From: Mike Ounsworth <mike.ounsworth@entrust.com> via pqc-forum <pqc-forum@list.nist.gov>
To: LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, jakemas@amazon.com, kpanos@amazon.com, sean@ssn3rd.com, bas@westerbaan.name
Subject: [pqc-forum] Whether to hash-then-sign with Dilithium and Falcon?
Date: Wednesday, August 17, 2022 01:27:19 PM ET

Hi Jake, Panos, Sean, Bas,

We notice that your IETF draft-massimo-lamps-pq-sig-certificates-00 has the following security consideration:

> Within the hash-then-sign paradigm, hash functions are used as a domain restrictor over the message to be signed. By pre-hashing, the onus of resistance to
> existential forgeries becomes heavily reliant on the collision-resistance of the hash function in use. As well as this security goal, the hash-then-sign paradigm also
> has the ability to improve performance by reducing the size of signed messages. As a corollary, hashing remains mandatory even for short messages and assigns a
> further computational requirement onto the verifier. This makes the performance of hash-then-sign schemes more consistent, but not necessarily more efficient.
> Dilithium diverges from the hash-then-sign paradigm by hashing the message during the signing procedure (at the point in which the challenge polynomial).
> However, due to the fact that Dilithium signatures may require the signing procedure to be repeated several times for a signature to be produced, Dilithium
> implementations can make use of pre-hashing the message to prevent rehashing with each attempt.

First, quoting from the Dilithium NIST Round 3 submission documents:

> Since our signing procedure may need to
> be repeated several times until a signature is produced, we also append a counter in order
> to make the SHAKE-256 output differ with each signing attempt of the same message.

So it seems like the Dilithium designers explicitly want the hash to differ across repeated attempts.

Second, we had a similar discussion within the context of composite signatures when figuring out how to combine Dilithium and Falcon with ECDSA and RSA. We came out with a different conclusion; that hash-then-sign reduces the security properties of Dilithium and Falcon down to the collision resistance of the hash function used to pre-hash.

We would like community opinion on this.

Here's the Security Consideration text that we're working on:

In the hash-then-sign paradigm, the message to be signed is hashed externally to the signature primitive, and then the hash value is signed.

The hash-then-sign paradigm is required, for example, with RSA signatures in order to sign messages larger than the RSA modulus. Hash-then-sign also gives performance and bandwidth benefits, for example, when the signature is performed by a networked cryptographic appliance since you only need to send a small hash value rather than streaming the entire message.

With Dilithium and Falcon signatures it is not recommended to pre-hash for the following reasons:

The Dilithium construction includes

~~~
Sign(sk,M):
10: mu \in {0, 1}^384 := CRH(tr || M)
~~~

where `CRH` is any collision-resistant hash function and `tr` is a component of the secret key. This provides strong security against pre-computed collision attacks since an attacker has no a-priori knowledge of `r` and provides per-key hash-domain separation of the message to be signed.

The Falcon construction includes

~~~
Sign (m, sk, beta^2):
1: r ← {0, 1}^320 uniformly
2: c ← HashToPoint(r || m, q, n)
~~~

where `HashToPoint` is a SHAKE-256-based construct. This provides strong security against pre-computed collision attacks since an attacker has no a-priori knowledge of `r` and provides per-signature hash-domain separation of the message to be signed.

If the message to be signed is pre-hashed, for example `m0 = SHA256(m)` and then m0 provided to Dilithium or Falcon to sign, then you have re-introduced the collision problem since two messages m1 and m2 where SHA256(m1) == SHA256(m2) hash value will result a single Falcon or Dilithium signature value which is simultaneously valid for both m1 and m2. This removes the extra collision resistance built in to the Dilithium and Falcon primitives and reduces it to the collision resistance strength of the underlying hash function. For this reason it is in general not recommended to pre-hash when using Dilithium or Falcon except in cases where the implementor is comfortable with this reduction in security.

Therefore, for the purpose of interoperability of composite signatures, implementations MUST NOT pre-hash messages for Dilithium and Falcon. If pre-hashed versions of these signatures are desired, then separate signature algorithms will need to be defined.

Third, I can imagine that some applications (like TLS) will want to use non-pre-hashed versions of Dilithium and Falcon, but other applications (like code-signing) would prefer pre-hashed versions. These are not interoperable with each other. Is NIST planning to produce algorithm definitions, OIDs, Codepoints, etc, for both versions?

—

Mike Ounsworth
Software Security Architect, Entrust

John Gray
Sr Prin Software Developer, Entrust

--

From: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Mike Ounsworth <mike.ounsworth@entrust.com>, LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, jakemas@amazon.com, kpanos@amazon.com, sean@ssn3rd.com, bas@westerbaan.name
Subject: [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Wednesday, August 17, 2022 01:50:24 PM ET

> ————Original Message————
> From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
> Sent: Wednesday, August 17, 2022 1:27 PM
> To: 'LAMPS' <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-
> forum@list.nist.gov>; jakemas@amazon.com; kpanos@amazon.com;
> sean@ssn3rd.com; bas@westerbaan.name
> Subject: [pqc-forum] Whether to hash-then-sign with Dilithium and Falcon?
>
> Hi Jake, Panos, Sean, Bas,
>
>
> We notice that your IETF draft-massimo-lamps-pq-sig-certificates-00 has the
> following security consideration:
>
> > Within the hash-then-sign paradigm, hash functions are used as a
> > domain restrictor over the message to be signed. By pre-hashing, the
> > onus of resistance to existential forgeries becomes heavily reliant on
> > the collision-resistance of the hash function in use. As well as this security
> goal, the hash-then-sign paradigm also has the ability to improve
> performance by reducing the size of signed messages. As a corollary, hashing
> remains mandatory even for short messages and assigns a further
> computational requirement onto the verifier. This makes the performance of
> hash-then-sign schemes more consistent, but not necessarily more efficient.
> > Dilithium diverges from the hash-then-sign paradigm by hashing the
> message during the signing procedure (at the point in which the challenge
> polynomial).
> > However, due to the fact that Dilithium signatures may require the
> > signing procedure to be repeated several times for a signature to be
> produced, Dilithium implementations can make use of pre-hashing the
> message to prevent rehashing with each attempt.
>

>
> First, quoting from the Dilithium NIST Round 3 submission documents:
>
> > Since our signing procedure may need to be repeated several times
> > until a signature is produced, we also append a counter in order to
> > make the SHAKE-256 output differ with each signing attempt of the same
> message.
>
> So it seems like the Dilithium designers explicitly want the hash to differ
> across repeated attempts.
>

Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.

In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...

>
>
> Second, we had a similar discussion within the context of composite
> signatures when figuring out how to combine Dilithium and Falcon with
> ECDSA and RSA. We came out with a different conclusion; that hash-then-
> sign reduces the security properties of Dilithium and Falcon down to the
> collision resistance of the hash function used to pre-hash.
>
> We would like community opinion on this.
>
>
> Here's the Security Consideration text that we're working on:
>
>
>
>
> In the hash-then-sign paradigm, the message to be signed is hashed
> externally to the signature primitive, and then the hash value is signed.

>
> The hash-then-sign paradigm is required, for example, with RSA signatures in
> order to sign messages larger than the RSA modulus. Hash-then-sign also
> gives performance and bandwidth benefits, for example, when the signature
> is performed by a networked cryptographic appliance since you only need to
> send a small hash value rather than streaming the entire message.
>
> With Dilithium and Falcon signatures it is not recommended to pre-hash for
> the following reasons:
>
>
> The Dilithium construction includes
>
> ~~~
> Sign(sk,M):
> 10: mu \in {0, 1}^384 := CRH(tr || M)
> ~~~
>
> where `CRH` is any collision-resistant hash function and `tr` is a component
> of the secret key.

A hash of the public key, actually; see line 7 of the key generation process (which explicitly computes it from the components of the public key) - Dilithium stores it in the private key so the signer doesn't need to recompute it every time.

> This provides strong security against pre-computed
> collision attacks since an attacker has no a-priori knowledge of `r` and
> provides per-key hash-domain separation of the message to be signed.

Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Now, it does probably add a constant factor to any attack that searches for a simultaneous collision between the hash that RSA/ECDSA uses (without the prepend) and the hash that Dilithium uses (with the known prepend) - I would hesitate to give a value to that constant factor, but it is likely not large.

>
>
> The Falcon construction includes
>
> ~~~
> Sign (m, sk, beta^2):
> 1: r ← {0, 1}^320 uniformly
> 2: c ← HashToPoint(r || m, q, n)
> ~~~
>
> where `HashToPoint` is a SHAKE-256-based construct. This provides strong
> security against pre-computed collision attacks since an attacker has no a-
> priori knowledge of `r` and provides per-signature hash-domain separation
> of the message to be signed.
>
> If the message to be signed is pre-hashed, for example `m0 = SHA256(m)`
> and then m0 provided to Dilithium or Falcon to sign, then you have re-
> introduced the collision problem since two messages m1 and m2 where
> SHA256(m1) == SHA256(m2) hash value will result a single Falcon or Dilithium
> signature value which is simultaneously valid for both m1 and m2. This
> removes the extra collision resistance built in to the Dilithium and Falcon
> primitives and reduces it to the collision resistance strength of the underlying
> hash function. For this reason it is in general not recommended to pre-hash
> when using Dilithium or Falcon except in cases where the implementor is
> comfortable with this reduction in security.
>
> Therefore, for the purpose of interoperability of composite signatures,
> implementations MUST NOT pre-hash messages for Dilithium and Falcon. If
> pre-hashed versions of these signatures are desired, then separate signature
> algorithms will need to be defined.
>
>

From: Mike Ounsworth <mike.ounsworth@entrust.com> via pqc-forum <pqc-forum@list.nist.gov>
To: LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, jakemas@amazon.com, kpanos@amazon.com, sean@ssn3rd.com, bas@westerbaan.name
Subject: [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Wednesday, August 17, 2022 02:42:52 PM ET

I want to break out and expand our third point as it is actually a question to NIST and not to the IETF authors.

> Third, I can imagine that some applications (like TLS) will want to use non-pre-hashed versions of Dilithium and Falcon, but other applications (like code-signing) would prefer pre-hashed versions. These are not interoperable with each other. Is NIST planning to produce algorithm definitions, OIDs, Codepoints, etc, for both versions?

Expanding on the code-signing example: the messages to be signed can be very large; consider a several GB firmware image. Assuming our understanding below is correct, a direct-sign algorithm would require the entire thing to be streamed to a network HSM for signing and to a TPM for verification. Conversely code-signing environments often include counter-signatures from Time Stamping Authorities which protect against future discovery of collision attacks against the hash function -- as an example, Windows still accepts RSA-SHA1 signatures produced before SHA1 was deprecated. I can imagine that the code-signing community will decide that the performance gains of hash-then-sign outweigh the security loss.

So, will NIST standardize both direct-sign and some variant of hash-then-sign for PQC signature primitives?

—
Mike Ounsworth

————Original Message————
From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
Sent: August 17, 2022 12:27 PM
To: 'LAMPS' <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-forum@list.nist.gov>; jakemas@amazon.com; kpanos@amazon.com; sean@ssn3rd.com; bas@westerbaan.name
Subject: [EXTERNAL] [pqc-forum] Whether to hash-then-sign with Dilithium and Falcon?

Hi Jake, Panos, Sean, Bas,

We notice that your IETF draft-massimo-lamps-pq-sig-certificates-00 has the following security consideration:

> Within the hash-then-sign paradigm, hash functions are used as a
> domain restrictor over the message to be signed. By pre-hashing, the
> onus of resistance to existential forgeries becomes heavily reliant on
> the collision-resistance of the hash function in use. As well as this security goal, the hash-then-sign paradigm also has the ability to improve performance by reducing the size of signed messages. As a corollary, hashing remains mandatory even for short messages and assigns a further computational requirement onto the verifier. This makes the performance of hash-then-sign schemes more consistent, but not necessarily more efficient.
> Dilithium diverges from the hash-then-sign paradigm by hashing the message during the signing procedure (at the point in which the challenge polynomial).
> However, due to the fact that Dilithium signatures may require the
> signing procedure to be repeated several times for a signature to be produced, Dilithium implementations can make use of pre-hashing the message to prevent rehashing with each attempt.

First, quoting from the Dilithium NIST Round 3 submission documents:

> Since our signing procedure may need to be repeated several times
> until a signature is produced, we also append a counter in order to
> make the SHAKE-256 output differ with each signing attempt of the same message.

So it seems like the Dilithium designers explicitly want the hash to differ across repeated attempts.

Second, we had a similar discussion within the context of composite signatures when figuring out how to combine Dilithium and Falcon with ECDSA and RSA. We came out with a different conclusion; that hash-then-sign reduces the security properties of Dilithium and Falcon down to the collision resistance of the hash function used to pre-hash.

We would like community opinion on this.

Here's the Security Consideration text that we're working on:

In the hash-then-sign paradigm, the message to be signed is hashed externally to the signature primitive, and then the hash value is signed.

The hash-then-sign paradigm is required, for example, with RSA signatures in order to sign messages larger than the RSA modulus. Hash-then-sign also gives performance and bandwidth benefits, for example, when the signature is performed by a networked cryptographic appliance since you only need to send a small hash value rather than streaming the entire message.

With Dilithium and Falcon signatures it is not recommended to pre-hash for the following reasons:

The Dilithium construction includes

~~~
Sign(sk,M):
10: mu \in {0, 1}^384 := CRH(tr || M)
~~~

where `CRH` is any collision-resistant hash function and `tr` is a component of the secret key. This provides strong security against pre-computed collision attacks since an attacker has no a-priori knowledge of `r` and provides per-key hash-domain separation of the message to be signed.

The Falcon construction includes

~~~
Sign (m, sk, beta^2):
1: r ← {0, 1}^320 uniformly
2: c ← HashToPoint(r || m, q, n)
~~~

where `HashToPoint` is a SHAKE-256-based construct. This provides strong security against pre-computed collision attacks since an attacker has no a-priori knowledge of `r` and provides per-signature hash-domain separation of the message to be signed.

If the message to be signed is pre-hashed, for example `m0 = SHA256(m)` and then m0 provided to Dilithium or Falcon to sign, then you have re-introduced the collision problem since two messages m1 and m2 where SHA256(m1) == SHA256(m2) hash value will result a single Falcon or Dilithium signature value which is simultaneously valid for both m1 and m2. This removes the extra collision resistance built in to the Dilithium and Falcon primitives and reduces it to the collision resistance strength of the underlying hash function. For this reason it is in general not recommended to pre-hash when using Dilithium or Falcon except in cases where the implementor is comfortable with this reduction in security.

Therefore, for the purpose of interoperability of composite signatures, implementations MUST NOT pre-hash messages for Dilithium and Falcon. If pre-hashed versions of these signatures are desired, then separate signature algorithms will need to be defined.

Third, I can imagine that some applications (like TLS) will want to use non-pre-hashed versions of Dilithium and Falcon, but other applications (like code-signing) would prefer pre-hashed versions. These are not interoperable with each other. Is NIST planning to produce algorithm definitions, OIDs, Codepoints, etc, for both versions?

___

Mike Ounsworth
Software Security Architect, Entrust

John Gray
Sr Prin Software Developer, Entrust

--

From: Massimo, Jake <jakemas@amazon.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>, Mike Ounsworth <mike.ounsworth@entrust.com>, LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, Kampanakis, Panos <kpanos@amazon.com>, sean@ssn3rd.com, bas@westerbaan.name
Subject: Re: [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Wednesday, August 17, 2022 03:40:39 PM ET

Thanks Mike, Scott.

I've added to the github repo so we can track discussions on this topic https://github.com/jakemas/draft-massimo-pq-pkix-00/issues/23

> >> So it seems like the Dilithium designers explicitly want the hash to differ
> >> across repeated attempts.
> >>

> Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.
> >In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...

I think we are referring to different parts of the signing process here. For reference, my security consideration was referring to page 4 of the Dilithium spec that states:
"Our full scheme in Fig. 4 also makes use of basic optimizations such as pre-hashing the message M so as to not rehash it with every signing attempt." and Figure 4 itself.

It was my understanding that the signing procedure may need to be repeated several times to produce a signature, and thus pre-hashing would prevent the need to individually hash the input message with each attempt. I believe the desired differing of the hash you mentioned is within the internals of the signing procedure and not on the input message itself.

>> Third, I can imagine that some applications (like TLS) will want to use non-pre-hashed versions of Dilithium and Falcon, but other applications (like code-signing) would prefer pre-hashed versions. These are not interoperable with each other. Is NIST planning to produce algorithm definitions, OIDs, Codepoints, etc, for both versions?

>Expanding on the code-signing example: the messages to be signed can be very large; consider a several GB firmware image. Assuming our understanding below is correct, a direct-sign algorithm would require the entire thing to be streamed to a network HSM for signing and to a TPM for verification. Conversely code-signing environments often include counter-signatures from Time Stamping Authorities which protect against future discovery of collision attacks against the hash function -- as an example, Windows still accepts RSA-SHA1 signatures produced before SHA1 was deprecated. I can imagine that the code-signing community will decide that the performance gains of hash-then-sign outweigh the security loss.

>So, will NIST standardize both direct-sign and some variant of hash-then-sign for PQC signature primitives?

I do agree that there may be optimizations that users may wish to make dependent on the context, i.e., hash-then-sign vs direct-sign. It's for this reason I tried to give an overview of the security of each option in the draft, but ultimately leave that up to the user. It is a good point regarding NISTs perspective on what should be explicitly standardized here.

>> This provides strong security against pre-computed
>> collision attacks since an attacker has no a-priori knowledge of `r` and
>> provides per-key hash-domain separation of the message to be signed.

>Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Right, more details on the advantages of message binding on the PQC-forum from C. Peikert's https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/eAaiJO1qzkA/m/K66R_ftNBwAJ. It was this discussion I was trying to encompass in the draft.

Cheers,
Jake

———

On 17/08/2022, 10:51, "'Scott Fluhrer (sfluhrer)' via pqc-forum" <pqc-forum@list.nist.gov> wrote:

CAUTION: This email originated from outside of the organization. Do not click links or open attachments unless you can confirm the sender and know the content is safe.

> ———Original Message———
> From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
> Sent: Wednesday, August 17, 2022 1:27 PM
> To: 'LAMPS' <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-
> forum@list.nist.gov>; jakemas@amazon.com; kpanos@amazon.com;
> sean@ssn3rd.com; bas@westerbaan.name
> Subject: [pqc-forum] Whether to hash-then-sign with Dilithium and Falcon?
>
> Hi Jake, Panos, Sean, Bas,
>
>
> We notice that your IETF draft-massimo-lamps-pq-sig-certificates-00 has the
> following security consideration:
>
> > Within the hash-then-sign paradigm, hash functions are used as a
> > domain restrictor over the message to be signed. By pre-hashing, the
> > onus of resistance to existential forgeries becomes heavily reliant on
> > the collision-resistance of the hash function in use. As well as this security
> goal, the hash-then-sign paradigm also has the ability to improve
> performance by reducing the size of signed messages. As a corollary, hashing
> remains mandatory even for short messages and assigns a further
> computational requirement onto the verifier. This makes the performance of
> hash-then-sign schemes more consistent, but not necessarily more efficient.

> > Dilithium diverges from the hash-then-sign paradigm by hashing the
> message during the signing procedure (at the point in which the challenge
> polynomial).
> > However, due to the fact that Dilithium signatures may require the
> > signing procedure to be repeated several times for a signature to be
> produced, Dilithium implementations can make use of pre-hashing the
> message to prevent rehashing with each attempt.
>
>
> First, quoting from the Dilithium NIST Round 3 submission documents:
>
> > Since our signing procedure may need to be repeated several times
> > until a signature is produced, we also append a counter in order to
> > make the SHAKE-256 output differ with each signing attempt of the same
> message.
>
> So it seems like the Dilithium designers explicitly want the hash to differ
> across repeated attempts.
>

Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.

In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...

>
>
> Second, we had a similar discussion within the context of composite
> signatures when figuring out how to combine Dilithium and Falcon with
> ECDSA and RSA. We came out with a different conclusion; that hash-then-
> sign reduces the security properties of Dilithium and Falcon down to the
> collision resistance of the hash function used to pre-hash.
>
> We would like community opinion on this.
>

>
> Here's the Security Consideration text that we're working on:
>
>
>
>
> In the hash-then-sign paradigm, the message to be signed is hashed
> externally to the signature primitive, and then the hash value is signed.
>
> The hash-then-sign paradigm is required, for example, with RSA signatures in
> order to sign messages larger than the RSA modulus. Hash-then-sign also
> gives performance and bandwidth benefits, for example, when the signature
> is performed by a networked cryptographic appliance since you only need to
> send a small hash value rather than streaming the entire message.
>
> With Dilithium and Falcon signatures it is not recommended to pre-hash for
> the following reasons:
>
>
> The Dilithium construction includes
>
> ~~~
> Sign(sk,M):
> 10: mu \in {0, 1}^384 := CRH(tr || M)
> ~~~
>
> where `CRH` is any collision-resistant hash function and `tr` is a component
> of the secret key.

A hash of the public key, actually; see line 7 of the key generation process (which explicitly computes it from the components of the public key) - Dilithium stores it in the private key so the signer doesn't need to recompute it every time.

> This provides strong security against pre-computed
> collision attacks since an attacker has no a-priori knowledge of `r` and
> provides per-key hash-domain separation of the message to be signed.

Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Now, it does probably add a constant factor to any attack that searches for a simultaneous collision between the hash that RSA/ECDSA uses (without the prepend) and the hash that Dilithium uses (with the known prepend) - I would hesitate to give a value to that constant factor, but it is likely not large.

>
>
> The Falcon construction includes
>
> ~~~
> Sign (m, sk, beta^2):
> 1: r ← {0, 1}^320 uniformly
> 2: c ← HashToPoint(r || m, q, n)
> ~~~
>
> where `HashToPoint` is a SHAKE-256-based construct. This provides strong
> security against pre-computed collision attacks since an attacker has no a-
> priori knowledge of `r` and provides per-signature hash-domain separation
> of the message to be signed.
>
> If the message to be signed is pre-hashed, for example `m0 = SHA256(m)`
> and then m0 provided to Dilithium or Falcon to sign, then you have re-
> introduced the collision problem since two messages m1 and m2 where
> SHA256(m1) == SHA256(m2) hash value will result a single Falcon or Dilithium
> signature value which is simultaneously valid for both m1 and m2. This
> removes the extra collision resistance built in to the Dilithium and Falcon
> primitives and reduces it to the collision resistance strength of the
underlying
> hash function. For this reason it is in general not recommended to pre-hash
> when using Dilithium or Falcon except in cases where the implementor is
> comfortable with this reduction in security.
>
> Therefore, for the purpose of interoperability of composite signatures,
> implementations MUST NOT pre-hash messages for Dilithium and Falcon. If

> pre-hashed versions of these signatures are desired, then separate signature
> algorithms will need to be defined.
>
>

--

--

From: Tadahiko Ito <tadahiko.ito.public@gmail.com> via Spasm <spasm-bounces@ietf.org>
To: Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>
CC: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>, Mike Ounsworth <mike.ounsworth@entrust.com>, LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, Kampanakis, Panos <kpanos@amazon.com>, sean@ssn3rd.com, bas@westerbaan.name
Subject: Re: [lamps] [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Thursday, August 18, 2022 01:13:44 AM ET
Attachments: ATT00001.txt

> It was my understanding that the signing procedure may need to be
> repeated several times to produce a signature, and thus pre-hashing
> would prevent the need to individually hash the input message with
> each attempt.

When we were trying to implement PQC in functional module of HSM for our use case, It was pain. I believe HSM vender will implement much better, but It may still have problem.
Hash then Sign was great that we can separate key management (and signing) function from data management (and hashing) function. It seems crypto module producer might need to implement scheduling function for data management function. I far those problems decrease efficiency, but we might need to care that.

>> (...) Assuming our understanding below is correct, a direct-sign algorithm
>> would require the entire thing to be streamed to a network HSM for signing
>> and to a TPM for verification.

Currently, I am doubting that we might not have that many protocols with direct-signing algorithm which would sign intolerable large data. For those protocol with direct-signing, I believe we can have several approaches.

1)Sign to smaller compressed data (e.g. by using CMS) instead of raw data.
It was biggest feedback I got so far, when I told about those stuff on IETF last year.
For this option, Users may need to change data structure, but If we cannot find that much direct-signing use case, it might be reasonable. Direct-signing use case holders may need to take other option.
In addition, when I ask our engineer for our use case, he said that was long recognized issue, and it might be good chance to do so.

2)Use pre-hash
Users do not need to change data structure, but we may meet interoperability challenge.

3)Separate PQC into key management function and data management function,
I tried, but I believe It was not good choice. <https://eprint.iacr.org/2020/990.pdf> (I am sorry that we

have not updates that document.)

4)Ask NIST to make hash-and-sign PQC
If they make one, it would be easy. (well.. I believe we should not assume that)
Regards Tadahiko

2022年8月18日(木) 4:41 Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>:

> Thanks Mike, Scott.
>
> I've added to the github repo so we can track discussions on this topic https://github.com/jakemas/draft-massimo-pq-pkix-00/issues/23
>
> >> So it seems like the Dilithium designers explicitly want the hash to differ
> >> across repeated attempts.
> >>
>
> > Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.
> >In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...
>
> I think we are referring to different parts of the signing process here. For reference, my security consideration was referring to page 4 of the Dilithium spec that states:
> "Our full scheme in Fig. 4 also makes use of basic optimizations such as pre-hashing the message M so as to not rehash it with every signing attempt." and Figure 4 itself.
>
> It was my understanding that the signing procedure may need to be repeated several times to produce a signature, and thus pre-hashing would prevent the need to individually hash the input message with each attempt. I believe the desired differing of the hash you mentioned is within the internals of the signing procedure and not on the input message itself.
>
> >> Third, I can imagine that some applications (like TLS) will want to use non-pre-hashed versions of Dilithium and Falcon, but other applications (like code-signing) would prefer pre-hashed versions. These are not interoperable with each other. Is NIST planning to produce algorithm definitions, OIDs, Codepoints, etc, for both versions?

>Expanding on the code-signing example: the messages to be signed can be very large; consider a several GB firmware image. Assuming our understanding below is correct, a direct-sign algorithm would require the entire thing to be streamed to a network HSM for signing and to a TPM for verification. Conversely code-signing environments often include counter-signatures from Time Stamping Authorities which protect against future discovery of collision attacks against the hash function -- as an example, Windows still accepts RSA-SHA1 signatures produced before SHA1 was deprecated. I can imagine that the code-signing community will decide that the performance gains of hash-then-sign outweigh the security loss.

>So, will NIST standardize both direct-sign and some variant of hash-then-sign for PQC signature primitives?

I do agree that there may be optimizations that users may wish to make dependent on the context, i.e., hash-then-sign vs direct-sign. It's for this reason I tried to give an overview of the security of each option in the draft, but ultimately leave that up to the user. It is a good point regarding NISTs perspective on what should be explicitly standardized here.

>> This provides strong security against pre-computed
>> collision attacks since an attacker has no a-priori knowledge of `r` and
>> provides per-key hash-domain separation of the message to be signed.

>Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Right, more details on the advantages of message binding on the PQC-forum from C. Peikert's https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/eAaiJO1qzkA/m/K66R_ftNBwAJ. It was this discussion I was trying to encompass in the draft.

Cheers,
Jake

On 17/08/2022, 10:51, "'Scott Fluhrer (sfluhrer)' via pqc-forum" <pqc-forum@list.nist.gov>

wrote:

CAUTION: This email originated from outside of the organization. Do not click links or open attachments unless you can confirm the sender and know the content is safe.

> -----Original Message-----
> From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
> Sent: Wednesday, August 17, 2022 1:27 PM
> To: 'LAMPS' <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-
> forum@list.nist.gov>; jakemas@amazon.com; kpanos@amazon.com;
> sean@ssn3rd.com; bas@westerbaan.name
> Subject: [pqc-forum] Whether to hash-then-sign with Dilithium and Falcon?
>
> Hi Jake, Panos, Sean, Bas,
>
>
> We notice that your IETF draft-massimo-lamps-pq-sig-certificates-00 has the
> following security consideration:
>
> > Within the hash-then-sign paradigm, hash functions are used as a
> > domain restrictor over the message to be signed. By pre-hashing, the
> > onus of resistance to existential forgeries becomes heavily reliant on
> > the collision-resistance of the hash function in use. As well as this security
> goal, the hash-then-sign paradigm also has the ability to improve
> performance by reducing the size of signed messages. As a corollary, hashing
> remains mandatory even for short messages and assigns a further
> computational requirement onto the verifier. This makes the performance of
> hash-then-sign schemes more consistent, but not necessarily more efficient.
> > Dilithium diverges from the hash-then-sign paradigm by hashing the
> message during the signing procedure (at the point in which the challenge
> polynomial).
> > However, due to the fact that Dilithium signatures may require the
> > signing procedure to be repeated several times for a signature to be
> produced, Dilithium implementations can make use of pre-hashing the
> message to prevent rehashing with each attempt.

>
>
> First, quoting from the Dilithium NIST Round 3 submission documents:
>
> > Since our signing procedure may need to be repeated several times
> > until a signature is produced, we also append a counter in order to
> > make the SHAKE-256 output differ with each signing attempt of the same
> message.
>
> So it seems like the Dilithium designers explicitly want the hash to differ
> across repeated attempts.
>

Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.

In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...

>
>
> Second, we had a similar discussion within the context of composite
> signatures when figuring out how to combine Dilithium and Falcon with
> ECDSA and RSA. We came out with a different conclusion; that hash-then-
> sign reduces the security properties of Dilithium and Falcon down to the
> collision resistance of the hash function used to pre-hash.
>
> We would like community opinion on this.
>
>
> Here's the Security Consideration text that we're working on:
>
>
>
>
>
> In the hash-then-sign paradigm, the message to be signed is hashed

> externally to the signature primitive, and then the hash value is signed.
>
> The hash-then-sign paradigm is required, for example, with RSA signatures in
> order to sign messages larger than the RSA modulus. Hash-then-sign also
> gives performance and bandwidth benefits, for example, when the signature
> is performed by a networked cryptographic appliance since you only need to
> send a small hash value rather than streaming the entire message.
>
> With Dilithium and Falcon signatures it is not recommended to pre-hash for
> the following reasons:
>
>
> The Dilithium construction includes
>
> ~~~
> Sign(sk,M):
> 10: mu \in {0, 1}^384 := CRH(tr || M)
> ~~~
>
> where `CRH` is any collision-resistant hash function and `tr` is a component
> of the secret key.

A hash of the public key, actually; see line 7 of the key generation process (which explicitly computes it from the components of the public key) - Dilithium stores it in the private key so the signer doesn't need to recompute it every time.

> This provides strong security against pre-computed
> collision attacks since an attacker has no a-priori knowledge of `r` and
> provides per-key hash-domain separation of the message to be signed.

Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Now, it does probably add a constant factor to any attack that searches for a simultaneous collision between the hash that RSA/ECDSA uses (without the prepend) and the hash that Dilithium uses (with the known prepend) - I would hesitate to give a value to that constant factor, but it is likely not large.

>
>
> The Falcon construction includes
>
> ~~~
> Sign (m, sk, beta^2):
> 1: r <- {0, 1}^320 uniformly
> 2: c <- HashToPoint(r || m, q, n)
> ~~~
>
> where `HashToPoint` is a SHAKE-256-based construct. This provides strong
> security against pre-computed collision attacks since an attacker has no a-
> priori knowledge of `r` and provides per-signature hash-domain separation
> of the message to be signed.
>
> If the message to be signed is pre-hashed, for example `m0 = SHA256(m)`
> and then m0 provided to Dilithium or Falcon to sign, then you have re-
> introduced the collision problem since two messages m1 and m2 where
> SHA256(m1) == SHA256(m2) hash value will result a single Falcon or Dilithium
> signature value which is simultaneously valid for both m1 and m2. This
> removes the extra collision resistance built in to the Dilithium and Falcon
> primitives and reduces it to the collision resistance strength of the underlying
> hash function. For this reason it is in general not recommended to pre-hash
> when using Dilithium or Falcon except in cases where the implementor is
> comfortable with this reduction in security.
>
> Therefore, for the purpose of interoperability of composite signatures,
> implementations MUST NOT pre-hash messages for Dilithium and Falcon. If
> pre-hashed versions of these signatures are desired, then separate signature
> algorithms will need to be defined.
>
>

--
You received this message because you are subscribed to the Google Groups "pqc-forum" group.

To unsubscribe from this group and stop receiving emails from it, send an email to pqc-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/CH0PR11MB5444B9D3A0CB6E447A2FA3E5C16A9%40CH0PR11MB5444.namprd11.prod.outlook.com.

_____

CFRG mailing list
CFRG@irtf.org
https://www.irtf.org/mailman/listinfo/cfrg

From: John Gray <john.gray@entrust.com> via Spasm <spasm-bounces@ietf.org>
To: Tadahiko Ito <tadahiko.ito.public@gmail.com>, Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>
CC: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>, Mike Ounsworth <mike.ounsworth@entrust.com>, LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, Kampanakis, Panos <kpanos@amazon.com>, sean@ssn3rd.com, bas@westerbaan.name
Subject: Re: [lamps] [EXTERNAL] Re: [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Thursday, August 18, 2022 11:30:11 AM ET
Attachments: ATT00001.txt

Thanks Tadahiko,

I read through your paper, and it covers exactly the usability issues we have come across! We were wondering if it is possible to perform the specific hashing external to the server (which could be an HSM as in your paper, or timestamp server, etc). For example, for Dilithium the mu := CRH(tr || M) and for Falcon it would be c <- HashToPoint(r || m, q, n). Your paper answers that question, it can be done for Falcon, but not Dilithium (without changing the signature output). So part of our question is whether using a regular external Hash as we do today for RSA and ECDSA (and what you call a boundary type B) somehow reduces the security and we shouldn't recommend it. We are interested in this because we are looking at defining composite pairs or triples which combine existing signature algorithms like RSAWithSHA256 and ECDSAWithSHA256 with Falcon or Dilithium. Having to change the operational paradigm for an HSM or something like a timestamping server would result in large amounts of data having to be piped across the internet for signatures (as you point out in your paper).

For our composite signature use case it brings up similar questions. We can support a mode where external hashing is done once, and then individually signed by the components (this makes it much more efficient) both internally and externally for the HSM, timestamping, code signing use-cases. However, in the case of Dilithium there would need to be two signature modes Sig = Dilithium (Message) and the other would be Sig = Dilithium (HASH (Message)). I don't think that is necessarily a bad thing as long as it is standardized and secure. Alternatively, we could support independent hashing for each component, but that gets strange if you are doing an external hash for ECDSA, but then need to send the whole data for Dilithium. We would likely have to end up supporting sending the whole data if external hashing compromises security of the PQC composites, but then it is even more inefficient as

each component would need to hash independently. You also covers this in section 4.3 your paper:

“We can construct type B cryptographic boundaries by adding one more hash function before the execution of PQC’s signature generation algorithm… This approach would improve the efficiency of lattice based digital signature schemes deployed in HSM. It would have a greater impact on Dilithium, but also be applicable to Falcon and other digital signature schemes. Two modes of PQC algorithms utilizing this approach will be able to exist, namely, a PQC algorithm without an additional hash (i.e. original PQC algorithm) and a PQC algorithm with an additional hash. If there are two modes of a digital signature scheme, then the asymmetric operation for those two modes must not be identical. The reason is that, obtaining a signature from the mode with an additional hash function would help attackers who can attack another mode which is without the additional hash function.”

So for example, Mode1 = Dilithium(Message) and Mode2 = Dilithium (HASH (Message)) where Mode1 is the original algorithm that does its own internal hashing, and Mode2 does an additional hash externally before the original algorithms internal hash. Then you are saying obtaining the signature from Mode2 would be able to attack Mode1? I don’t quite understand that part. If you could explain how such an attack works in a bit more detail it would be helpful.

I see you suggest mitigations by changing the Dilithium algorithm itself (section 4.3 of your paper). Perhaps such mitigations could be considered by the standards bodies? Otherwise switching from boundary type B (external hash then sign) to boundary type A (full message signing) will be another major hurdle for the industry, adding additional complication and with that possible bugs.

Thanks for sharing your paper with us Tadahiko and the valuable work you are doing!

John Gray

From: Spasm <spasm-bounces@ietf.org> **On Behalf Of** Tadahiko Ito
Sent: Thursday, August 18, 2022 1:12 AM
To: Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>
Cc: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>; Mike Ounsworth <Mike.Ounsworth@entrust.com>; LAMPS <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-forum@list.nist.gov>; Kampanakis, Panos <kpanos@amazon.com>; sean@ssn3rd.com; bas@westerbaan.name

Subject: [EXTERNAL] Re: [lamps] [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?

> It was my understanding that the signing procedure may need to be

> repeated several times to produce a signature, and thus pre-hashing

> would prevent the need to individually hash the input message with

> each attempt.

When we were trying to implement PQC in functional module of HSM for our use case, It was pain. I believe HSM vender will implement much better, but It may still have problem.

Hash then Sign was great that we can separate key management (and signing) function from data management (and hashing) function. It seems crypto module producer might need to implement scheduling function for data management function. I far those problems decrease efficiency, but we might need to care that.

>> (...) Assuming our understanding below is correct, a direct-sign algorithm

>> would require the entire thing to be streamed to a network HSM for signing

>> and to a TPM for verification.

Currently, I am doubting that we might not have that many protocols with direct-signing algorithm which would sign intolerable large data. For those protocol with direct-signing, I believe we can have several approaches.

1)Sign to smaller compressed data (e.g. by using CMS) instead of raw data.
It was biggest feedback I got so far, when I told about those stuff on IETF last year.
For this option, Users may need to change data structure, but If we cannot find that much direct-signing use case, it might be reasonable. Direct-signing use case holders may need to take other option.
In addition, when I ask our engineer for our use case, he said that was long recognized issue, and it might be good chance to do so.

2)Use pre-hash
Users do not need to change data structure, but we may meet interoperability challenge.

3)Separate PQC into key management function and data management function,
I tried, but I believe It was not good choice. <https://eprint.iacr.org/2020/990.pdf> (I am sorry that we have not updates that document.)

4)Ask NIST to make hash-and-sign PQC
If they make one, it would be easy. (well.. I believe we should not assume that)

Regards Tadahiko

2022年8月18日(木) 4:41 Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>:

> Thanks Mike, Scott.
>
> I've added to the github repo so we can track discussions on this topic https://github.com/jakemas/draft-massimo-pq-pkix-00/issues/23
>
> >> So it seems like the Dilithium designers explicitly want the hash to differ
> >> across repeated attempts.
> >>
>
> > Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.
> >In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...
>
> I think we are referring to different parts of the signing process here. For reference, my security consideration was referring to page 4 of the Dilithium spec that states:
> "Our full scheme in Fig. 4 also makes use of basic optimizations such as pre-hashing the message M so as to not rehash it with every signing attempt." and Figure 4 itself.
>
> It was my understanding that the signing procedure may need to be repeated several times to produce a signature, and thus pre-hashing would prevent the need to individually hash the input message with each attempt. I believe the desired differing of the hash you mentioned is within the internals of the signing procedure and not on the input message itself.
>
> >> Third, I can imagine that some applications (like TLS) will want to use non-pre-hashed versions of Dilithium and Falcon, but other applications (like code-signing) would prefer pre-

hashed versions. These are not interoperable with each other. Is NIST planning to produce algorithm definitions, OIDs, Codepoints, etc, for both versions?

>Expanding on the code-signing example: the messages to be signed can be very large; consider a several GB firmware image. Assuming our understanding below is correct, a direct-sign algorithm would require the entire thing to be streamed to a network HSM for signing and to a TPM for verification. Conversely code-signing environments often include counter-signatures from Time Stamping Authorities which protect against future discovery of collision attacks against the hash function -- as an example, Windows still accepts RSA-SHA1 signatures produced before SHA1 was deprecated. I can imagine that the code-signing community will decide that the performance gains of hash-then-sign outweigh the security loss.

>So, will NIST standardize both direct-sign and some variant of hash-then-sign for PQC signature primitives?

I do agree that there may be optimizations that users may wish to make dependent on the context, i.e., hash-then-sign vs direct-sign. It's for this reason I tried to give an overview of the security of each option in the draft, but ultimately leave that up to the user. It is a good point regarding NISTs perspective on what should be explicitly standardized here.

>> This provides strong security against pre-computed
>> collision attacks since an attacker has no a-priori knowledge of `r` and
>> provides per-key hash-domain separation of the message to be signed.

>Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Right, more details on the advantages of message binding on the PQC-forum from C. Peikert's https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/eAaiJO1qzkA/m/K66R_ftNBwAJ. It was this discussion I was trying to encompass in the draft.

Cheers,
Jake

On 17/08/2022, 10:51, "'Scott Fluhrer (sfluhrer)' via pqc-forum" <pqc-forum@list.nist.gov> wrote:

CAUTION: This email originated from outside of the organization. Do not click links or open attachments unless you can confirm the sender and know the content is safe.

> -----Original Message-----
> From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
> Sent: Wednesday, August 17, 2022 1:27 PM
> To: 'LAMPS' <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-
> forum@list.nist.gov>; jakemas@amazon.com; kpanos@amazon.com;
> sean@ssn3rd.com; bas@westerbaan.name
> Subject: [pqc-forum] Whether to hash-then-sign with Dilithium and Falcon?
>
> Hi Jake, Panos, Sean, Bas,
>
>
> We notice that your IETF draft-massimo-lamps-pq-sig-certificates-00 has the
> following security consideration:
>
> > Within the hash-then-sign paradigm, hash functions are used as a
> > domain restrictor over the message to be signed. By pre-hashing, the
> > onus of resistance to existential forgeries becomes heavily reliant on
> > the collision-resistance of the hash function in use. As well as this security
> goal, the hash-then-sign paradigm also has the ability to improve
> performance by reducing the size of signed messages. As a corollary, hashing
> remains mandatory even for short messages and assigns a further
> computational requirement onto the verifier. This makes the performance of
> hash-then-sign schemes more consistent, but not necessarily more efficient.
> > Dilithium diverges from the hash-then-sign paradigm by hashing the
> message during the signing procedure (at the point in which the challenge
> polynomial).
> > However, due to the fact that Dilithium signatures may require the
> > signing procedure to be repeated several times for a signature to be

> produced, Dilithium implementations can make use of pre-hashing the
> message to prevent rehashing with each attempt.
>
>
> First, quoting from the Dilithium NIST Round 3 submission documents:
>
> > Since our signing procedure may need to be repeated several times
> > until a signature is produced, we also append a counter in order to
> > make the SHAKE-256 output differ with each signing attempt of the same
> message.
>
> So it seems like the Dilithium designers explicitly want the hash to differ
> across repeated attempts.
>

Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.

In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...

>
>
> Second, we had a similar discussion within the context of composite
> signatures when figuring out how to combine Dilithium and Falcon with
> ECDSA and RSA. We came out with a different conclusion; that hash-then-
> sign reduces the security properties of Dilithium and Falcon down to the
> collision resistance of the hash function used to pre-hash.
>
> We would like community opinion on this.
>
>
> Here's the Security Consideration text that we're working on:
>
>
>

>
> In the hash-then-sign paradigm, the message to be signed is hashed
> externally to the signature primitive, and then the hash value is signed.
>
> The hash-then-sign paradigm is required, for example, with RSA signatures in
> order to sign messages larger than the RSA modulus. Hash-then-sign also
> gives performance and bandwidth benefits, for example, when the signature
> is performed by a networked cryptographic appliance since you only need to
> send a small hash value rather than streaming the entire message.
>
> With Dilithium and Falcon signatures it is not recommended to pre-hash for
> the following reasons:
>
>
> The Dilithium construction includes
>
> ~~~
> Sign(sk,M):
> 10: mu \in {0, 1}^384 := CRH(tr || M)
> ~~~
>
> where `CRH` is any collision-resistant hash function and `tr` is a component
> of the secret key.

A hash of the public key, actually; see line 7 of the key generation process (which explicitly computes it from the components of the public key) - Dilithium stores it in the private key so the signer doesn't need to recompute it every time.

> This provides strong security against pre-computed
> collision attacks since an attacker has no a-priori knowledge of `r` and
> provides per-key hash-domain separation of the message to be signed.

Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Now, it does probably add a constant factor to any attack that searches for a simultaneous collision between the hash that RSA/ECDSA uses (without the prepend) and the hash that

Dilithium uses (with the known prepend) - I would hesitate to give a value to that constant factor, but it is likely not large.

>
>
> The Falcon construction includes
>
> ~~~
> Sign (m, sk, beta^2):
> 1: r <- {0, 1}^320 uniformly
> 2: c <- HashToPoint(r || m, q, n)
> ~~~
>
> where `HashToPoint` is a SHAKE-256-based construct. This provides strong
> security against pre-computed collision attacks since an attacker has no a-
> priori knowledge of `r` and provides per-signature hash-domain separation
> of the message to be signed.
>
> If the message to be signed is pre-hashed, for example `m0 = SHA256(m)`
> and then m0 provided to Dilithium or Falcon to sign, then you have re-
> introduced the collision problem since two messages m1 and m2 where
> SHA256(m1) == SHA256(m2) hash value will result a single Falcon or Dilithium
> signature value which is simultaneously valid for both m1 and m2. This
> removes the extra collision resistance built in to the Dilithium and Falcon
> primitives and reduces it to the collision resistance strength of the underlying
> hash function. For this reason it is in general not recommended to pre-hash
> when using Dilithium or Falcon except in cases where the implementor is
> comfortable with this reduction in security.
>
> Therefore, for the purpose of interoperability of composite signatures,
> implementations MUST NOT pre-hash messages for Dilithium and Falcon. If
> pre-hashed versions of these signatures are desired, then separate signature
> algorithms will need to be defined.
>
>

--

> You received this message because you are subscribed to the Google Groups "pqc-forum" group.
> To unsubscribe from this group and stop receiving emails from it, send an email to pqc-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/CH0PR11MB5444B9D3A0CB6E447A2FA3E5C16A9%40CH0PR11MB5444.namprd11.prod.outlook.com.
>
> _____
> CFRG mailing list
> CFRG@irtf.org
> https://www.irtf.org/mailman/listinfo/cfrg

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: John Gray <john.gray=40entrust.com@dmarc.ietf.org>
CC: Tadahiko Ito <tadahiko.ito.public@gmail.com>, Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>, Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>, Mike Ounsworth <mike.ounsworth@entrust.com>, LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, Kampanakis, Panos <kpanos@amazon.com>, sean@ssn3rd.com, bas@westerbaan.name
Subject: Re: [lamps] [EXTERNAL] Re: [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Thursday, August 18, 2022 11:29:39 PM ET

On Thu, Aug 18, 2022 at 11:29 AM John Gray <John.Gray=40entrust.com@dmarc.ietf.org> wrote:

> Thanks Tadahiko,
>
> I read through your paper, and it covers exactly the usability issues we have come across! We were wondering if it is possible to perform the specific hashing external to the server (which could be an HSM as in your paper, or timestamp server, etc). For example, for Dilithium the mu := CRH(tr || M) and for Falcon it would be c <- HashToPoint(r || m, q, n). Your paper answers that question, it can be done for Falcon, but not Dilithium (without changing the signature output). So part of our question is whether using a regular external Hash as we do today for RSA and ECDSA (and what you call a boundary type B) somehow reduces the security and we shouldn't recommend it. We are interested in this because we are looking at defining composite pairs or triples which combine existing signature algorithms like RSAWithSHA256 and ECDSAWithSHA256 with Falcon or Dilithium. Having to change the operational paradigm for an HSM or something like a timestamping server would result in large amounts of data having to be piped across the internet for signatures (as you point out in your paper).
>
> For our composite signature use case it brings up similar questions. We can support a mode where external hashing is done once, and then individually signed by the components (this makes it much more efficient) both internally and externally for the HSM, timestamping, code signing use-cases. However, in the case of Dilithium there would need to be two signature modes Sig = Dilithium (Message) and the other would be Sig = Dilithium (HASH (Message)). I don't think that is necessarily a bad thing as long as it is standardized and secure. Alternatively, we could support independent hashing for each component, but that gets strange if you are doing an external hash for ECDSA, but then need to send the whole

> data for Dilithium. We would likely have to end up supporting sending the whole data if external hashing compromises security of the PQC composites, but then it is even more inefficient as each component would need to hash independently. You also covers this in section 4.3 your paper:
>
> “We can construct type B cryptographic boundaries by adding one more hash function before the execution of PQC’s signature generation algorithm... This approach would improve the efficiency of lattice based digital signature schemes deployed in HSM. It would have a greater impact on Dilithium, but also be applicable to Falcon and other digital signature schemes. Two modes of PQC algorithms utilizing this approach will be able to exist, namely, a PQC algorithm without an additional hash (i.e. original PQC algorithm) and a PQC algorithm with an additional hash. If there are two modes of a digital signature scheme, then the asymmetric operation for those two modes must not be identical. The reason is that, obtaining a signature from the mode with an additional hash function would help attackers who can attack another mode which is without the additional hash function.”

That is not a new issue and it is one that we discussed at great length for RSA. The hash of the digest is part of the signature payload to prevent substitution attacks.

I am finding the description of the problem here to be making assumptions about what 'formal proofs' are demonstrating that are probably demonstrating something else. If you assume there is only one true digest function, you 'solve' the digest substitution problem by pretending it does not exist...

In my view, the signature function should be over the payload and the additional authenticated data. Whether this is the OID of the digest algorithm alone or something more depends on the signature scheme. That is what JOSE etc. have to sort out.

Preventing a digest downgrade attack may be considered the concern of the cryptographic envelope format because there may be more than one form of downgrade attack to be considered. But it is also something we have traditionally included in the signature...

--

From: Mike Ounsworth <mike.ounsworth@entrust.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Phillip Hallam-Baker <phill@hallambaker.com>, John Gray <john.gray=40entrust.com@dmarc.ietf.org>
CC: Tadahiko Ito <tadahiko.ito.public@gmail.com>, Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>, Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>, LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>, Kampanakis, Panos <kpanos@amazon.com>, sean@ssn3rd.com, bas@westerbaan.name
Subject: RE: [lamps] [EXTERNAL] Re: [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Friday, August 19, 2022 01:56:22 PM ET

Phillip,

> That is not a new issue and it is one that we discussed at great length for RSA. The hash of the digest is part of the signature payload to prevent substitution attacks.

I think you are in fact mistaking the problem that we're trying to solve / trying not to un-solve. Nobody is suggesting hash downgrade or substitution attacks. As you point out, that is already solved by existing protocols.

Taking Falcon as an example, the problem as I understand it is that the first internal step of Falcon is:

Sign(m):

r = rand(320);

c = SHAKE(r || m);

...

output (r, s)

Verify(r, s, m):

c = SHAKE(r || m);

this has the nice property that `r` is generated at signing time by the signer. So should SHAKE develop a collision attack similar to the one we saw for SHA1, then the collision pre-computation as still intractable for the attacker because the attacker cannot not know `r` in advance. (Disclaimer: I am just an engineer trying to wrap my head around this, corrections welcome!).

So the concern with hash-then-sign is that if `m` above is itself a message digest, ie `m = SHA256(M)`, then you have un-done the collision resistance that is built in to the Falcon primitive, essentially reducing the security properties of Falcon. Adding the message digest to the signed content does not help if the attacker is holding two messages with the same SHA256 hash.

My simplified-to-the-point-of-being-wrong understanding is that this would be no worse than what we do with RSA and therefore might be acceptable in some use cases, but it is also ignoring cryptographic improvements beyond RSA.

Disclaimer: this whole email should be viewed as a question that we are seeking confirmation of.

Mike Ounsworth

From: Phillip Hallam-Baker <phill@hallambaker.com>
Sent: August 18, 2022 10:29 PM
To: John Gray <John.Gray=40entrust.com@dmarc.ietf.org>
Cc: Tadahiko Ito <tadahiko.ito.public@gmail.com>; Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>; Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>; Mike Ounsworth <Mike.Ounsworth@entrust.com>; LAMPS <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-forum@list.nist.gov>; Kampanakis, Panos <kpanos@amazon.com>; sean@ssn3rd.com; bas@westerbaan.name
Subject: Re: [lamps] [EXTERNAL] Re: [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?

On Thu, Aug 18, 2022 at 11:29 AM John Gray <John.Gray=40entrust.com@dmarc.ietf.org> wrote:

> Thanks Tadahiko,
>
> I read through your paper, and it covers exactly the usability issues we have come across! We were wondering if it is possible to perform the specific hashing external to the server (which could be an HSM as in your paper, or timestamp server, etc). For example, for Dilithium the mu := CRH(tr || M) and for Falcon it would be c <- HashToPoint(r || m, q, n). Your paper answers that question, it can be done for Falcon, but not Dilithium (without changing the signature output). So part of our question is whether using a regular external Hash as we do today for RSA and ECDSA (and what you call a boundary type B) somehow

> reduces the security and we shouldn't recommend it. We are interested in this because we are looking at defining composite pairs or triples which combine existing signature algorithms like RSAWithSHA256 and ECDSAWithSHA256 with Falcon or Dilithium. Having to change the operational paradigm for an HSM or something like a timestamping server would result in large amounts of data having to be piped across the internet for signatures (as you point out in your paper).
>
> For our composite signature use case it brings up similar questions. We can support a mode where external hashing is done once, and then individually signed by the components (this makes it much more efficient) both internally and externally for the HSM, timestamping, code signing use-cases. However, in the case of Dilithium there would need to be two signature modes Sig = Dilithium (Message) and the other would be Sig = Dilithium (HASH (Message)). I don't think that is necessarily a bad thing as long as it is standardized and secure. Alternatively, we could support independent hashing for each component, but that gets strange if you are doing an external hash for ECDSA, but then need to send the whole data for Dilithium. We would likely have to end up supporting sending the whole data if external hashing compromises security of the PQC composites, but then it is even more inefficient as each component would need to hash independently. You also covers this in section 4.3 your paper:
>
> "We can construct type B cryptographic boundaries by adding one more hash function before the execution of PQC's signature generation algorithm… This approach would improve the efficiency of lattice based digital signature schemes deployed in HSM. It would have a greater impact on Dilithium, but also be applicable to Falcon and other digital signature schemes. Two modes of PQC algorithms utilizing this approach will be able to exist, namely, a PQC algorithm without an additional hash (i.e. original PQC algorithm) and a PQC algorithm with an additional hash. If there are two modes of a digital signature scheme, then the asymmetric operation for those two modes must not be identical. The reason is that, obtaining a signature from the mode with an additional hash function would help attackers who can attack another mode which is without the additional hash function."

That is not a new issue and it is one that we discussed at great length for RSA. The hash of the digest is part of the signature payload to prevent substitution attacks.

I am finding the description of the problem here to be making assumptions about what 'formal proofs' are demonstrating that are probably demonstrating something else. If you assume there is only one true digest function, you 'solve' the digest substitution problem by pretending it does not exist…

In my view, the signature function should be over the payload and the additional authenticated data. Whether this is the OID of the digest algorithm alone or something more depends on the signature scheme. That is what JOSE etc. have to sort out.

Preventing a digest downgrade attack may be considered the concern of the cryptographic envelope format because there may be more than one form of downgrade attack to be considered. But it is also something we have traditionally included in the signature…

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: Mike Ounsworth <mike.ounsworth@entrust.com>
CC: LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>
Subject: Re: [lamps] [EXTERNAL] Re: [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Friday, August 19, 2022 03:44:45 PM ET

On Fri, Aug 19, 2022 at 1:55 PM Mike Ounsworth <Mike.Ounsworth@entrust.com> wrote:

> Phillip,
>
> > That is not a new issue and it is one that we discussed at great length for RSA. The hash of the digest is part of the signature payload to prevent substitution attacks.
>
> I think you are in fact mistaking the problem that we're trying to solve / trying not to un-solve. Nobody is suggesting hash downgrade or substitution attacks. As you point out, that is already solved by existing protocols.
>
> Taking Falcon as an example, the problem as I understand it is that the first internal step of Falcon is:
>
> Sign(m):
>
> r = rand(320);
>
> c = SHAKE(r || m);
>
> ...
>
> output (r, s)
>
> Verify(r, s, m):
>
> c = SHAKE(r || m);

Oh right. So far I am up the Kyber pass and have not done Dilithium yet.

> this has the nice property that `r` is generated at signing time by the signer. So should SHAKE develop a collision attack similar to the one we saw for SHA1, then the collision pre-computation as still intractable for the attacker because the attacker cannot not know `r` in

> advance. (Disclaimer: I am just an engineer trying to wrap my head around this, corrections welcome!).

Meh, seems like an unnecessary concern. And I am not sure how this actually helps against collision attacks in general.

If SHAKE is broken, then we are going to have so many problems that the security of the signature algorithm is irrelevant.

As a matter of protocol composition, the signature algorithm should not be trying to fix a potential issue in the digest algorithm because things are going to be breaking all over.

> So the concern with hash-then-sign is that if `m` above is itself a message digest, ie `m = SHA256(M)`, then you have un-done the collision resistance that is built in to the Falcon primitive, essentially reducing the security properties of Falcon. Adding the message digest to the signed content does not help if the attacker is holding two messages with the same SHA256 hash.

If this is a useful approach, then move it out of the signature primitive and push it into the envelope format.

> My simplified-to-the-point-of-being-wrong understanding is that this would be no worse than what we do with RSA and therefore might be acceptable in some use cases, but it is also ignoring cryptographic improvements beyond RSA.
>
> Disclaimer: this whole email should be viewed as a question that we are seeking confirmation of.
>
> ---
>
> **Mike** Ounsworth
>
> ---
>
> **From:** Phillip Hallam-Baker <phill@hallambaker.com>
> **Sent:** August 18, 2022 10:29 PM
> **To:** John Gray <John.Gray=40entrust.com@dmarc.ietf.org>
> **Cc:** Tadahiko Ito <tadahiko.ito.public@gmail.com>; Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>; Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>; Mike Ounsworth <Mike.Ounsworth@entrust.com>; LAMPS <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-forum@list.nist.gov>; Kampanakis, Panos <kpanos@amazon.com>; sean@ssn3rd.com; bas@westerbaan.name
> **Subject:** Re: [lamps] [EXTERNAL] Re: [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?

On Thu, Aug 18, 2022 at 11:29 AM John Gray <John.Gray=40entrust.com@dmarc.ietf.org> wrote:

> Thanks Tadahiko,
>
> I read through your paper, and it covers exactly the usability issues we have come across! We were wondering if it is possible to perform the specific hashing external to the server (which could be an HSM as in your paper, or timestamp server, etc). For example, for Dilithium the mu := CRH(tr || M) and for Falcon it would be c <- HashToPoint(r || m, q, n). Your paper answers that question, it can be done for Falcon, but not Dilithium (without changing the signature output). So part of our question is whether using a regular external Hash as we do today for RSA and ECDSA (and what you call a boundary type B) somehow reduces the security and we shouldn't recommend it. We are interested in this because we are looking at defining composite pairs or triples which combine existing signature algorithms like RSAWithSHA256 and ECDSAWithSHA256 with Falcon or Dilithium. Having to change the operational paradigm for an HSM or something like a timestamping server would result in large amounts of data having to be piped across the internet for signatures (as you point out in your paper).
>
> For our composite signature use case it brings up similar questions. We can support a mode where external hashing is done once, and then individually signed by the components (this makes it much more efficient) both internally and externally for the HSM, timestamping, code signing use-cases. However, in the case of Dilithium there would need to be two signature modes Sig = Dilithium (Message) and the other would be Sig = Dilithium (HASH (Message)). I don't think that is necessarily a bad thing as long as it is standardized and secure. Alternatively, we could support independent hashing for each component, but that gets strange if you are doing an external hash for ECDSA, but then need to send the whole data for Dilithium. We would likely have to end up supporting sending the whole data if external hashing compromises security of the PQC composites, but then it is even more inefficient as each component would need to hash independently. You also covers this in section 4.3 your paper:
>
> "We can construct type B cryptographic boundaries by adding one more hash function before the execution of PQC's signature generation algorithm... This approach would improve the efficiency of lattice based digital signature schemes deployed in HSM. It would have a greater impact on Dilithium, but also be applicable to Falcon and other digital signature schemes. Two modes of PQC algorithms utilizing this approach will be able to exist, namely, a PQC algorithm without an additional hash (i.e. original PQC algorithm) and a PQC algorithm with an additional hash. If there are two modes of a

> digital signature scheme, then the asymmetric operation for those two modes must not be identical. The reason is that, obtaining a signature from the mode with an additional hash function would help attackers who can attack another mode which is without the additional hash function."

That is not a new issue and it is one that we discussed at great length for RSA. The hash of the digest is part of the signature payload to prevent substitution attacks.

I am finding the description of the problem here to be making assumptions about what 'formal proofs' are demonstrating that are probably demonstrating something else. If you assume there is only one true digest function, you 'solve' the digest substitution problem by pretending it does not exist...

In my view, the signature function should be over the payload and the additional authenticated data. Whether this is the OID of the digest algorithm alone or something more depends on the signature scheme. That is what JOSE etc. have to sort out.

Preventing a digest downgrade attack may be considered the concern of the cryptographic envelope format because there may be more than one form of downgrade attack to be considered. But it is also something we have traditionally included in the signature...

--

From: D. J. Bernstein <djb@cr.yp.to> via pqc-forum@list.nist.gov
To: pqc-forum@list.nist.gov, spasm@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] Re: Whether to hash-then-sign with Dilithium and Falcon?
Date: Saturday, August 20, 2022 02:30:31 PM ET

Phillip Hallam-Baker writes:
> If this is a useful approach, then move it out of the signature
> primitive and push it into the envelope format.

Sure, people converting a signature system m ⟼ Sign(m) into a one-pass signature system m ⟼ Sign(H(m)) can maybe be convinced to instead convert it into a safer one-pass signature system m ⟼ r,Sign(H(r,m)) where r is chosen randomly at signing time. Both conversions are generic, and the performance differences are minor.

But moving this _out_ of the underlying signature system is dangerous. Applications will often expose the underlying signature system directly to attackers. For example, an RSA HSM that returns h^d given h, trusting the environment to choose h as a hash, is breakable by essentially the attack of https://link.springer.com/article/10.1007/s00145-015-9205-5.

> If SHAKE is broken, then we are going to have so many problems that the
> security of the signature algorithm is irrelevant.

Certainly we'd like to end up in a world where everyone is using a hash function that we can confidently assume is collision-resistant. However, avoiding this assumption turns out to simplify security review:

* Avoiding collision resistance means that cryptanalysts can avoid worrying about some important, hard-to-analyze attack avenues (e.g., what Wang dubbed "message modification").

* Some components of security analyses can be computer-verified today, but (because of well-known formalization difficulties) assuming collision resistance creates problems for this.

* In the opposite direction, the random r above creates its own

> complications in security review, but looking at the details shows
> that these complications are _relatively_ easy to analyze.

Simplifying security review is important because the overall situation in post-quantum cryptography is that security reviewers are terribly overloaded. Compare, e.g., https://gcc02.safelinks.protection.outlook.com/?url=https%3A%2F%2Feprint.iacr.org%2F2021%2F543&data=05%7C01%7Cyi-kai.liu%40nist.gov%7C5395515a1400445d1e7e08da82da0e27%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C637966170308472940%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=SaOvjmCts7bxkgfWYZAZDJIbhT7dg1aqOJM1%2Bospzi8%3D&reserved=0 ("A decade unscathed") to https://gcc02.safelinks.protection.outlook.com/?url=https%3A%2F%2Feprint.iacr.org%2F2022%2F975&data=05%7C01%7Cyi-kai.liu%40nist.gov%7C5395515a1400445d1e7e08da82da0e27%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C637966170308472940%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=INjAs2w4ZW9TYsa33Hsd7JLNTgJjednr7BBjK9Q7ERA%3D&reserved=0. For many more examples see https://gcc02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fntruprime.cr.yp.to%2Flatticerisks-20211031.pdf&data=05%7C01%7Cyi-kai.liu%40nist.gov%7C5395515a1400445d1e7e08da82da0e27%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C637966170308472940%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=0QYhvqU6D1hioYE%2B5xqcFWBlaMex2Q6fFhgKvY%2Fgg%2F8%3D&reserved=0.

If a cryptosystem takes 10 unnecessary risks, each independently incurring a 10% chance of disaster, then the cryptosystem has, overall, a 65% chance (1-0.9^10) of disaster from these risks. Systematically avoiding risks is much safer.

——D. J. Bernstein

From: Tadahiko Ito <tadahiko.ito.public@gmail.com> via Spasm <spasm-bounces@ietf.org>
To: John Gray <john.gray@entrust.com>
CC: LAMPS <spasm@ietf.org>, cfrg@irtf.org, pqc-forum <pqc-forum@list.nist.gov>
Subject: Re: [lamps] [EXTERNAL] Re: [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?
Date: Monday, August 22, 2022 01:43:43 PM ET
Attachments: ATT00001.txt

I was replying to an individual by mistake..

Hi John

I believe timestamp is good example of separating data control and key management, I am not sure if it would be like

> timestamping server would result in large amounts of data having to be piped across the internet for signatures

that.

I might be misunderstanding,,, but at least for RFC3161 timestamp,

to timestamp message M, timestamp client first calculate requesing-hash value (let say Hash(M)) and send to timestamp server.

Upon timestamp server receive Hash(M), he concatenate some value, hash-then-sign, and return. returning value would be something like... hash-then-sign(Hash(M)||TIME||etc...).

Data on Internet should be not that big. (data between timestamp server and (internal) HSM should be small also).

I was just thinking that, above hash functions in "Hash()" and "hash-then-sign" can be different one, and may introduce additional complexity, on migration.

Regards Tadahiko Ito

2022年8月19日(金) 0:03 John Gray <John.Gray@entrust.com>:

> Thanks Tadahiko,

I read through your paper, and it covers exactly the usability issues we have come across! We were wondering if it is possible to perform the specific hashing external to the server (which could be an HSM as in your paper, or timestamp server, etc). For example, for Dilithium the mu := CRH(tr || M) and for Falcon it would be c <- HashToPoint(r || m, q, n). Your paper answers that question, it can be done for Falcon, but not Dilithium (without changing the signature output). So part of our question is whether using a regular external Hash as we do today for RSA and ECDSA (and what you call a boundary type B) somehow reduces the security and we shouldn't recommend it. We are interested in this because we are looking at defining composite pairs or triples which combine existing signature algorithms like RSAWithSHA256 and ECDSAWithSHA256 with Falcon or Dilithium. Having to change the operational paradigm for an HSM or something like a timestamping server would result in large amounts of data having to be piped across the internet for signatures (as you point out in your paper).

For our composite signature use case it brings up similar questions. We can support a mode where external hashing is done once, and then individually signed by the components (this makes it much more efficient) both internally and externally for the HSM, timestamping, code signing use-cases. However, in the case of Dilithium there would need to be two signature modes Sig = Dilithium (Message) and the other would be Sig = Dilithium (HASH (Message)). I don't think that is necessarily a bad thing as long as it is standardized and secure. Alternatively, we could support independent hashing for each component, but that gets strange if you are doing an external hash for ECDSA, but then need to send the whole data for Dilithium. We would likely have to end up supporting sending the whole data if external hashing compromises security of the PQC composites, but then it is even more inefficient as each component would need to hash independently. You also covers this in section 4.3 your paper:

"We can construct type B cryptographic boundaries by adding one more hash function before the execution of PQC's signature generation algorithm... This approach would improve the efficiency of lattice based digital signature schemes deployed in HSM. It would have a greater impact on Dilithium, but also be applicable to Falcon and other digital signature schemes. Two modes of PQC algorithms utilizing this approach will be able to exist, namely, a PQC algorithm without an additional hash (i.e. original PQC algorithm) and a PQC algorithm with an additional hash. If there are two modes of a digital signature scheme, then the asymmetric operation for those two modes must not be identical. The reason is that, obtaining a signature from the mode with an additional hash function would help attackers who can attack another mode which is without the additional hash function."

So for example, Mode1 = Dilithium(Message) and Mode2 = Dilithium (HASH (Message)) where Mode1 is the original algorithm that does its own internal hashing, and Mode2 does an additional hash externally before the original algorithms internal hash. Then you are saying obtaining the signature from Mode2 would be able to attack Mode1? I don't quite understand that part. If you could explain how such an attack works in a bit more detail it would be helpful.

I see you suggest mitigations by changing the Dilithium algorithm itself (section 4.3 of your paper). Perhaps such mitigations could be considered by the standards bodies? Otherwise switching from boundary type B (external hash then sign) to boundary type A (full message signing) will be another major hurdle for the industry, adding additional complication and with that possible bugs.

Thanks for sharing your paper with us Tadahiko and the valuable work you are doing!

John Gray

From: Spasm <spasm-bounces@ietf.org> **On Behalf Of** Tadahiko Ito
Sent: Thursday, August 18, 2022 1:12 AM
To: Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>
Cc: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>; Mike Ounsworth <Mike.Ounsworth@entrust.com>; LAMPS <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-forum@list.nist.gov>; Kampanakis, Panos <kpanos@amazon.com>; sean@ssn3rd.com; bas@westerbaan.name
Subject: [EXTERNAL] Re: [lamps] [CFRG] [pqc-forum] RE: Whether to hash-then-sign with Dilithium and Falcon?

WARNING: This email originated outside of Entrust.
DO NOT CLICK links or attachments unless you trust the sender and know the content is safe.

> It was my understanding that the signing procedure may need to be

> repeated several times to produce a signature, and thus pre-hashing

> would prevent the need to individually hash the input message with

> each attempt.

When we were trying to implement PQC in functional module of HSM for our use case, It was pain. I believe HSM vender will implement much better, but It may still have problem.

Hash then Sign was great that we can separate key management (and signing) function from data management (and hashing) function. It seems crypto module producer might need to implement scheduling function for data management function. I far those problems decrease efficiency, but we might need to care that.

>> (...) Assuming our understanding below is correct, a direct-sign algorithm

>> would require the entire thing to be streamed to a network HSM for signing

>> and to a TPM for verification.

Currently, I am doubting that we might not have that many protocols with direct-signing algorithm which would sign intolerable large data. For those protocol with direct-signing, I believe we can have several approaches.

1)Sign to smaller compressed data (e.g. by using CMS) instead of raw data.
It was biggest feedback I got so far, when I told about those stuff on IETF last year.
For this option, Users may need to change data structure, but If we cannot find that much direct-signing use case, it might be reasonable. Direct-signing use case holders may need to take other option.
In addition, when I ask our engineer for our use case, he said that was long recognized issue, and it might be good chance to do so.

2)Use pre-hash
Users do not need to change data structure, but we may meet interoperability challenge.

3)Separate PQC into key management function and data management function,
I tried, but I believe It was not good choice. <https://eprint.iacr.org/2020/990.pdf> (I am sorry that we have not updates that document.)

4)Ask NIST to make hash-and-sign PQC
If they make one, it would be easy. (well.. I believe we should not assume that)

Regards Tadahiko

2022年8月18日(木) 4:41 Massimo, Jake <jakemas=40amazon.com@dmarc.ietf.org>:

> Thanks Mike, Scott.
>
> I've added to the github repo so we can track discussions on this topic https://github.com/jakemas/draft-massimo-pq-pkix-00/issues/23

>> So it seems like the Dilithium designers explicitly want the hash to differ
>> across repeated attempts.
>>

> Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.
>In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...

I think we are referring to different parts of the signing process here. For reference, my security consideration was referring to page 4 of the Dilithium spec that states:
"Our full scheme in Fig. 4 also makes use of basic optimizations such as pre-hashing the message M so as to not rehash it with every signing attempt." and Figure 4 itself.

It was my understanding that the signing procedure may need to be repeated several times to produce a signature, and thus pre-hashing would prevent the need to individually hash the input message with each attempt. I believe the desired differing of the hash you mentioned is within the internals of the signing procedure and not on the input message itself.

>> Third, I can imagine that some applications (like TLS) will want to use non-pre-hashed versions of Dilithium and Falcon, but other applications (like code-signing) would prefer pre-hashed versions. These are not interoperable with each other. Is NIST planning to produce algorithm definitions, OIDs, Codepoints, etc, for both versions?

>Expanding on the code-signing example: the messages to be signed can be very large; consider a several GB firmware image. Assuming our understanding below is correct, a direct-sign algorithm would require the entire thing to be streamed to a network HSM for signing and to a TPM for verification. Conversely code-signing environments often include counter-signatures from Time Stamping Authorities which protect against future discovery of collision attacks against the hash function -- as an example, Windows still accepts RSA-SHA1 signatures produced before SHA1 was deprecated. I can imagine that the code-signing community will decide that the performance gains of hash-then-sign outweigh the security loss.

>So, will NIST standardize both direct-sign and some variant of hash-then-sign for PQC

signature primitives?

I do agree that there may be optimizations that users may wish to make dependent on the context, i.e., hash-then-sign vs direct-sign. It's for this reason I tried to give an overview of the security of each option in the draft, but ultimately leave that up to the user. It is a good point regarding NISTs perspective on what should be explicitly standardized here.

>> This provides strong security against pre-computed
>> collision attacks since an attacker has no a-priori knowledge of `r` and
>> provides per-key hash-domain separation of the message to be signed.

>Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Right, more details on the advantages of message binding on the PQC-forum from C. Peikert's https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/eAaiJO1qzkA/m/K66R_ftNBwAJ. It was this discussion I was trying to encompass in the draft.

Cheers,
Jake

On 17/08/2022, 10:51, "'Scott Fluhrer (sfluhrer)' via pqc-forum" <pqc-forum@list.nist.gov> wrote:

CAUTION: This email originated from outside of the organization. Do not click links or open attachments unless you can confirm the sender and know the content is safe.

> -----Original Message-----
> From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
> Sent: Wednesday, August 17, 2022 1:27 PM
> To: 'LAMPS' <spasm@ietf.org>; cfrg@irtf.org; pqc-forum <pqc-

> forum@list.nist.gov>; jakemas@amazon.com; kpanos@amazon.com;
> sean@ssn3rd.com; bas@westerbaan.name
> Subject: [pqc-forum] Whether to hash-then-sign with Dilithium and Falcon?
>
> Hi Jake, Panos, Sean, Bas,
>
>
> We notice that your IETF draft-massimo-lamps-pq-sig-certificates-00 has the
> following security consideration:
>
> > Within the hash-then-sign paradigm, hash functions are used as a
> > domain restrictor over the message to be signed. By pre-hashing, the
> > onus of resistance to existential forgeries becomes heavily reliant on
> > the collision-resistance of the hash function in use. As well as this security
> goal, the hash-then-sign paradigm also has the ability to improve
> performance by reducing the size of signed messages. As a corollary, hashing
> remains mandatory even for short messages and assigns a further
> computational requirement onto the verifier. This makes the performance of
> hash-then-sign schemes more consistent, but not necessarily more efficient.
> > Dilithium diverges from the hash-then-sign paradigm by hashing the
> message during the signing procedure (at the point in which the challenge
> polynomial).
> > However, due to the fact that Dilithium signatures may require the
> > signing procedure to be repeated several times for a signature to be
> produced, Dilithium implementations can make use of pre-hashing the
> message to prevent rehashing with each attempt.
>
>
> First, quoting from the Dilithium NIST Round 3 submission documents:
>
> > Since our signing procedure may need to be repeated several times
> > until a signature is produced, we also append a counter in order to
> > make the SHAKE-256 output differ with each signing attempt of the same
> message.
>
> So it seems like the Dilithium designers explicitly want the hash to differ
> across repeated attempts.

>

Hmmm, I don't see that in Dilithium; are they referring to the internal ExpandMask function? That isn't applied to the input message.

In any case, it's easy to derive SHAKE(M || 1), SHAKE(M || 2), ... without multiple passes through M; you compute the partial SHAKE state after process M, and then apply that partial state to 1, 2, ...

>
>
> Second, we had a similar discussion within the context of composite
> signatures when figuring out how to combine Dilithium and Falcon with
> ECDSA and RSA. We came out with a different conclusion; that hash-then-
> sign reduces the security properties of Dilithium and Falcon down to the
> collision resistance of the hash function used to pre-hash.
>
> We would like community opinion on this.
>
>
> Here's the Security Consideration text that we're working on:
>
>
>
>
> In the hash-then-sign paradigm, the message to be signed is hashed
> externally to the signature primitive, and then the hash value is signed.
>
> The hash-then-sign paradigm is required, for example, with RSA signatures in
> order to sign messages larger than the RSA modulus. Hash-then-sign also
> gives performance and bandwidth benefits, for example, when the signature
> is performed by a networked cryptographic appliance since you only need to
> send a small hash value rather than streaming the entire message.
>
> With Dilithium and Falcon signatures it is not recommended to pre-hash for
> the following reasons:
>

>
> The Dilithium construction includes
>
> ~~~
> Sign(sk,M):
> 10: mu \in {0, 1}^384 := CRH(tr || M)
> ~~~
>
> where `CRH` is any collision-resistant hash function and `tr` is a component
> of the secret key.

A hash of the public key, actually; see line 7 of the key generation process (which explicitly computes it from the components of the public key) - Dilithium stores it in the private key so the signer doesn't need to recompute it every time.

> This provides strong security against pre-computed
> collision attacks since an attacker has no a-priori knowledge of `r` and
> provides per-key hash-domain separation of the message to be signed.

Rather, it limits the usability of any found collision to a specific public key; however it does nothing to frustrate a collision attack against a specific public key.

Now, it does probably add a constant factor to any attack that searches for a simultaneous collision between the hash that RSA/ECDSA uses (without the prepend) and the hash that Dilithium uses (with the known prepend) - I would hesitate to give a value to that constant factor, but it is likely not large.

>
>
> The Falcon construction includes
>
> ~~~
> Sign (m, sk, beta^2):
> 1: r <- {0, 1}^320 uniformly
> 2: c <- HashToPoint(r || m, q, n)
> ~~~
>

> where `HashToPoint` is a SHAKE-256-based construct. This provides strong
> security against pre-computed collision attacks since an attacker has no a-
> priori knowledge of `r` and provides per-signature hash-domain separation
> of the message to be signed.
>
> If the message to be signed is pre-hashed, for example `m0 = SHA256(m)`
> and then m0 provided to Dilithium or Falcon to sign, then you have re-
> introduced the collision problem since two messages m1 and m2 where
> SHA256(m1) == SHA256(m2) hash value will result a single Falcon or Dilithium
> signature value which is simultaneously valid for both m1 and m2. This
> removes the extra collision resistance built in to the Dilithium and Falcon
> primitives and reduces it to the collision resistance strength of the underlying
> hash function. For this reason it is in general not recommended to pre-hash
> when using Dilithium or Falcon except in cases where the implementor is
> comfortable with this reduction in security.
>
> Therefore, for the purpose of interoperability of composite signatures,
> implementations MUST NOT pre-hash messages for Dilithium and Falcon. If
> pre-hashed versions of these signatures are desired, then separate signature
> algorithms will need to be defined.
>
>

> *Any email and files/attachments transmitted with it are confidential and are intended solely for the use of the individual or entity to whom they are addressed. If this message has been sent to you in error, you must not copy, distribute or disclose of the information it contains. Please notify Entrust immediately and delete the message from your system.*