devices (neither in SRAM nor in NVM). This means that A has to be re-generated in each iteration of the rejection loop, adding a huge performance penalty.

The impact of such a re-generation was analyzed in https://ia.cr/2020/1278. The numbers in the paper maybe can't be used to derive the exact overhead of re-generating A (Paper uses 2nd-round Dilithium, and the implementation that re-generates A also computes y twice to save even more memory), but they do show the general direction: the reported runtime overhead is over 3x. A switch to TurboSHAKE would be highly beneficial there, even more in absolute than in relative terms.

BR Peter

mi...@shiftleft.org schrieb am Montag, 5. Dezember 2022 um 14:00:02 UTC+1:

> [Attempting to re-send, since the list server is flaking out. Apologies if you received this more than once.]
>
> Hi all,
> I mostly agree with Markku here.
>
> I think it’s better to domain separate based on input encoding instead of based on cSHAKE. It’s not as broadly supported as SHAKE, and if you have several separate instances then you either need an extra permutation call or else several precomputed 200-byte states, which is significant in hardware or perhaps in embedded software. For ThreeBears I used cSHAKE(parameters || purpose || data) with a fixed personalization string “ThreeBears”, but raw SHAKE would have been about as good — using cSHAKE and personalization was probably overkill. Here the `parameters` are different between different KEM instances (in

your case eg Kyber512 vs Kyber768 vs Kyber1024). The goal is that if someone accidentally reuses a seed string for different key sizes, they won't get closely related keys. The `purpose` is what you're using it for (hash, keygen, generate A, encrypt, PRF etc). The whole (parameters || purpose) block has a fixed 8-byte aligned size, in case clients are optimized for that. You could also put the counter in the purpose block, but I put it at the end to simplify use of parallel hashing APIs.

I agree that it would be cryptographically fine to generate A using TurboSHAKE, and whether to do this is mostly a matter of support (in hardware, ISEs, parallel or serial libraries etc) vs speed. The counterpoint is that Kyber is basically fast enough already, and that introducing a different function causes more disruption than it's worth even pre-standardization. I would not want to go beyond the existing mode (sponge construction in counter mode) to avoid causing problems for hardware. In particular I would not want to use a Farfalle construction. But since expanding A is called on fixed-size objects, it's possible that it would not go beyond sponge mode even if TurboSHAKE can do this in general.

To help weigh this, do you have a profile for what fraction of Kyber's (or Dilithium's) runtime is specifically the generation of A?

Regards,

— Mike

> On Dec 5, 2022, at 4:04 AM, Markku-Juhani O. Saarinen <mjos....@gmail.com> wrote:
>
> Hi All,
> I'd think that as long as the hardware module can handle various rate/capacity values, differences between SHA3-256/512, SHAKE128, and SHAKE256 are indeed trivial (effectively the contents of one padding byte.) As Peter noted, it is purely a matter of domain separation.
>
> - Side-channel secure implementations complicate this a bit. Using the function names of the Kyber spec, the "G", "PRF", and "KDF" instances need a masked Keccak, while the "XOF" or "H" doesn't need to be (if I recall correctly!). If there is no direct hardware support for it, masked Keccak permutations are much more expensive than regular ones. So having the minimum required "rate" parameter

to the security level, there is preferable (assuming that the input is long enough to affect the number of permutations; for short inputs, it makes no difference.)

- I would prefer domain separation based on input encoding rather than using cSHAKE: Most hardware architects probably agree that zeroing the state at initialization is preferable to having a (muxed) 1600-bit holding register -- as would be required to skip the first permutation in cSHAKE. cSHAKE may also force additional masking refreshes on "secret hashes" if some particularly unfortunate mode selection is made.

- To me, a 12-round Keccak, especially as an XOF for the expansion of A matrix (where security requirements are low -- the input and output are always public), makes total sense. Dilithium would equivalently benefit from this. TurboSHAKE is very natural as long as the round constants are selected as they have "always" been selected (e.g. in KangarooTwelve - "K12.")

- The TurboSHAKE talk discussed features that go a little bit beyond K12; I interpreted "multi-string" as input domain separation in the style of TupleHash (which is in SP 800-185). Additionally, any design decision that increases the potential for parallelism of permutations would be preferable. The theory of permutation-based cryptography facilitates a bit more of this than is currently used in NIST standard Keccak modes (yep, I know that the "A" extension XOFs in Kyber and Dilithium are already running in a de facto counter mode.)

Cheers,

- markku

Dr. Markku-Juhani O. Saarinen <mj...@iki.fi>

On Monday, December 5, 2022 at 2:39:14 AM UTC+1 Peter Schwabe wrote:

> "Blumenthal, Uri - 0553 - MITLL" <u...@ll.mit.edu> wrote:
>
> Dear Uri, dear all,
>
> > Both of these proposals make sense. However, I'd like to underscore

> the importance of:
>
>
> Kyber uses one PRF, not a bunch (perhaps, with one exception for
> matrix expansion); and Whatever PRF it uses – is a NIST standard.

This may be nitpicking, but I just want to clarify so that everybody is on the same page: Just one PRF is not sufficient as symmetric building block for Kyber; we *do* need hash functions and we do need a XOF.

I would like to understand the reasons for preferring the cSHAKE solution. At least for software implementations, this will not significantly reduce code complexity. For any of the proposed solutions, the costly core routine is the Keccak permutation. For hardware implementations it depends on what exactly is implemented in hardware and what the interfaces look like.

All the best,

Peter

From: Peter Schwabe <peter@cryptojedi.org> via pqc-forum@list.nist.gov
To: Mike Hamburg <mike@shiftleft.org>
CC: Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>, pqc-forum <pqc-forum@list.nist.gov>, Peter Schwabe <peter@cryptojedi.org>, John Mattsson <john.mattsson@ericsson.com>, u...@ll.mit.edu <uri@ll.mit.edu>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Friday, December 09, 2022 07:15:20 AM ET

Mike Hamburg <mike@shiftleft.org> wrote:

> Hi all,

Hi Mike, hi all,

> I mostly agree with Markku here.
>
> I think it's better to domain separate based on input encoding instead
> of based on cSHAKE. It's not as broadly supported as SHAKE, and if
> you have several separate instances then you either need an extra
> permutation call or else several precomputed 200-byte states, which is
> significant in hardware or perhaps in embedded software. For
> ThreeBears I used cSHAKE(parameters || purpose || data) with a fixed
> personalization string "ThreeBears", but raw SHAKE would have been
> about as good — using cSHAKE and personalization was probably
> overkill. Here the `parameters` are different between different KEM
> instances (in your case eg Kyber512 vs Kyber768 vs Kyber1024). The
> goal is that if someone accidentally reuses a seed string for
> different key sizes, they won't get closely related keys. The
> `purpose` is what you're using it for (hash, keygen, generate A,
> encrypt, PRF etc). The whole (parameters || purpose) block has a
> fixed 8-byte aligned size, in case clients are optimized for that.
> You could also put the counter in the purpose block, but I put it at
> the end to simplify use of parallel hashing APIs.
>
> I agree that it would be cryptographically fine to generate A using
> TurboSHAKE, and whether to do this is mostly a matter of support (in
> hardware, ISEs, parallel or serial libraries etc) vs speed. The
> counterpoint is that Kyber is basically fast enough already, and that
> introducing a different function causes more disruption than it's

> worth even pre-standardization. I would not want to go beyond the
> existing mode (sponge construction in counter mode) to avoid causing
> problems for hardware. In particular I would not want to use a
> Farfalle construction. But since expanding A is called on fixed-size
> objects, it’s possible that it would not go beyond sponge mode even if
> TurboSHAKE can do this in general.
>
> To help weigh this, do you have a profile for what fraction of Kyber’s
> (or Dilithium’s) runtime is specifically the generation of A?

That's highly platform dependent. I just ran some benchmarks on a
Haswell machine for the AVX2-optimized implementation. What I'm getting
is:

═══ KYBER 512 ═══

keygen: 25120
encaps: 39180
decaps: 30868
gen_A: 7472
xof: 6088

═══ KYBER 768 ═══

keygen: 43596
encaps: 59464
decaps: 47684
gen_A: 20320
xof: 16865

═══ KYBER 1024 ═══

keygen: 60396
encaps: 83576
decaps: 67656

```
gen_A:  29664
xof:    24109
```

Here, "gen_A" is the full cycles for generation of the matrix A, including the cycles for rejection sampling; "xof" is the cycles used on SHAKE-128, i.e., the cycles that would pretty exactly halve when moving to TurboSHAKE.

Note that in this implementation, the generation of A is using a fast vectorized implementation of Keccak (vectorizing across different matrix entries), while many other calls to Keccak are sequential and thus much slower. On embedded platforms like the M4 I would expect the relative cost for "xof" to be considerably higher, simply because there is no speedup from vectorization. I can run those benchmarks when I'm back from my travels.

On the other hand, in a masked implementation I would expect the relative cost of "xof" to be much lower, simply because we don't require masking in the generation of A, but we do for most other Keccak calls.

With HW acceleration of Keccak it's somewhat hard to predict and it will depend a lot on how large the acceleration is: on the one hand, all Keccak calls will likely have the same cost (so matrix generation will be relatively more costly compared to other Keccak calls than in the AVX2 implementation), but on the other hand, one would expect polynomial arithmetic to use a more significant portion of the cycles, so overall the cost for Keccak permutations might not matter all that much.

Does this help?

All the best,

Peter

From: Mike Hamburg <mike@shiftleft.org> via pqc-forum@list.nist.gov
To: Peter Schwabe <peter@cryptojedi.org>
CC: Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>, pqc-forum <pqc-forum@list.nist.gov>, John Mattsson <john.mattsson@ericsson.com>, u...@ll.mit.edu <uri@ll.mit.edu>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Friday, December 09, 2022 09:53:38 AM ET

On Dec 9, 2022, at 1:14 PM, Peter Schwabe <peter@cryptojedi.org> wrote:

Mike Hamburg <mike@shiftleft.org> wrote:

> Hi all,

Hi Mike, hi all,

> I mostly agree with Markku here.
>
> I think it's better to domain separate based on input encoding instead of based on cSHAKE. It's not as broadly supported as SHAKE, and if you have several separate instances then you either need an extra permutation call or else several precomputed 200-byte states, which is significant in hardware or perhaps in embedded software. For ThreeBears I used cSHAKE(parameters || purpose || data) with a fixed personalization string "ThreeBears", but raw SHAKE would have been about as good — using cSHAKE and personalization was probably overkill. Here the `parameters` are different between different KEM instances (in your case eg Kyber512 vs Kyber768 vs Kyber1024). The goal is that if someone accidentally reuses a seed string for different key sizes, they won't get closely related keys. The `purpose` is what you're using it for (hash, keygen, generate A, encrypt, PRF etc). The whole (parameters || purpose) block has a fixed 8-byte aligned size, in case clients are optimized for that. You could also put the counter in the purpose block, but I put it at the end to simplify use of parallel hashing APIs.
>
> I agree that it would be cryptographically fine to generate A using TurboSHAKE, and whether to do this is mostly a matter of support (in hardware, ISEs, parallel or serial libraries etc) vs speed. The counterpoint is that Kyber is basically fast enough already, and that introducing a different function causes more disruption than it's worth even pre-standardization. I would not want to go beyond the existing mode (sponge construction in counter mode) to avoid causing

> problems for hardware. In particular I would not want to use a Farfalle construction. But since expanding A is called on fixed-size objects, it’s possible that it would not go beyond sponge mode even if TurboSHAKE can do this in general.
>
> To help weigh this, do you have a profile for what fraction of Kyber’s (or Dilithium’s) runtime is specifically the generation of A?

That's highly platform dependent. I just ran some benchmarks on a Haswell machine for the AVX2-optimized implementation. What I'm getting is:

====== KYBER 512 ======

keygen: 25120

encaps: 39180

decaps: 30868

gen_A: 7472

xof: 6088

====== KYBER 768 ======

keygen: 43596

encaps: 59464

decaps: 47684

gen_A: 20320

xof: 16865

====== KYBER 1024 ======

keygen: 60396

encaps: 83576

decaps: 67656

gen_A: 29664

xof: 24109

Here, "gen_A" is the full cycles for generation of the matrix A,
including the cycles for rejection sampling; "xof" is the cycles used on
SHAKE-128, i.e., the cycles that would pretty exactly halve when moving
to TurboSHAKE.

Note that in this implementation, the generation of A is using a fast
vectorized implementation of Keccak (vectorizing across different matrix
entries), while many other calls to Keccak are sequential and thus much
slower. On embedded platforms like the M4 I would expect the relative
cost for "xof" to be considerably higher, simply because there is no
speedup from vectorization. I can run those benchmarks when I'm back
from my travels.

On the other hand, in a masked implementation I would expect the
relative cost of "xof" to be much lower, simply because we don't require
masking in the generation of A, but we do for most other Keccak calls.

With HW acceleration of Keccak it's somewhat hard to predict and it will
depend a lot on how large the acceleration is: on the one hand, all
Keccak calls will likely have the same cost (so matrix generation will
be relatively more costly compared to other Keccak calls than in the
AVX2 implementation), but on the other hand, one would expect polynomial
arithmetic to use a more significant portion of the cycles, so overall
the cost for Keccak permutations might not matter all that much.

Does this help?

All the best,

Peter

Hi Peter, thanks for this data.

It looks like the savings in the cases above, for a full key exchange assuming

that decaps regenerates A, are about 10%, 17% and 17% for the three security levels. That's not nothing, but it's not a huge amount either. Also, in terms of absolute time savings on eg 3 GHz Haswell, even for Kyber 1024 we're looking at about 12μs savings for the entire key exchange. I'm not convinced that's worth the change, even though it's cryptographically fine.

Maybe on the M4 it will be relevant, though again I'm not sure many M4-class devices will be doing key exchanges that a human or time-relevant process will wait for.

I agree that in hardware, switching to TurboSHAKE won't typically save as much because, at least if Keccak is implemented in the most common one-cycle-per-round configuration, it won't be the bottleneck unless the NTT unit is very large. In fact, the savings might be almost zero if the XOF and NTT operate in parallel.

TurboSHAKE will also be annoying for hardware designers, because (speaking as a hardware designer) we will need to alter our accelerators to support it but as far as I can tell it isn't spec'd yet. Since hardware has a long development cycle, this will mean adding a bunch of extra knobs to the Keccak accelerator and hoping we picked the right ones.

Overall, I'm not in favor of this change but I don't think it will be a disaster either. There will be less risk for hardware projects if you can specify what TurboSHAKE will look like, possibly even before deciding whether to use it.

Regards,

— Mike

From: Stott, David - 0553 - MITLL <david.stott@ll.mit.edu> via pqc-forum@list.nist.gov
To: Mike Hamburg <mike@shiftleft.org>, Peter Schwabe <peter@cryptojedi.org>, pqc-forum <pqc-forum@list.nist.gov>
CC: Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>, John Mattsson <john.mattsson@ericsson.com>, Blumenthal, Uri - 0553 - MITLL <uri@ll.mit.edu>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Friday, December 09, 2022 10:47:43 AM ET
Attachments: smime.p7m

Thanks, Mike.

We have looked at a softcore based design with a SHAKE coprocessor (without the NTT acceleration). Our findings support your comments below.

- The benefit of TurboSHAKE would be very small for HW (12 cycles per keccak-permute round), where our current unoptimized coprocessor interface spends more than 100 cycles transferring data
- With HW acceleration of SHA3 (SHAKE or TurboSHAKE) the critical path becomes NTT

The actual changes to the code to go to 12 rounds seem rather minor (granted we are not working on a production-quality implementation).

The software-only version, of course, is a different story.

-david

From: on behalf of Mike Hamburg
Date: Friday, December 9, 2022 at 9:54 AM
To: Peter Schwabe
Cc: "Markku-Juhani O. Saarinen" , pqc-forum , John Mattsson , "Blumenthal, Uri - 0553 - MITLL"
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto

> On Dec 9, 2022, at 1:14 PM, Peter Schwabe wrote:
>
> Mike Hamburg <mike@shiftleft.org> wrote:
>
> > Hi all,

Hi Mike, hi all,

I mostly agree with Markku here.

I think it's better to domain separate based on input encoding instead of based on cSHAKE. It's not as broadly supported as SHAKE, and if you have several separate instances then you either need an extra permutation call or else several precomputed 200-byte states, which is significant in hardware or perhaps in embedded software. For ThreeBears I used cSHAKE(parameters || purpose || data) with a fixed personalization string "ThreeBears", but raw SHAKE would have been about as good — using cSHAKE and personalization was probably overkill. Here the `parameters` are different between different KEM instances (in your case eg Kyber512 vs Kyber768 vs Kyber1024). The goal is that if someone accidentally reuses a seed string for different key sizes, they won't get closely related keys. The `purpose` is what you're using it for (hash, keygen, generate A, encrypt, PRF etc). The whole (parameters || purpose) block has a fixed 8-byte aligned size, in case clients are optimized for that. You could also put the counter in the purpose block, but I put it at the end to simplify use of parallel hashing APIs.

I agree that it would be cryptographically fine to generate A using TurboSHAKE, and whether to do this is mostly a matter of support (in hardware, ISEs, parallel or serial libraries etc) vs speed. The counterpoint is that Kyber is basically fast enough already, and that introducing a different function causes more disruption than it's worth even pre-standardization. I would not want to go beyond the existing mode (sponge construction in counter mode) to avoid causing problems for hardware. In particular I would not want to use a Farfalle construction. But since expanding A is called on fixed-size objects, it's possible that it would not go beyond sponge mode even if TurboSHAKE can do this in general.

> To help weigh this, do you have a profile for what fraction of Kyber's
> (or Dilithium's) runtime is specifically the generation of A?

That's highly platform dependent. I just ran some benchmarks on a
Haswell machine for the AVX2-optimized implementation. What I'm getting
is:

====== KYBER 512 ======

keygen: 25120
encaps: 39180
decaps: 30868
gen_A: 7472
xof: 6088

====== KYBER 768 ======

keygen: 43596
encaps: 59464
decaps: 47684
gen_A: 20320
xof: 16865

====== KYBER 1024 ======

keygen: 60396
encaps: 83576
decaps: 67656
gen_A: 29664
xof: 24109

Here, "gen_A" is the full cycles for generation of the matrix A,
including the cycles for rejection sampling; "xof" is the cycles used on

SHAKE-128, i.e., the cycles that would pretty exactly halve when moving to TurboSHAKE.

Note that in this implementation, the generation of A is using a fast vectorized implementation of Keccak (vectorizing across different matrix entries), while many other calls to Keccak are sequential and thus much slower. On embedded platforms like the M4 I would expect the relative cost for "xof" to be considerably higher, simply because there is no speedup from vectorization. I can run those benchmarks when I'm back from my travels.

On the other hand, in a masked implementation I would expect the relative cost of "xof" to be much lower, simply because we don't require masking in the generation of A, but we do for most other Keccak calls.

With HW acceleration of Keccak it's somewhat hard to predict and it will depend a lot on how large the acceleration is: on the one hand, all Keccak calls will likely have the same cost (so matrix generation will be relatively more costly compared to other Keccak calls than in the AVX2 implementation), but on the other hand, one would expect polynomial arithmetic to use a more significant portion of the cycles, so overall the cost for Keccak permutations might not matter all that much.

Does this help?

All the best,

Peter

Hi Peter, thanks for this data.

It looks like the savings in the cases above, for a full key exchange assuming that decaps regenerates A, are about 10%, 17% and 17% for the three security levels. That’s not nothing, but it’s not a huge amount either. Also, in terms of absolute time savings on eg 3 GHz Haswell, even for Kyber 1024 we’re looking

at about 12µs savings for the entire key exchange. I'm not convinced that's worth the change, even though it's cryptographically fine.

Maybe on the M4 it will be relevant, though again I'm not sure many M4-class devices will be doing key exchanges that a human or time-relevant process will wait for.

I agree that in hardware, switching to TurboSHAKE won't typically save as much because, at least if Keccak is implemented in the most common one-cycle-per-round configuration, it won't be the bottleneck unless the NTT unit is very large. In fact, the savings might be almost zero if the XOF and NTT operate in parallel.

TurboSHAKE will also be annoying for hardware designers, because (speaking as a hardware designer) we will need to alter our accelerators to support it but as far as I can tell it isn't spec'd yet. Since hardware has a long development cycle, this will mean adding a bunch of extra knobs to the Keccak accelerator and hoping we picked the right ones.

Overall, I'm not in favor of this change but I don't think it will be a disaster either. There will be less risk for hardware projects if you can specify what TurboSHAKE will look like, possibly even before deciding whether to use it.

Regards,

— Mike

forum/D58209AC-08F2-4854-813D-06FB0875C2F4%40shiftleft.org.

--

From: dustin...@nist.gov <dustin.moody@nist.gov> via pqc-forum <pqc-forum@list.nist.gov>
To: pqc-forum <pqc-forum@list.nist.gov>
CC: Peter Schwabe <peter@cryptojedi.org>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto
Date: Tuesday, January 24, 2023 12:15:01 PM ET

At the 4th NIST PQC Standardization Conference, Gilles Van Assche gave a talk entitled "12 round-Keccak for secure hashing". Shortly after, Peter Schwabe asked for feedback (on behalf of the Kyber team) if the generation of the public matrix A should use a 12-round version of Keccak (i.e. "TurboSHAKE") instead of the standard 24-round version. The main reason for the change would be that using TurboSHAKE would speed up one of the costly subroutines of Kyber by about a factor of 2.

NIST appreciates the feedback provided back to the Kyber team on the pqc-forum. There were several advantages and disadvantages pointed out. We carefully read the arguments, and also discussed the question with our larger crypto team (and not just our PQC team). We wanted to let you know that we are NOT planning on incorporating a 12-round version of Keccak for any of the PQC standards we are currently working on. Our reasons for making this decision are provided below.

While speedups would be welcome, in our view the overall effect on a fully implementation of Kyber (or another PQC algorithm) would likely be modest - something on the order of around 20-25% or so. However, this would be a significant change to make, which would obviously be occurring after the third round, which means it may not receive as much public scrutiny and analysis. While the security requirements on the output of a reduced-round version of Keccak used for generating the A matrix are weak, we do not feel they have been completely described and studied. The Kyber team also pointed out a disadvantage in that moving to a 12-round version of Keccak would require phrasing the function in terms of lower-level functions of FIPS202 instead of simply using one of the SHA3/SHAKE functions. Considering the above, we did not feel the potential advantages outweighed the disadvantages. NIST encourages further study of TurboSHAKE.

Dustin Moody

NIST PQC team

On Friday, December 9, 2022 at 10:47:37 AM UTC-5 Stott, David - 0553 - MITLL wrote:

Thanks, Mike.

We have looked at a softcore based design with a SHAKE coprocessor (without the NTT acceleration). Our findings support your comments below.

- The benefit of TurboSHAKE would be very small for HW (12 cycles per keccak-permute round), where our current unoptimized coprocessor interface spends more than 100 cycles transferring data
- With HW acceleration of SHA3 (SHAKE or TurboSHAKE) the critical path becomes NTT

The actual changes to the code to go to 12 rounds seem rather minor (granted we are not working on a production-quality implementation).

The software-only version, of course, is a different story.

-david

From: <pqc-forum@list.nist.gov> on behalf of Mike Hamburg <mike@shiftleft.org>
Date: Friday, December 9, 2022 at 9:54 AM
To: Peter Schwabe <peter@cryptojedi.org>
Cc: "Markku-Juhani O. Saarinen" <mjos.crypto@gmail.com>, pqc-forum <pqc-forum@list.nist.gov>, John Mattsson <john.mattsson@ericsson.com>, "Blumenthal, Uri - 0553 - MITLL" <uri@ll.mit.edu>
Subject: Re: [pqc-forum] Kyber decisions, part 1: Symmetric crypto

> On Dec 9, 2022, at 1:14 PM, Peter Schwabe <peter@cryptojedi.org> wrote:
>
> Mike Hamburg <mike@shiftleft.org> wrote:
>
>> Hi all,
>
> Hi Mike, hi all,
>
>> I mostly agree with Markku here.
>>
>> I think it's better to domain separate based on input encoding instead
>> of based on cSHAKE. It's not as broadly supported as SHAKE, and if

> you have several separate instances then you either need an extra permutation call or else several precomputed 200-byte states, which is significant in hardware or perhaps in embedded software. For ThreeBears I used cSHAKE(parameters || purpose || data) with a fixed personalization string “ThreeBears”, but raw SHAKE would have been about as good — using cSHAKE and personalization was probably overkill. Here the `parameters` are different between different KEM instances (in your case eg Kyber512 vs Kyber768 vs Kyber1024). The goal is that if someone accidentally reuses a seed string for different key sizes, they won’t get closely related keys. The `purpose` is what you’re using it for (hash, keygen, generate A, encrypt, PRF etc). The whole (parameters || purpose) block has a fixed 8-byte aligned size, in case clients are optimized for that.
> You could also put the counter in the purpose block, but I put it at the end to simplify use of parallel hashing APIs.
>
> I agree that it would be cryptographically fine to generate A using TurboSHAKE, and whether to do this is mostly a matter of support (in hardware, ISEs, parallel or serial libraries etc) vs speed. The counterpoint is that Kyber is basically fast enough already, and that introducing a different function causes more disruption than it’s worth even pre-standardization. I would not want to go beyond the existing mode (sponge construction in counter mode) to avoid causing problems for hardware. In particular I would not want to use a Farfalle construction. But since expanding A is called on fixed-size objects, it’s possible that it would not go beyond sponge mode even if TurboSHAKE can do this in general.
>
> To help weigh this, do you have a profile for what fraction of Kyber’s (or Dilithium’s) runtime is specifically the generation of A?

That's highly platform dependent. I just ran some benchmarks on a Haswell machine for the AVX2-optimized implementation. What I'm getting is:

====== KYBER 512 ======

keygen: 25120

encaps: 39180

decaps: 30868

gen_A: 7472

xof: 6088

====== KYBER 768 ======

keygen: 43596

encaps: 59464

decaps: 47684

gen_A: 20320

xof: 16865

====== KYBER 1024 ======

keygen: 60396

encaps: 83576

decaps: 67656

gen_A: 29664

xof: 24109

Here, "gen_A" is the full cycles for generation of the matrix A,
including the cycles for rejection sampling; "xof" is the cycles used on
SHAKE-128, i.e., the cycles that would pretty exactly halve when moving
to TurboSHAKE.

Note that in this implementation, the generation of A is using a fast
vectorized implementation of Keccak (vectorizing across different matrix
entries), while many other calls to Keccak are sequential and thus much
slower. On embedded platforms like the M4 I would expect the relative
cost for "xof" to be considerably higher, simply because there is no
speedup from vectorization. I can run those benchmarks when I'm back

> from my travels.
>
> On the other hand, in a masked implementation I would expect the
> relative cost of "xof" to be much lower, simply because we don't require
> masking in the generation of A, but we do for most other Keccak calls.
>
> With HW acceleration of Keccak it's somewhat hard to predict and it will
> depend a lot on how large the acceleration is: on the one hand, all
> Keccak calls will likely have the same cost (so matrix generation will
> be relatively more costly compared to other Keccak calls than in the
> AVX2 implementation), but on the other hand, one would expect polynomial
> arithmetic to use a more significant portion of the cycles, so overall
> the cost for Keccak permutations might not matter all that much.
>
> Does this help?
>
> All the best,
>
> Peter

Hi Peter, thanks for this data.

It looks like the savings in the cases above, for a full key exchange assuming that decaps regenerates A, are about 10%, 17% and 17% for the three security levels. That’s not nothing, but it’s not a huge amount either. Also, in terms of absolute time savings on eg 3 GHz Haswell, even for Kyber 1024 we’re looking at about 12µs savings for the entire key exchange. I’m not convinced that’s worth the change, even though it’s cryptographically fine.

Maybe on the M4 it will be relevant, though again I’m not sure many M4-class devices will be doing key exchanges that a human or time-relevant process will wait for.

I agree that in hardware, switching to TurboSHAKE won’t typically save as much

> because, at least if Keccak is implemented in the most common one-cycle-per-round configuration, it won't be the bottleneck unless the NTT unit is very large.
>
> In fact, the savings might be almost zero if the XOF and NTT operate in parallel.
>
> TurboSHAKE will also be annoying for hardware designers, because (speaking as a hardware designer) we will need to alter our accelerators to support it but as far as I can tell it isn't spec'd yet. Since hardware has a long development cycle, this will mean adding a bunch of extra knobs to the Keccak accelerator and hoping we picked the right ones.
>
> Overall, I'm not in favor of this change but I don't think it will be a disaster either.
>
> There will be less risk for hardware projects if you can specify what TurboSHAKE will look like, possibly even before deciding whether to use it.
>
> Regards,
>
> — Mike
>
> --
>
>

--

To view this discussion on the web visit https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/c2c3ddca-6286-43c0-9f97-82526a4fcf24n%40list.nist.gov.