From: Mike Ounsworth <mike.ounsworth@entrust.com> via pqc-forum <pqc-forum@list.nist.gov>
To: pqc-forum <pqc-forum@list.nist.gov>
CC: pqc@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Sunday, September 18, 2022 03:43:01 PM ET

Hi NIST PQC Forum!

This is bubble-over from an IETF thread I started last week.

Context: hash-then-sign schemes are good. For example, they allow you to pre-hash your potentially very large message and then send just the hash value to your cryptographic module to sign or verify. We like this pattern, it's good for bandwidth and latency of cryptographic modules. We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number; in the case of non-randomized SPHINCS+ and Dilithium, that message digest is keyed with values derived from the public key (for completeness: randomized SPHINCS+ seems to be the only to do both).

A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).

But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

Question:

So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? Is it ok to take m' = SHA256(m) and then sign m' ? If we care about the built-in collision-resistance, then the answer is probably "No". Is it safe to externalize the keyed message digest step of SPHINCS+, Dilithium, FALCON? In the non-randomized versions where the keyed message digest only relies on values in the public key, I would think the answer is "Yes". For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a “No”.

I’m eager to hear more on the design rationale for starting with a randomized or deterministic keyed message digest, and recommendations for the safe way to external pre-hashes with these schemes.

Mike Ounsworth
Software Security Architect, Entrust

From: Christopher J Peikert <cpeikert@alum.mit.edu> via pqc-forum@list.nist.gov
To: Mike Ounsworth <mike.ounsworth@entrust.com>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Sunday, September 18, 2022 10:46:02 PM ET

On Sun, Sep 18, 2022 at 3:42 PM 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov> wrote:

> We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number…
>
> A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).

For Falcon in particular, there is a much more essential reason for "keying" the hash by a random number: releasing two different signatures for the same hash digest breaks the security reduction, and likely has very bad practical consequences as well. Note that the "core" signing procedure -- given a hash digest, output a signature -- is randomized, so different runs on the same digest will tend to produce different signatures.

Randomized hashing ensures that the same digest essentially never reoccurs (assuming quality randomness), even when signing the same message multiple times. See Section 2.2.2 of the Falcon spec for further details.

"Derandomizing" the signing procedure is another approach that may be suitable in some (but not all) scenarios. We have designed and implemented a version of this approach as a "thin wrapper" around the Falcon API; see the specification for the motivation, details, and caveats. Signatures produced using this deterministic method can easily be transformed into ones that are accepted by a verifier who expects randomized ones (but not vice versa). But due to this compatibility, our approach has the same issues as randomized Falcon when it comes to externalized hashing.

> But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

This looks right to me, though it's not clear how much of a benefit this really is. For example, Falcon's security analysis models the hash function as a "random oracle," which is an even stronger assumption than collision resistance.

> Question:
>
> So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? ... For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".

Unfortunately, the "dream" goal of externalized, deterministic, stateless hashing inherently requires collision resistance (of that external hash function): the adversary knows the function, so it can attempt to find collisions in it. Any valid signature for one of the colliding messages is inherently also a valid signature for the other colliding message.

One remark is that, when modeling the hash function as a random oracle, it is not strictly necessary that the "random" value actually be random -- it is enough that it never be *repeated* (at least not for repeated signings of the same message under the same key). In this sense, it need only be a "nonce" in the true sense of the word: a Number used only ONCE.

So, if there is a way to generate a true nonce outside the cryptographic module, perhaps that can be used. However, if the nonce is fairly predictable then we shouldn't expect to get the above-noted benefits of randomization (whatever they may be). And this is not even getting into the serious misuse risks of an API that allows the caller to set the nonce.

Given all this, to me it seems that (for Falcon at least) the best way to externalize hashing of huge data is through some extra "envelope" layer that pre-hashes the data in a suitable way, then signs the result as if it were the true message. One can then focus on the security properties of the envelope.

Sincerely yours in cryptography,

Chris

From: Simon Josefsson <simon@josefsson.org> via CFRG <cfrg-bounces@irtf.org>
To: Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 04:55:34 AM ET
Attachments: signature.asc
ATT00001.txt

Mike Ounsworth <Mike.Ounsworth=40entrust.com@dmarc.ietf.org> writes:

> Hi NIST PQC Forum!
>
> This is bubble-over from an IETF thread I started last week.
>
> Context: hash-then-sign schemes are good.

My take is that we should question your assumption.

Engineers have used hash-then-sign as a design axiom and forced that design on cryptographic primitives. Cryptographers led engineers form that thinking because historic public-key signature schemes used the hash-then-sign approach. However, cryptographic primitives has moved away from that design long time ago. Engineers are still firmly in the old thinking.

We had the same discussion related to Ed25519, and the CFRG ended up in a compromise to re-introduce the less secure hash-then-sign variant for, what I can tell, practical engineering trade-offs with existing protocols. We all know how this tends to play out in the long term.

So please, design your protocols to not assume the less secure hash-then-sign approach of public-key signing schemes.

/Simon

From:	Dmitry Belyavsky <beldmit@gmail.com> via CFRG <cfrg-bounces@irtf.org>
To:	Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>
CC:	pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject:	Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date:	Monday, September 19, 2022 07:12:10 AM ET
Attachments:	ATT00001.txt

Just 2c

FIPS 140-3 seem to forbid hashing outside the module boundary and signing inside.

On Sun, Sep 18, 2022 at 9:42 PM Mike Ounsworth <Mike.Ounsworth=40entrust.com@dmarc.ietf.org> wrote:

> Hi NIST PQC Forum!
>
> This is bubble-over from an IETF thread I started last week.
>
> Context: hash-then-sign schemes are good. For example, they allow you to pre-hash your potentially very large message and then send just the hash value to your cryptographic module to sign or verify. We like this pattern, it's good for bandwidth and latency of cryptographic modules. We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number; in the case of non-randomized SPHINCS+ and Dilithium, that message digest is keyed with values derived from the public key (for completeness: randomized SPHINCS+ seems to be the only to do both).
>
> A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).
>
> But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

> Question:
>
> So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? Is it ok to take m' = SHA256(m) and then sign m' ? If we care about the built-in collision-resistance, then the answer is probably "No". Is it safe to externalize the keyed message digest step of SPHINCS+, Dilithium, FALCON? In the non-randomized versions where the keyed message digest only relies on values in the public key, I would think the answer is "Yes". For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".
>
> I'm eager to hear more on the design rationale for starting with a randomized or deterministic keyed message digest, and recommendations for the safe way to external pre-hashes with these schemes.
>
> ---
> Mike Ounsworth
> Software Security Architect, Entrust
>
>
>
> ---
>
>

--
SY, Dmitry Belyavsky

From: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Mike Ounsworth <mike.ounsworth@entrust.com>, pqc-forum <pqc-forum@list.nist.gov>
CC: pqc@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] RE: Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 08:42:43 AM ET

If you're looking for design justifications, in the case of Sphincs+, we prepend both data from the public key and a random value for two reasons:

- We prepend random data, as you suspected, to avoid collision attacks; we don't assume collision resistance anywhere else in the Sphincs+ structure, but instead rely on weaker assumptions of the hash function.
- We prepend data from the public key to avoid multitarget attacks; that is, if the attacker has multiple public keys that he is attacking, we try to make it so that he has no advantage over an attacker with a single public key.

Of course, Sphincs+ is an intentionally conservative design; one could argue that the second attack doesn't really apply to SHA256(m) (because even if the attacker has 2^64 public keys that all signed messages, the direct multicollision attack without this protection would still take an expected 2^192 time on a conventional computer, which is completely infeasible, and at least 2^96 time on a Quantum one (and likely more – coming up with a more realistic lower bound is nontrivial) – I would argue that is similarly infeasible.

On the other hand, if one were to use prehashing, I would argue that the prehash should be with a very conservative hash function (say, SHA-512 or SHA3-512); we are putting all our hybrid eggs in this one hashing basket, and so we should make sure this one basket is a good one.

From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Sunday, September 18, 2022 3:42 PM
To: pqc-forum <pqc-forum@list.nist.gov>
Cc: pqc@ietf.org; cfrg@irtf.org
Subject: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?

Hi NIST PQC Forum!

This is bubble-over from an IETF thread I started last week.

Context: hash-then-sign schemes are good. For example, they allow you to pre-hash your potentially very large message and then send just the hash value to your cryptographic module to sign or verify. We like this pattern, it's good for bandwidth and latency of cryptographic modules. We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number; in the case of non-randomized SPHINCS+ and Dilithium, that message digest is keyed with values derived from the public key (for completeness: randomized SPHINCS+ seems to be the only to do both).

A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).

But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

Question:

So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? Is it ok to take m' = SHA256(m) and then sign m' ? If we care about the built-in collision-resistance, then the answer is probably "No". Is it safe to externalize the keyed message digest step of SPHINCS+, Dilithium, FALCON? In the non-randomized versions where the keyed message digest only relies on values in the public key, I would think the answer is "Yes". For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".

I'm eager to hear more on the design rationale for starting with a randomized or deterministic keyed message digest, and recommendations for the safe way to external pre-hashes with these schemes.

Mike Ounsworth
Software Security Architect, Entrust

From: Jeff Burdges <burdges@gnunet.org> via pqc-forum@list.nist.gov
To: Simon Josefsson <simon=40josefsson.org@dmarc.ietf.org>
CC: Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>, pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 09:23:07 AM ET

On Sep 19 2022, at 10:54 am, Simon Josefsson <simon=40josefsson.org@dmarc.ietf.org> wrote:

> So please, design your protocols to not assume the less secure
> hash-then-sign approach of public-key signing schemes.

We need the actual concerns to be spelled out clearly, so people can work within them, not simply an opaque rule against hashing.

It's clear people shall sign Merkle tree roots, like if making a certificate on a SPHINCS+ public key for example. ;)

Jeff

From: Soatok Dreamseeker <soatok.dhole@gmail.com> via CFRG <cfrg-bounces@irtf.org>
To: Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 09:27:52 AM ET
Attachments: ATT00001.txt

> Is it ok to take m’ = SHA256(m) and then sign m’

My preference would be to consider three design constraints:

1. Domain separation
2. Exclusive ownership
3. Context binding

If m is a low-entropy message, you might have some interesting cross-protocol interactions from just a simple hash. Instead of m' = SHA256(m), m' = HMAC_SHA256(c, m) where c is a constant specific to the signature algorithm (i.e. "SPHINCS+SHA3-256"). This gives you the "domain separation" property from my above list.

For exclusive ownership, include the public key in the message being hashed. For multipart messages, prefix each part with the length (as a 64-bit unsigned integer) of the piece. So now you have HMAC_SHA256(c, len(pk) || pk || len(m) || m).

In code, this might look like this:

```
// Given (secretKey, publicKey, message, SIG_ALGO)
state = crypto.createHmac('sha256', SIG_ALGO)
state.update(len64(publicKey))
state.update(publicKey)
state.update(len64(message))
state.update(lmessage)
return sphincsPlus.sign(state.digest(), secretKey)
```

Does this need to be specified at a lower level? Perhaps not. For starters, the EO concerns are relevant for RSA and ECDSA. I don't know offhand if they apply to Lattice signatures or hash-based signatures.

That being said, addressing the final concern (context-binding) extends the above code snippet to include some protocol-specific constant. This might be useful if you wanted to use the same keypair for multiple protocols, but only deal with the overhead of one CA certificate.

```
// Given (secretKey, publicKey, message, SIG_ALGO, CONTEXT)
state = crypto.createHmac('sha256', SIG_ALGO)
state.update(len64(CONTEXT))
state.update(CONTEXT)
state.update(len64(publicKey))
state.update(publicKey)
state.update(len64(message))
state.update(lmessage)
return sphincsPlus.sign(state.digest(), secretKey)
```

Just my 2c.

On Sun, Sep 18, 2022 at 3:43 PM Mike Ounsworth <Mike.Ounsworth=40entrust.com@dmarc.ietf.org> wrote:

> Hi NIST PQC Forum!
>
> This is bubble-over from an IETF thread I started last week.
>
> Context: hash-then-sign schemes are good. For example, they allow you to pre-hash your potentially very large message and then send just the hash value to your cryptographic module to sign or verify. We like this pattern, it's good for bandwidth and latency of cryptographic modules. We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number; in the case of non-randomized SPHINCS+ and Dilithium, that message digest is keyed with values derived from the public key (for completeness: randomized SPHINCS+ seems to be the only to do both).
>
> A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same

message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).

But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

Question:

So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? Is it ok to take m' = SHA256(m) and then sign m' ? If we care about the built-in collision-resistance, then the answer is probably "No". Is it safe to externalize the keyed message digest step of SPHINCS+, Dilithium, FALCON? In the non-randomized versions where the keyed message digest only relies on values in the public key, I would think the answer is "Yes". For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".

I'm eager to hear more on the design rationale for starting with a randomized or deterministic keyed message digest, and recommendations for the safe way to external pre-hashes with these schemes.

Mike Ounsworth
Software Security Architect, Entrust

From: Jeff Burdges <burdges@gnunet.org> via pqc-forum@list.nist.gov
To: Scott Fluhrer (sfluhrer) <sfluhrer=40cisco.com@dmarc.ietf.org>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 09:42:36 AM ET

On Sep 19 2022, at 2:42 pm, Scott Fluhrer (sfluhrer)
<sfluhrer=40cisco.com@dmarc.ietf.org> wrote:

> If you're looking for design justifications, in the case of Sphincs+,
> we prepend both data from the public key and a random value for two reasons:

There exist protocols that require verifiable random functions (VRFs),
of which the only performant conservative PQ scheme is stateful
single-layer XMSS, meaning a simple hash-based signature without
internal certificates.

It's not good that keys must rotate frequently of course. In fact key
rotation kinda kills some nice VRF protocols like NSEC5.

> We prepend random data, as you suspected, to avoid collision attacks;
> we don't assume collision resistance anywhere else in the Sphincs+
> structure, but instead rely on weaker assumptions of the hash function.

VRFs cannot prepend random data obviously.

> We prepend data from the public key to avoid multitarget attacks; that
> is, if the attacker has multiple public keys that he is attacking, we
> try to make it so that he has no advantage over an attacker with a
> single public key.

VRFs can prepend the public key however (and typically do so already).

Best,
Jeff

From: Riad S. Wahby <rsw@jfet.org> via CFRG <cfrg-bounces@irtf.org>
To: Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 10:34:52 AM ET

Hello Mike,

For Falcon, randomization is extremely important for another reason.
Section 2.2.2 of the specification has details.
https://gcc02.safelinks.protection.outlook.com/?url=https%3A%2F%2Ffalcon-sign.info%2Ffalcon.pdf&amp;data=05%7C01%7Cquynh.dang%40nist.gov%7C26c90a00a5ff45cf1fc008da9a4c1af5%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C637991948918592597%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&amp;sdata=N1CtL0E7IMa637GOm5djllH95QRQmbHYAK1NT7vBecc%3D&amp;reserved=0

In short: revealing two distinct signatures on the same message breaks
the scheme. If the signer were completely deterministic that wouldn't
be an issue, but in practice it's not: for performance reasons, the Falcon
signer uses hardware-accelerated floating-point arithmetic to sample from
a specific distribution, and different FPUs are allowed by IEEE-754 to give
different results. This means that signing the same message with the same
key on two different machines could accidentally reveal the signing key.

To address this, the Falcon signer salts the message with a long enough
random string that the signer will never sign the same message twice.
This implicitly gives protection against the attacks that you mention,
but even if those attacks are not a concern the Falcon signer must still
be randomized (or the scheme must be made stateful, or the signer must
use deterministic software floating-point routines, or the key must only
be used on one machine——each a potentially annoying limitation).

Based on the above I would be terrified of a Falcon implementation that
depended on the caller to salt and pre-hash.

Cheers,

-=rsw

Mike Ounsworth <Mike.Ounsworth=40entrust.com@dmarc.ietf.org> wrote:

> Hi NIST PQC Forum!

>

> This is bubble-over from an IETF thread I started last week.

>

> Context: hash-then-sign schemes are good. For example, they allow you to pre-hash your potentially very large message and then send just the hash value to your cryptographic module to sign or verify. We like this pattern, it's good for bandwidth and latency of cryptographic modules. We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest - in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number; in the case of non-randomized SPHINCS+ and Dilithium, that message digest is keyed with values derived from the public key (for completeness: randomized SPHINCS+ seems to be the only to do both).

>

> A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).

>

> But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

>

> Question:

> So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? Is it ok to take m' = SHA256(m) and then sign m' ? If we care about the built-in collision-resistance, then the answer is probably "No". Is it safe to externalize the keyed message digest step of SPHINCS+, Dilithium, FALCON? In the non-randomized versions where the keyed message digest only relies on values in the public key, I would think the answer is "Yes". For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".

>

>

> I'm eager to hear more on the design rationale for starting with a randomized or deterministic keyed message digest, and recommendations for the safe way to external pre-hashes with these schemes.

>

> ---

> Mike Ounsworth

> Software Security Architect, Entrust

>

From: Kampanakis, Panos <kpanos@amazon.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Riad S. Wahby <rsw@jfet.org>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] RE: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 11:27:39 AM ET

> Based on the above I would be terrified of a Falcon implementation that depended on the caller to salt and pre-hash.

Thx for the clarifications. Let's say we passed the SHA2/3-512 digest of the message to Falcon for randomized signing (as specified). Are there any practical concerns with that?

-----Original Message-----
From: CFRG <cfrg-bounces@irtf.org> On Behalf Of Riad S. Wahby
Sent: Monday, September 19, 2022 10:34 AM
To: Mike Ounsworth <Mike.Ounsworth=40entrust.com@dmarc.ietf.org>
Cc: pqc-forum <pqc-forum@list.nist.gov>; pqc@ietf.org; cfrg@irtf.org
Subject: RE: [EXTERNAL][CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?

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.

Hello Mike,

For Falcon, randomization is extremely important for another reason.
Section 2.2.2 of the specification has details.

https://gcc02.safelinks.protection.outlook.com/?url=https%3A%2F%2Ffalcon-sign.info%2Ffalcon.pdf&amp;data=05%7C01%7Cyi-kai.liu%40nist.gov%7C87d0f3047ec44d5c1b1608da9a537ab5%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C637991980589813367%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C2000%7C%7C%7C&amp;sdata=oVqh1×4TTdKMI%2BDoS67okOIuPURP%2FL8×6r1LYKPGEBw%3D&amp;reserved=0

In short: revealing two distinct signatures on the same message breaks the scheme. If the signer were completely deterministic that wouldn't be an issue, but in practice it's not: for performance reasons, the Falcon signer uses hardware-accelerated floating-point arithmetic to sample from a specific distribution, and different FPUs are allowed by IEEE-754 to give different results. This means that signing the same message with the same key on two different machines could accidentally reveal the signing key.

To address this, the Falcon signer salts the message with a long enough random string that the signer will never sign the same message twice.
This implicitly gives protection against the attacks that you mention, but even if those attacks are not a concern the Falcon signer must still be randomized (or the scheme must be made stateful, or the signer must use deterministic software floating-point routines, or the key must only be used on one machine——each a potentially annoying limitation).

Based on the above I would be terrified of a Falcon implementation that depended on the caller to salt and pre-hash.

Cheers,

-=rsw

From: Riad S. Wahby <rsw@jfet.org> via CFRG <cfrg-bounces@irtf.org>
To: Kampanakis, Panos <kpanos@amazon.com>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 01:53:09 PM ET

"Kampanakis, Panos" <kpanos@amazon.com> wrote:
>> Based on the above I would be terrified of a Falcon implementation that
>> depended on the caller to salt and pre-hash.
>
> Thx for the clarifications. Let's say we passed the SHA2/3-512 digest
> of the message to Falcon for randomized signing (as specified). Are there
> any practical concerns with that?

No concerns beyond the more general question as to whether pre-hashing is appropriate or not. In other words: if you're willing to pre-hash in other cases, you should also be willing to pre-hash in this case provided that the Falcon signer salts its inputs in compliance with the specification.

Cheers,

-=rsw

From: Mike Ounsworth <mike.ounsworth@entrust.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Mike Ounsworth <mike.ounsworth@entrust.com>, pqc-forum <pqc-forum@list.nist.gov>
CC: pqc@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] RE: Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 05:17:54 PM ET

Thank you all for the wholesome discussion all!

Here is my attempt to summarize: we have a few fundamental options:

Option 0: Do not pre-hash; send the whole message to the cryptographic primitive.

Discussion: There will be at least some applications where this is not practical; for example imagine signing a 25 mb email with a smartcard. Streaming the entire 25 mb to the smartcard sounds like you'd be sitting there waiting for a human-irritating amount of time. Validation of firmware images during secure boot is another case that comes to mind where you want to digest in-place and not stream the firmware image over an internal BUS.

Option 1: Simple pre-hash m' = SHA256(m); sign(m)

Discussion: Don't, for various reasons, none of which are catastrophic to the algorithm security, but there are better ways.

Option 2: Externalize the keyed digest step of SPHINCS+, Dilithium, FALCON to the client.

Discussion: REALLY DON'T! This can be private-key-recovery-level catastrophic for FALCON. For Dilithium and non-randomized SPHINCS+ this might be cryptographically sound, but regardless, moving part of the algorithm outside the crypto module boundary is unlikely to ever pass a FIPS validation.

Option 3: Application-level envelopes

Discussion: if your application has a need to only send a small amount of data to the crypto module, then your application needs to define a compressing envelope format, and sign that. How fancy the envelope format needs to be is dictated by the security needs of the protocol – ie collision resistance, entropy, contains a nonce, algorithm code footprint, performance, etc. Downside is that we're not solving this problem centrally, but delegating the problem of doing this securely to each protocol design team.

This seems to be the winning option.

Have I understood and summarized correctly?

Mike Ounsworth
Software Security Architect, Entrust

From: 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov>
Sent: September 18, 2022 2:42 PM
To: pqc-forum <pqc-forum@list.nist.gov>
Cc: pqc@ietf.org; cfrg@irtf.org
Subject: [EXTERNAL] [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, 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.

Hi NIST PQC Forum!

This is bubble-over from an IETF thread I started last week.

Context: hash-then-sign schemes are good. For example, they allow you to pre-hash your potentially very large message and then send just the hash value to your cryptographic module to sign or verify. We like this pattern, it's good for bandwidth and latency of cryptographic modules. We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number; in the case of non-randomized SPHINCS+ and Dilithium, that message digest is keyed with values derived from the public key (for completeness: randomized SPHINCS+ seems to be the only to do both).

A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).

But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

Question:

So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? Is it ok to take m' = SHA256(m) and then sign m' ? If we care about the built-in collision-resistance, then the answer is probably "No". Is it safe to externalize the keyed message digest step of SPHINCS+, Dilithium, FALCON? In the non-randomized versions where the keyed message digest only relies on values in the public key, I would think the answer is "Yes". For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".

I'm eager to hear more on the design rationale for starting with a randomized or deterministic keyed message digest, and recommendations for the safe way to external pre-hashes with these schemes.

Mike Ounsworth
Software Security Architect, Entrust

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: Soatok Dreamseeker <soatok.dhole@gmail.com>
CC: Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>, pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: [pqc-forum] Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Monday, September 19, 2022 05:24:39 PM ET

On Mon, Sep 19, 2022 at 9:26 AM Soatok Dreamseeker <soatok.dhole@gmail.com> wrote:

> > Is it ok to take m' = SHA256(m) and then sign m'
>
> My preference would be to consider three design constraints:
>
> 1. Domain separation
> 2. Exclusive ownership
> 3. Context binding

I agree with those concerns, but...

Seems to me that there is a layering issue here and two sets of concerns driving a very similar technical control on both sides of the abstraction boundary where these are probably better addressed in one place.

And we are getting a lot of abstract reasoning that is being justified with technical terms whose meaning is not necessarily shared by all the people involved.

So lets have a concrete example. I have a 5TB hard drive on it. I want to create a signature on the image taken by the forensics tool and it is 2TB after compression. As far as the application is concerned, that is the signature payload. That is not necessarily the input to the signature.

A signature over the raw payload is has never been very useful on its own. There has been data bound into the signature since PKCS#1 which has always bound the digest algorithm into the sig to prevent digest substitution attacks.

The question is not whether you want to tie in additional data but what data and where. In the case of some of the PDQ algorithms we have the question of whether to make the signature deterministic or not and if so, where that happens

If the core signature algorithm is s(m,k) where k is the key and we want to include a nonce, do we want to create the signature over:

A: s (payload + meta + nonce, k)

B: s (H(payload) + meta + nonce, k)

C: s (H(H(payload) + meta1) + meta2 + nonce, k)

The best performance wise is B since we only end up doing one extra hash. A is clearly ludicrous and isn't going to happen no matter what because a HSM likely doesn't have the bandwidth to process that amount of data.

My preference is to always use a hash over an input that includes a randomized nonce because it provides a lot of protection against a lot of implementation and protocol blunders. The only downside is that it makes testing problematic.

Back in the day, PCKS#1 and PKCS#7 were coming from the same place. Now they are different. I want my cryptographic packaging format to be doing all the work of assembling the input that goes into the signature function because my experience is that unless great care is taken, having a responsibility divided across two layers tends to end up with it being performed by neither rather than both.

If I care about a signature enough to use Dilithium, I am almost certainly going to be enrolling it in a notary chain.

From: Michael Richardson <mcr+ietf@sandelman.ca> via CFRG <cfrg-bounces@irtf.org>
To: Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>
CC: pqc@ietf.org, pqc-forum <pqc-forum@list.nist.gov>, cfrg@irtf.org
Subject: Re: [CFRG] [Pqc] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Tuesday, September 20, 2022 06:08:39 AM ET
Attachments: signature.asc
ATT00001.txt

```
Mike Ounsworth <Mike.Ounsworth=40entrust.com@dmarc.ietf.org> wrote:
    > time. Validation of firmware images during secure boot is another case
    > that comes to mind where you want to digest in-place and not stream the
    > firmware image over an internal BUS.

...

    > Option 3: Application-level envelopes

    > Discussion: if your application has a need to only send a small amount
    > of data to the crypto module, then your application needs to define a
    > compressing envelope format, and sign that. How fancy the envelope
    > format needs to be is dictated by the security needs of the protocol –
    > ie collision resistance, entropy, contains a nonce, algorithm code

So, basically the SUIT manifest is a "compressing envelope" that deals with
validating firmware images.  The manifest is somewhere between 500bytes and
10K (in really rare cases), so it should not be a problem to feed it all into
the crypto module.

    > This seems to be the winning option.
    > Have I understood and summarized correctly?

I think so.

    > footprint, performance, etc. Downside is that we’re not solving this
    > problem centrally, but delegating the problem of doing this securely to
    > each protocol design team.
```

So, on the "delegating" question: having a few good designs written down
would make the protocol designers happier.

--
Michael Richardson <mcr+IETF@sandelman.ca>, Sandelman Software Works
 -= IPv6 IoT consulting =-

From: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com> via pqc-forum <pqc-forum@list.nist.gov>
To: Christopher J Peikert <cpeikert@alum.mit.edu>, Mike Ounsworth <mike.ounsworth@entrust.com>
CC: pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: RE: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Tuesday, September 20, 2022 08:52:59 AM ET

If Falcon has an issue with bad entropy during signing, might I suggest a minor modification? I believe that the issue is that if Falcon gets the same randomized hash twice, and then uses different coins in the ffSampling to generate signatures, Bad Things happen.

If that is the case, the obvious approach (and Chris, please correct me if I missed something) would be to use a RFC-6979 style approach; take the randomized hash and some secret data (from the private key), and use that the generate the random coins for ffSampling, perhaps something as simple as SHAKE(private_key || hash). That way, if you somehow do come up with the same randomized hash value because of an entropy failure, you'll generate identical signatures, which is safe.

Now, this isn't perfect – if someone manages to do a fault attack on the deterministic coin generation, that'd still leak the key, and that'd be hard to detect. That may be enough to kill the idea in the HSM/third party hashing scenario. I believe it still makes sense if the same device does the entire Falcon operation (because it avoids a potential pitfall on poor entropy)

One might claim that the single-device scenario doesn't really need it, because the same rbg generates random bits for both the randomized hash and ffSampling. On the other hand, we have seen cases (with RSA key generation) where entropy was bad in the first part of the process (selecting the prime 'p') and became better in the second part (selecting the prime 'q') – because it has happened in the past, I believe that the possibility of it happening in the future is not implausible.

> **From:** pqc-forum@list.nist.gov <pqc-forum@list.nist.gov> **On Behalf Of** Christopher J Peikert
> **Sent:** Sunday, September 18, 2022 10:45 PM
> **To:** Mike Ounsworth <Mike.Ounsworth@entrust.com>
> **Cc:** pqc-forum <pqc-forum@list.nist.gov>; pqc@ietf.org; cfrg@irtf.org
> **Subject:** Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?

> On Sun, Sep 18, 2022 at 3:42 PM 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov> wrote:
>
>> We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number...
>>
>> A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).
>
> For Falcon in particular, there is a much more essential reason for "keying" the hash by a random number: releasing two different signatures for the same hash digest breaks the security reduction, and likely has very bad practical consequences as well. Note that the "core" signing procedure -- given a hash digest, output a signature -- is randomized, so different runs on the same digest will tend to produce different signatures.
>
> Randomized hashing ensures that the same digest essentially never reoccurs (assuming quality randomness), even when signing the same message multiple times. See Section 2.2.2 of the Falcon spec for further details.
>
> "Derandomizing" the signing procedure is another approach that may be suitable in some (but not all) scenarios. We have designed and implemented a version of this approach as a "thin wrapper" around the Falcon API; see the specification for the motivation, details, and caveats. Signatures produced using this deterministic method can easily be transformed into ones that are accepted by a verifier who expects randomized ones (but not vice versa). But due to this compatibility, our approach has the same issues as randomized Falcon when it comes to externalized hashing.
>
>> But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).
>
> This looks right to me, though it's not clear how much of a benefit this really is. For example, Falcon's security analysis models the hash function as a "random oracle," which is an even stronger assumption than collision resistance.

> > Question:
> >
> > So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? … For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a “No”.
>
> Unfortunately, the "dream" goal of externalized, deterministic, stateless hashing inherently requires collision resistance (of that external hash function): the adversary knows the function, so it can attempt to find collisions in it. Any valid signature for one of the colliding messages is inherently also a valid signature for the other colliding message.
>
> One remark is that, when modeling the hash function as a random oracle, it is not strictly necessary that the "random" value actually be random -- it is enough that it never be *repeated* (at least not for repeated signings of the same message under the same key). In this sense, it need only be a "nonce" in the true sense of the word: a Number used only ONCE.
>
> So, if there is a way to generate a true nonce outside the cryptographic module, perhaps that can be used. However, if the nonce is fairly predictable then we shouldn't expect to get the above-noted benefits of randomization (whatever they may be). And this is not even getting into the serious misuse risks of an API that allows the caller to set the nonce.
>
> Given all this, to me it seems that (for Falcon at least) the best way to externalize hashing of huge data is through some extra "envelope" layer that pre-hashes the data in a suitable way, then signs the result as if it were the true message. One can then focus on the security properties of the envelope.
>
> Sincerely yours in cryptography,
>
> Chris
>
>

From:	Christopher Peikert <christopher.peikert@algorand.com> via CFRG <cfrg-bounces@irtf.org>
To:	Scott Fluhrer (sfluhrer) <sfluhrer=40cisco.com@dmarc.ietf.org>
CC:	pqc@ietf.org, pqc-forum <pqc-forum@list.nist.gov>, cfrg@irtf.org, Christopher J Peikert <cpeikert@alum.mit.edu>
Subject:	Re: [CFRG] [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date:	Tuesday, September 20, 2022 09:10:30 AM ET
Attachments:	ATT00001.txt

On Tue, Sep 20, 2022 at 8:52 AM Scott Fluhrer (sfluhrer) <sfluhrer=40cisco.com@dmarc.ietf.org> wrote:

> If Falcon has an issue with bad entropy during signing, might I suggest a minor modification? I believe that the issue is that if Falcon gets the same randomized hash twice, and then uses different coins in the ffSampling to generate signatures, Bad Things happen.

That's correct.

> If that is the case, the obvious approach (and Chris, please correct me if I missed something) would be to use a RFC-6979 style approach; take the randomized hash and some secret data (from the private key), and use that the generate the random coins for ffSampling, perhaps something as simple as SHAKE(private_key || hash). That way, if you somehow do come up with the same randomized hash value because of an entropy failure, you'll generate identical signatures, which is safe.

This is the "derandomization" approach we take in the variant I previously mentioned; see here.

Note that this approach requires care even in "normal" usage scenarios. In particular, re-generating the same random coins still might *not* always guarantee the same output: floating-point operations can yield slightly different results in different implementations, as can different versions of the sampling algorithm itself.

For the latter issue, our variant includes a "salt version" that should be changed if/when the sampling algorithm changes.

For the former issue, the Falcon reference implementation happily includes a "floating-point emulation" mode, which is designed to ensure consistent results across platforms (and is much faster than you might think!). See Section 3.4 of the above-linked document for many details.

Sincerely yours in cryptography,

Chris

> Now, this isn't perfect – if someone manages to do a fault attack on the deterministic coin generation, that'd still leak the key, and that'd be hard to detect. That may be enough to kill the idea in the HSM/third party hashing scenario. I believe it still makes sense if the same device does the entire Falcon operation (because it avoids a potential pitfall on poor entropy)
>
> One might claim that the single-device scenario doesn't really need it, because the same rbg generates random bits for both the randomized hash and ffSampling. On the other hand, we have seen cases (with RSA key generation) where entropy was bad in the first part of the process (selecting the prime 'p') and became better in the second part (selecting the prime 'q') – because it has happened in the past, I believe that the possibility of it happening in the future is not implausible.
>
> **From:**pqc-forum@list.nist.gov <pqc-forum@list.nist.gov> **On Behalf Of** Christopher J Peikert
> **Sent:** Sunday, September 18, 2022 10:45 PM
> **To:** Mike Ounsworth <Mike.Ounsworth@entrust.com>
> **Cc:** pqc-forum <pqc-forum@list.nist.gov>; pqc@ietf.org; cfrg@irtf.org
> **Subject:** Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
>
> On Sun, Sep 18, 2022 at 3:42 PM 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov> wrote:
>
>> We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number...
>>
>> A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).
>
> For Falcon in particular, there is a much more essential reason for "keying" the hash by a random number: releasing two different signatures for the same hash digest breaks the

security reduction, and likely has very bad practical consequences as well. Note that the "core" signing procedure -- given a hash digest, output a signature -- is randomized, so different runs on the same digest will tend to produce different signatures.

Randomized hashing ensures that the same digest essentially never reoccurs (assuming quality randomness), even when signing the same message multiple times. See Section 2.2.2 of the Falcon spec for further details.

"Derandomizing" the signing procedure is another approach that may be suitable in some (but not all) scenarios. We have designed and implemented a version of this approach as a "thin wrapper" around the Falcon API; see the specification for the motivation, details, and caveats. Signatures produced using this deterministic method can easily be transformed into ones that are accepted by a verifier who expects randomized ones (but not vice versa). But due to this compatibility, our approach has the same issues as randomized Falcon when it comes to externalized hashing.

> But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

This looks right to me, though it's not clear how much of a benefit this really is. For example, Falcon's security analysis models the hash function as a "random oracle," which is an even stronger assumption than collision resistance.

> Question:
>
> So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? … For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".

Unfortunately, the "dream" goal of externalized, deterministic, stateless hashing inherently requires collision resistance (of that external hash function): the adversary knows the function, so it can attempt to find collisions in it. Any valid signature for one of the colliding messages is inherently also a valid signature for the other colliding message.

One remark is that, when modeling the hash function as a random oracle, it is not strictly necessary that the "random" value actually be random -- it is enough that it never be *repeated* (at least not for repeated signings of the same message under the same key). In this sense, it need only be a "nonce" in the true sense of the word: a Number used only ONCE.

So, if there is a way to generate a true nonce outside the cryptographic module, perhaps that can be used. However, if the nonce is fairly predictable then we shouldn't expect to get the above-noted benefits of randomization (whatever they may be). And this is not even getting into the serious misuse risks of an API that allows the caller to set the nonce.

Given all this, to me it seems that (for Falcon at least) the best way to externalize hashing of huge data is through some extra "envelope" layer that pre-hashes the data in a suitable way, then signs the result as if it were the true message. One can then focus on the security properties of the envelope.

Sincerely yours in cryptography,

Chris

From: Christopher J Peikert <cpeikert@alum.mit.edu> via pqc-forum@list.nist.gov
To: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>
CC: Mike Ounsworth <mike.ounsworth@entrust.com>, pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Tuesday, September 20, 2022 09:18:48 AM ET

(Re-posting to pqc-forum from a different address, due to list-membership fussiness. Apologies for any duplication in your inbox.)

On Tue, Sep 20, 2022 at 8:52 AM 'Scott Fluhrer (sfluhrer)' via pqc-forum <pqc-forum@list.nist.gov> wrote:

> If Falcon has an issue with bad entropy during signing, might I suggest a minor modification? I believe that the issue is that if Falcon gets the same randomized hash twice, and then uses different coins in the ffSampling to generate signatures, Bad Things happen.

That's correct.

> If that is the case, the obvious approach (and Chris, please correct me if I missed something) would be to use a RFC-6979 style approach; take the randomized hash and some secret data (from the private key), and use that the generate the random coins for ffSampling, perhaps something as simple as SHAKE(private_key || hash). That way, if you somehow do come up with the same randomized hash value because of an entropy failure, you'll generate identical signatures, which is safe.

This is the "derandomization" approach we take in the variant I previously mentioned; see here.

Note that this approach requires care even in "normal" usage scenarios. In particular, re-generating the same random coins still might *not* always guarantee the same output: floating-point operations can yield slightly different results in different implementations, as can different versions of the sampling algorithm itself.

For the latter issue, our variant includes a "salt version" that should be changed if/when the sampling algorithm changes.

For the former issue, the Falcon reference implementation happily includes a "floating-point emulation" mode, which is designed to ensure consistent results across platforms (and is much faster than you might think!). See Section 3.4 of the above-linked document for many details.

Sincerely yours in cryptography,

Chris

> Now, this isn't perfect – if someone manages to do a fault attack on the deterministic coin generation, that'd still leak the key, and that'd be hard to detect. That may be enough to kill the idea in the HSM/third party hashing scenario. I believe it still makes sense if the same device does the entire Falcon operation (because it avoids a potential pitfall on poor entropy)
>
> One might claim that the single-device scenario doesn't really need it, because the same rbg generates random bits for both the randomized hash and ffSampling. On the other hand, we have seen cases (with RSA key generation) where entropy was bad in the first part of the process (selecting the prime 'p') and became better in the second part (selecting the prime 'q') – because it has happened in the past, I believe that the possibility of it happening in the future is not implausible.
>
> > **From:**pqc-forum@list.nist.gov <pqc-forum@list.nist.gov> **On Behalf Of** Christopher J Peikert
> > **Sent:** Sunday, September 18, 2022 10:45 PM
> > **To:** Mike Ounsworth <Mike.Ounsworth@entrust.com>
> > **Cc:** pqc-forum <pqc-forum@list.nist.gov>; pqc@ietf.org; cfrg@irtf.org
> > **Subject:** Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
> >
> > On Sun, Sep 18, 2022 at 3:42 PM 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov> wrote:
> >
> > > We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number…
> > >
> > > A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the

> same message. Unless I missed something, I don't see any other justification given for the use of keyed message digests (randomized or deterministic).

For Falcon in particular, there is a much more essential reason for "keying" the hash by a random number: releasing two different signatures for the same hash digest breaks the security reduction, and likely has very bad practical consequences as well. Note that the "core" signing procedure -- given a hash digest, output a signature -- is randomized, so different runs on the same digest will tend to produce different signatures.

Randomized hashing ensures that the same digest essentially never reoccurs (assuming quality randomness), even when signing the same message multiple times. See Section 2.2.2 of the Falcon spec for further details.

"Derandomizing" the signing procedure is another approach that may be suitable in some (but not all) scenarios. We have designed and implemented a version of this approach as a "thin wrapper" around the Falcon API; see the specification for the motivation, details, and caveats. Signatures produced using this deterministic method can easily be transformed into ones that are accepted by a verifier who expects randomized ones (but not vice versa). But due to this compatibility, our approach has the same issues as randomized Falcon when it comes to externalized hashing.

> But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme's security may not need to rely on the hash function being collision resistant).

This looks right to me, though it's not clear how much of a benefit this really is. For example, Falcon's security analysis models the hash function as a "random oracle," which is an even stronger assumption than collision resistance.

> Question:
>
> So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? ... For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a "No".

Unfortunately, the "dream" goal of externalized, deterministic, stateless hashing inherently requires collision resistance (of that external hash function): the adversary knows the function, so it can attempt to find collisions in it. Any valid signature for one of the colliding messages is inherently also a valid signature for the other colliding message.

One remark is that, when modeling the hash function as a random oracle, it is not strictly necessary that the "random" value actually be random -- it is enough that it never be *repeated* (at least not for repeated signings of the same message under the same key). In this sense, it need only be a "nonce" in the true sense of the word: a Number used only ONCE.

So, if there is a way to generate a true nonce outside the cryptographic module, perhaps that can be used. However, if the nonce is fairly predictable then we shouldn't expect to get the above-noted benefits of randomization (whatever they may be). And this is not even getting into the serious misuse risks of an API that allows the caller to set the nonce.

Given all this, to me it seems that (for Falcon at least) the best way to externalize hashing of huge data is through some extra "envelope" layer that pre-hashes the data in a suitable way, then signs the result as if it were the true message. One can then focus on the security properties of the envelope.

Sincerely yours in cryptography,

Chris

> CH0PR11MB5444E50C9F95D88360C40A47C14C9%40CH0PR11MB5444.namprd11.prod.outlook.com.

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: Scott Fluhrer (sfluhrer) <sfluhrer@cisco.com>
CC: Christopher J Peikert <cpeikert@alum.mit.edu>, Mike Ounsworth <mike.ounsworth@entrust.com>, pqc-forum <pqc-forum@list.nist.gov>, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Tuesday, September 20, 2022 12:35:09 PM ET

+1

We went through all this with RFC-6879 and that is work we can and should re-use. Except that I would like to have a combination of the rigid approach AND also include a random nonce.

Having dug into the double spending / malleability issue that allegedly sunk MtGox, I am pretty sure that relying on signatures being deterministic is an anti-pattern. There is so much going on with a signature scheme that relying on the outputs being deterministic seems to be a step too far. While RFC-6879 specifies deterministic signature, there is no way to verify compliance, i.e. that a given signature is deterministic unless you have the private key.

So the series of APIs that I would like to see are

Outermost -> Innermost

byte[] SignData (byte[] data, PrivateKey key, EnvelopeID envelopeID, byte[]? attributes=null)

byte[] SignDigest (byte[] digest, AlgorithmID digestAlgorithm, PrivateKey key, byte[]? attributes=null)

----HSM boundary----

byte[] SignManifest (byte[] manifest, PrivateKey key)

byte[] SignNonce (byte[] nonce, byte[] manifest, PrivateKey key)

byte[] SignInner (byte[] signatureInput, PrivateKey key)

So the code using this approach is going to be (mostly) calling SignData to request an envelope in COSE, CMS/PKCS#7/OpenPGP etc format.

The specs for the envelope format specify how to marshall the data to produce a byte sequence that is the manifest.

The common specification for public key signing specifies how to go from (digest, digestAlgorithm, attributes) to a manifest and from (nonce, manifest) to signatureInput.

The individual signature specifications specify how to go from (signatureInput, key) to the signature data.

One of the core goals here is to not just prevent bad things happening, but to make it as easy as possible to verify compliance. We have many degrees of freedom here. We have a lot of envelope formats, a lot of digest algorithms and a lot of signature algorithms. And there is the possibility of mix-n-match attacks across each of these domains. Each part looks simple but the composition of those parts is not. And we want to get to apply formal methods to prove certain properties of the result. I am not up to speed with the modern formal methods tools available these days which we didn't have back when that was my thing but I can still decompose problems to the point where formal approaches are feasible.

This approach limits the variation introduced by the choice of envelope format to the SignData method. Everything from that point on is fixed. It also limits the scope of the signature algorithm designers to SignInner. The implementation of SignDigest, SignManifest and SignNonce are each fixed and common across all envelope formats and all signature algorithms. One consequence of this approach being that it may be possible to convert a COSE signature to a CMS signature provided that the verifier can parse any additional attributes provided.

A HSM would only expose the SignDigest method for private keys held internally. The device itself would probably need to expose the SignNonce and SignInner APIs for conformance testing purposes but allowing those methods to be used on protected keys is obviously prohibited. Specifying the conformance testing API in this fashion allows a test to be added to check that attempting to use a non-compliant method on a protected key is refused.

One consequence of this approach is that we have to agree what scheme is used to represent digest algorithm IDs in the construction of the manifest. It seems pretty clear to me that this has to be ASN.1 OIDs because that is what the only party which really cares about the ID representation cares about using. My code reduces all the ASN.1 OIDS to byte sequences before the compiler gets to see them so I don't care.

We can discuss how to construct the manifest and nonced manifest separately. I don't much care and the only people who are likely to really care are the people attempting to construct formal proofs. So while I would likely implement in DER if it was me doing it today, that is not really an implementation decision, it is a verification driven decision.

We should probably drop an algorithmID in for the signing algorithm as well. That is probably best added at the same time as the nonce is added in SignInner since that allows code to generate a single manifest and then reuse it across multiple signature algorithms. Since I don't have access to a threshold PQC algorithm, all my PQC signatures are going to be hybrid systems using Ed448 + Dilithium for quite a while.

On Tue, Sep 20, 2022 at 8:52 AM 'Scott Fluhrer (sfluhrer)' via pqc-forum <pqc-forum@list.nist.gov> wrote:

If Falcon has an issue with bad entropy during signing, might I suggest a minor modification? I believe that the issue is that if Falcon gets the same randomized hash twice, and then uses different coins in the ffSampling to generate signatures, Bad Things happen.

If that is the case, the obvious approach (and Chris, please correct me if I missed something) would be to use a RFC-6979 style approach; take the randomized hash and some secret data (from the private key), and use that the generate the random coins for ffSampling, perhaps something as simple as SHAKE(private_key || hash). That way, if you somehow do come up with the same randomized hash value because of an entropy failure, you'll generate identical signatures, which is safe.

Now, this isn't perfect – if someone manages to do a fault attack on the deterministic coin generation, that'd still leak the key, and that'd be hard to detect. That may be enough to kill the idea in the HSM/third party hashing scenario. I believe it still makes sense if the same device does the entire Falcon operation (because it avoids a potential pitfall on poor entropy)

One might claim that the single-device scenario doesn't really need it, because the same rbg generates random bits for both the randomized hash and ffSampling. On the other hand, we have seen cases (with RSA key generation) where entropy was bad in the first part of the process (selecting the prime 'p') and became better in the second part (selecting the prime 'q') – because it has happened in the past, I believe that the possibility of it happening in the future is not implausible.

From:pqc-forum@list.nist.gov <pqc-forum@list.nist.gov> **On Behalf Of** Christopher J Peikert
Sent: Sunday, September 18, 2022 10:45 PM
To: Mike Ounsworth <Mike.Ounsworth@entrust.com>
Cc: pqc-forum <pqc-forum@list.nist.gov>; pqc@ietf.org; cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?

On Sun, Sep 18, 2022 at 3:42 PM 'Mike Ounsworth' via pqc-forum <pqc-forum@list.nist.gov> wrote:

> We notice that SPHINCS+, CRYSTALS-Dilithium, and FALCON all start with a keyed message digest – in the case of randomized SPHINCS+ and FALCON, that message digest is keyed with a random number...

> > A quick skim through the submission documents for the three schemes shows that the message randomization is intended as a protection against differential and fault attacks since the traces would not be repeatable between subsequent signatures even of the same message. Unless I missed something, I don’t see any other justification given for the use of keyed message digests (randomized or deterministic).
>
> For Falcon in particular, there is a much more essential reason for "keying" the hash by a random number: releasing two different signatures for the same hash digest breaks the security reduction, and likely has very bad practical consequences as well. Note that the "core" signing procedure -- given a hash digest, output a signature -- is randomized, so different runs on the same digest will tend to produce different signatures.
>
> Randomized hashing ensures that the same digest essentially never reoccurs (assuming quality randomness), even when signing the same message multiple times. See Section 2.2.2 of the Falcon spec for further details.
>
> "Derandomizing" the signing procedure is another approach that may be suitable in some (but not all) scenarios. We have designed and implemented a version of this approach as a "thin wrapper" around the Falcon API; see the specification for the motivation, details, and caveats. Signatures produced using this deterministic method can easily be transformed into ones that are accepted by a verifier who expects randomized ones (but not vice versa). But due to this compatibility, our approach has the same issues as randomized Falcon when it comes to externalized hashing.
>
> > But it seems to me that, especially the randomized version, keyed message digests also protect against yet-to-be-discovered collision attacks in the underlying hash function because an attacker cannot pre-compute against an `r` chosen at signing time (ie the signature scheme’s security may not need to rely on the hash function being collision resistant).
>
> This looks right to me, though it's not clear how much of a benefit this really is. For example, Falcon's security analysis models the hash function as a "random oracle," which is an even stronger assumption than collision resistance.
>
> > Question:
> >
> > So what is the safe way to externally pre-hash messages for these schemes in order to achieve a hash-then-sign scheme? … For randomized versions, that would mean having access to a cryptographic RNG value outside the cryptographic module boundary, which, at least for FIPS validation, is probably a “No”.

> Unfortunately, the "dream" goal of externalized, deterministic, stateless hashing inherently requires collision resistance (of that external hash function): the adversary knows the function, so it can attempt to find collisions in it. Any valid signature for one of the colliding messages is inherently also a valid signature for the other colliding message.
>
> One remark is that, when modeling the hash function as a random oracle, it is not strictly necessary that the "random" value actually be random -- it is enough that it never be *repeated* (at least not for repeated signings of the same message under the same key). In this sense, it need only be a "nonce" in the true sense of the word: a Number used only ONCE.
>
> So, if there is a way to generate a true nonce outside the cryptographic module, perhaps that can be used. However, if the nonce is fairly predictable then we shouldn't expect to get the above-noted benefits of randomization (whatever they may be). And this is not even getting into the serious misuse risks of an API that allows the caller to set the nonce.
>
> Given all this, to me it seems that (for Falcon at least) the best way to externalize hashing of huge data is through some extra "envelope" layer that pre-hashes the data in a suitable way, then signs the result as if it were the true message. One can then focus on the security properties of the envelope.
>
> Sincerely yours in cryptography,
>
> Chris
>
>

> pqc-forum/
> CH0PR11MB5444E50C9F95D88360C40A47C14C9%40CH0PR11MB5444.namprd11.prod.outl
> ook.com.

From: Derek Atkins <datkins@veridify.com> via pqc-forum@list.nist.gov
To: sfluhrer@cisco.com, phill@hallambaker.com
CC: cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Tuesday, September 20, 2022 01:03:39 PM ET

Phill,

On Tue, 2022-09-20 at 12:34 -0400, Phillip Hallam-Baker wrote:

[snip]

> Outermost -> Innermost
>
> byte[] SignData (byte[] data, PrivateKey key, EnvelopeID envelopeID, byte[]? attributes=null)
>
> byte[] SignDigest (byte[] digest, AlgorithmID digestAlgorithm, PrivateKey key, byte[]? attributes=null)
>
> ----HSM boundary----
>
> byte[] SignManifest (byte[] manifest, PrivateKey key)
>
> byte[] SignNonce (byte[] nonce, byte[] manifest, PrivateKey key)
>
> byte[] SignInner (byte[] signatureInput, PrivateKey key)
>
> So the code using this approach is going to be (mostly) calling SignData to request an envelope in COSE, CMS/PKCS#7/OpenPGP etc format.

[snip]

> A HSM would only expose the SignDigest method for private keys held internally. The device itself would probably need to expose the SignNonce and SignInner APIs for conformance testing purposes but allowing those methods to be used on protected keys is obviously prohibited. Specifying the conformance testing API in this fashion allows a test to be added to check that attempting to use a non-compliant method on a protected key is refused.

Just to make sure I'm not confused, is this a typo, I would have expected you to say "An HSM only exposes the SignManifest method". Am I missing something?

-derek

--

Derek Atkins
Chief Technology Officer
Veridify Security - *Securing the Internet of Things®*

Office: 203.227.3151 x1343
Direct: 617.623.3745
Mobile: 617.290.5355
Email: DAtkins@Veridify.com

From: Brent Kimberley <brent.kimberley@durham.ca> via pqc-forum <pqc-forum@list.nist.gov>
To: Derek Atkins <datkins@veridify.com>, sfluhrer@cisco.com, phill@hallambaker.com
CC: cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, cfrg@irtf.org
Subject: RE: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Tuesday, September 20, 2022 05:28:53 PM ET

Likewise, one might want to dive into HSM reliability, availability, and serviceability.

From: pqc-forum@list.nist.gov <pqc-forum@list.nist.gov> **On Behalf Of** Derek Atkins
Sent: September 20, 2022 1:03 PM
To: sfluhrer@cisco.com; phill@hallambaker.com
Cc: cpeikert@alum.mit.edu; Mike.Ounsworth@entrust.com; pqc-forum@list.nist.gov; pqc@ietf.org; cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?

Phill,

On Tue, 2022-09-20 at 12:34 -0400, Phillip Hallam-Baker wrote:

[snip]

> Outermost -> Innermost
>
> byte[] SignData (byte[] data, PrivateKey key, EnvelopeID envelopeID, byte[]? attributes=null)
>
> byte[] SignDigest (byte[] digest, AlgorithmID digestAlgorithm, PrivateKey key, byte[]? attributes=null)
>
> ----HSM boundary----
>
> byte[] SignManifest (byte[] manifest, PrivateKey key)
>
> byte[] SignNonce (byte[] nonce, byte[] manifest, PrivateKey key)
>
> byte[] SignInner (byte[] signatureInput, PrivateKey key)
>
> So the code using this approach is going to be (mostly) calling SignData to request an envelope in COSE, CMS/PKCS#7/OpenPGP etc format.

[snip]

> A HSM would only expose the SignDigest method for private keys held internally. The device itself would probably need to expose the SignNonce and SignInner APIs for conformance testing purposes but allowing those methods to be used on protected keys is obviously prohibited. Specifying the conformance testing API in this fashion allows a test to be added to check that attempting to use a non-compliant method on a protected key is refused.

Just to make sure I'm not confused, is this a typo, I would have expected you to say "An HSM only exposes the SignManifest method". Am I missing something?

-derek

--

Derek Atkins
Chief Technology Officer

Veridify Security - *Securing the Internet of Things®*

Office: 203.227.3151 x1343
Direct: 617.623.3745
Mobile: 617.290.5355
Email: DAtkins@Veridify.com

--

From: Hubert Kario <hkario@redhat.com> via CFRG <cfrg-bounces@irtf.org>
To: Phillip Hallam-Baker <phill@hallambaker.com>
CC: pqc@ietf.org, pqc-forum <pqc-forum@list.nist.gov>, cfrg@irtf.org, Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>
Subject: Re: [CFRG] [Pqc] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Thursday, September 22, 2022 07:58:36 AM ET

```
On Monday, 19 September 2022 23:23:36 CEST, Phillip Hallam-Baker wrote:
> On Mon, Sep 19, 2022 at 9:26 AM Soatok Dreamseeker
> <soatok.dhole@gmail.com> wrote:
> Is it  ok to take m’ = SHA256(m) and then sign m’
>
> My preference would be to consider three design constraints:
>
> 1. Domain separation
> 2. Exclusive ownership
> 3. Context binding
>
> I agree with those concerns, but ...
>
> Seems to me that there is a layering issue here and two sets of
> concerns driving a very similar technical control on both sides
> of the abstraction boundary where these are probably better
> addressed in one place.
>
> And we are getting a lot of abstract reasoning that is being
> justified with technical terms whose meaning is not necessarily
> shared by all the people involved.
>
> So lets have a concrete example. I have a 5TB hard drive on it.
> I want to create a signature on the image taken by the
> forensics tool and it is 2TB after compression. As far as the
> application is concerned, that is the signature payload. That is
> not necessarily the input to the signature.
>
> A signature over the raw payload is has never been very useful
> on its own. There has been data bound into the signature since
> PKCS#1 which has always bound the digest algorithm into the sig
> to prevent digest substitution attacks.
>
> The question is not whether you want to tie in additional data
> but what data and where. In the case of some of the PDQ
> algorithms we have the question of whether to make the signature
> deterministic or not and if so, where that happens
>
> If the core signature algorithm is s(m,k)  where k is the key
> and we want to include a nonce, do we want to create the
> signature over:
>
> A: s (payload + meta + nonce, k)
> B: s (H(payload) + meta + nonce, k)
> C: s (H(H(payload) + meta1) + meta2 + nonce, k)
>
> The best performance wise is B since we only end up doing one
> extra hash. A is clearly ludicrous and isn't going to happen no
> matter what because a HSM likely doesn't have the bandwidth to
> process that amount of data.

I don't understand how this is a problem.

You can have a cryptographic module defined as the HSM + PKCS#11 driver,
then the hashing can happen in the driver, in software, in a way that
ensures
that the HSM signs only the data that was hashed by the driver, not by
software
external to the cryptographic module.

PKCS#11 already defines a C_SignInit/C_SignUpdate/C_SignFinal API for the
combined hash+sign operation.

Or, you can define the cryptographic module interface as the PCIe connector
(other connectors are available). Then FIPS 140-3 already requires
that the module ensures that the the signature was made over a hash
calculated
```

with a FIPS certified module, so in practice you already have to push that
data over the PCIe connector.

--
Regards,
Hubert Kario
Principal Quality Engineer, RHEL Crypto team
Web: https://gcc02.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.cz.redhat.com%2F&amp;data=05%7C01%7Cquynh.dang%40nist.gov%7C323e0d87598c495a5ae908da9c91c485%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C637994447168499923%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&amp;sdata=%2By%2BrbwWl10tk%2F4Zqaw7AsvbZiX1IVSnZRrwEA8q43dY%3D&amp;reserved=0
Red Hat Czech s.r.o., Purkyňova 99/71, 612 45, Brno, Czech Republic

---

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: Derek Atkins <datkins@veridify.com>
CC: sfluhrer@cisco.com, cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Wednesday, September 28, 2022 11:49:34 AM ET

On Tue, Sep 20, 2022 at 1:03 PM Derek Atkins <datkins@veridify.com> wrote:

> Phill,
>
> On Tue, 2022-09-20 at 12:34 -0400, Phillip Hallam-Baker wrote:
>
> [snip]
>
>> Outermost -> Innermost
>>
>> byte[] SignData (byte[] data, PrivateKey key, EnvelopeID envelopeID, byte[]? attributes=null)
>>
>> byte[] SignDigest (byte[] digest, AlgorithmID digestAlgorithm, PrivateKey key, byte[]? attributes=null)
>>
>> ----HSM boundary----
>>
>> byte[] SignManifest (byte[] manifest, PrivateKey key)
>>
>> byte[] SignNonce (byte[] nonce, byte[] manifest, PrivateKey key)
>>
>> byte[] SignInner (byte[] signatureInput, PrivateKey key)
>>
>> So the code using this approach is going to be (mostly) calling SignData to request an envelope in COSE, CMS/PKCS#7/OpenPGP etc format.
>
> [snip]
>
>> A HSM would only expose the SignDigest method for private keys held internally. The device itself would probably need to expose the SignNonce and SignInner APIs for conformance testing purposes but allowing those methods to be used on protected keys is obviously prohibited. Specifying the conformance testing API in this fashion allows a test to be added to check that attempting to use a non-compliant method on a protected key is refused.
>
> Just to make sure I'm not confused, is this a typo, I would have expected you to say "An HSM only exposes the SignManifest method". Am I missing something?

An HSM only exposes the SignManifest method in FIPS mode. If you put it in FIPS Mode, it can never be put in test mode and can never give the debug information or provide non-FIPS functions.

We used to spray the Test HSMs yellow with TEST in black so nobody would ever be silly enough to create a trust chain to 'em.

From: Derek Atkins <datkins@veridify.com> via pqc-forum@list.nist.gov
To: phill@hallambaker.com
CC: cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, sfluhrer@cisco.com, cfrg@irtf.org
Subject: Re: [pqc-forum] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?
Date: Wednesday, September 28, 2022 12:11:42 PM ET

Hi,

On Wed, 2022-09-28 at 11:49 -0400, Phillip Hallam-Baker wrote:

> On Tue, Sep 20, 2022 at 1:03 PM Derek Atkins <datkins@veridify.com> wrote:
>
>> Phill,
>>
>> On Tue, 2022-09-20 at 12:34 -0400, Phillip Hallam-Baker wrote:
>>
>> [snip]
>>
>>> Outermost -> Innermost
>>>
>>> byte[] SignData (byte[] data, PrivateKey key, EnvelopeID envelopeID, byte[]? attributes=null)
>>>
>>> byte[] SignDigest (byte[] digest, AlgorithmID digestAlgorithm, PrivateKey key, byte[]? attributes=null)
>>>
>>> ----HSM boundary----
>>>
>>> byte[] SignManifest (byte[] manifest, PrivateKey key)
>>>
>>> byte[] SignNonce (byte[] nonce, byte[] manifest, PrivateKey key)
>>>
>>> byte[] SignInner (byte[] signatureInput, PrivateKey key)
>>>
>>> So the code using this approach is going to be (mostly) calling SignData to request an envelope in COSE, CMS/PKCS#7/OpenPGP etc format.
>>
>> [snip]
>>
>>> A HSM would only expose the SignDigest method for private keys held internally. The device itself would probably need to expose the SignNonce and SignInner APIs for conformance testing purposes but allowing those methods to be used on protected keys is obviously prohibited.

> > > Specifying the conformance testing API in this fashion allows a test to be added to check that attempting to use a non-compliant method on a protected key is refused.
> >
> > Just to make sure I'm not confused, is this a typo, I would have expected you to say "An HSM only exposes the SignManifest method". Am I missing something?
>
> An HSM only exposes the SignManifest method in FIPS mode. If you put it in FIPS Mode, it can never be put in test mode and can never give the debug information or provide non-FIPS functions.
>
> We used to spray the Test HSMs yellow with TEST in black so nobody would ever be silly enough to create a trust chain to 'em.

You still didn't clarify. You originally said "An HSM would only expose the SignDigest method"... I was questioning whether this was a typo, as I would only expect an HSM to expose SignManifest.

Can you please clarify?

Thanks,

-derek

--

Derek Atkins
Chief Technology Officer
Veridify Security - *Securing the Internet of Things®*

Office: 203.227.3151 x1343
Direct: 617.623.3745
Mobile: 617.290.5355
Email: DAtkins@Veridify.com

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: Derek Atkins <datkins@veridify.com>
CC: cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, sfluhrer@cisco.com, cfrg@irtf.org
Subject: [pqc-forum] Covert channel attack means signing should be deterministic
Date: Wednesday, September 28, 2022 01:08:36 PM ET

Scroll through my reply to Derek to see the argument. Covert channels to leak RSA keys were absolutely a thing that occurred in the real world.

The problem with NOBUS (Nobody but US) is that it fails when there is an insider threat. And as recent events have shown, there are entirely repeatable mechanisms for creating insider threats.

On Wed, Sep 28, 2022 at 12:11 PM Derek Atkins <datkins@veridify.com> wrote:

> Hi,
>
> On Wed, 2022-09-28 at 11:49 -0400, Phillip Hallam-Baker wrote:
>
>> On Tue, Sep 20, 2022 at 1:03 PM Derek Atkins <datkins@veridify.com> wrote:
>>
>>> Phill,
>>>
>>> On Tue, 2022-09-20 at 12:34 -0400, Phillip Hallam-Baker wrote:
>>>
>>> [snip]
>>>
>>>> Outermost -> Innermost
>>>>
>>>> byte[] SignData (byte[] data, PrivateKey key, EnvelopeID envelopeID, byte[]? attributes=null)
>>>>
>>>> byte[] SignDigest (byte[] digest, AlgorithmID digestAlgorithm, PrivateKey key, byte[]? attributes=null)
>>>>
>>>> ----HSM boundary----
>>>>
>>>> byte[] SignManifest (byte[] manifest, PrivateKey key)
>>>>
>>>> byte[] SignNonce (byte[] nonce, byte[] manifest, PrivateKey key)
>>>>
>>>> byte[] SignInner (byte[] signatureInput, PrivateKey key)

> > > So the code using this approach is going to be (mostly) calling SignData to request an envelope in COSE, CMS/PKCS#7/OpenPGP etc format.
> >
> > [snip]
> >
> > > A HSM would only expose the SignDigest method for private keys held internally. The device itself would probably need to expose the SignNonce and SignInner APIs for conformance testing purposes but allowing those methods to be used on protected keys is obviously prohibited. Specifying the conformance testing API in this fashion allows a test to be added to check that attempting to use a non-compliant method on a protected key is refused.
> >
> > Just to make sure I'm not confused, is this a typo, I would have expected you to say "An HSM only exposes the SignManifest method". Am I missing something?
>
> An HSM only exposes the SignManifest method in FIPS mode. If you put it in FIPS Mode, it can never be put in test mode and can never give the debug information or provide non-FIPS functions.
>
> We used to spray the Test HSMs yellow with TEST in black so nobody would ever be silly enough to create a trust chain to 'em.

You still didn't clarify. You originally said "An HSM would only expose the SignDigest method"... I was questioning whether this was a typo, as I would only expect an HSM to expose SignManifest.

My proposal is that the method SignDigest is entirely constrained by a specification that is common across all algorithms and envelope formats. So there is never a reason to vary the implementation. JOSE, CMS, COSE, DARE, OpenPGP will all use the same implementation from that point on. So that is all that an HSM needs to expose for FIPS certified processing.

If however, we are talking about something less than a full HSM, a cryptographic co-processor on a CPU, we might well want to expose SignManifest so we can offload that complexity from the processor. So yes, you can move that part out to your driver :-)

Looking at the CPU co-processor implementation, there is a subtle issue here in that we inevitably expose a side channel that would allow the co-processor to leak the secret key because it chooses the nonce so the algorithm is inherently non-deterministic.

I will give the whole algorithm here because I don't want GCHQ visiting me with a secrecy order.

Goal: NOBUS compliant leakage of secret key of a nondeterministic signature scheme from an HSM with only twice the processing load.

First step is to reduce the private key to manageable size, I will use a deterministic key generation approach from a 128 bit seed.

Let us imagine that the HSM creates and publishes signatures frequently so we can expect to collect tens of thousands of examples. The approach I am going to take is trial and error. The signer chooses a random nonce, creates two signatures signature and then checks to see if the first resulting signature matches a template f(seed, sig) => bool. If the signature matches, the first is returned, otherwise the second.

What this means is that the probability that the signature matches the template is 0.75 instead of 0.5. Which is more than sufficient to leak the value of seed over a few thousand signatures. Not least because all that you need to do is reduce the search space sufficiently to make an exhaustive search of the rest feasible.

The pattern could be quite simple, put the signature value through a digest, take the first byte. Use the low 7 bits to specify the bit position of the seed to be leaked and return true if the top bit matches the corresponding bit in the seed.

For a bit more obfuscation, we could use a MAC with a secret key so that the basis for the statistical variations is hidden.

I guess I should read just Motti's book instead of reinventing it all myself.

OK, so this seems to make the argument that a signing scheme MUST be deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.

From:	Natanael <natanael.l@gmail.com> via CFRG <cfrg-bounces@irtf.org>
To:	Phillip Hallam-Baker <phill@hallambaker.com>
CC:	Derek Atkins <datkins@veridify.com>, pqc@ietf.org, IRTF CFRG <cfrg@irtf.org>, sfluhrer@cisco.com, pqc-forum@list.nist.gov, cpeikert@alum.mit.edu
Subject:	Re: [CFRG] Covert channel attack means signing should be deterministic
Date:	Wednesday, September 28, 2022 01:24:53 PM ET
Attachments:	ATT00001.txt

Den ons 28 sep. 2022 19:08Phillip Hallam-Baker <phill@hallambaker.com> skrev:

> Looking at the CPU co-processor implementation, there is a subtle issue here in that we inevitably expose a side channel that would allow the co-processor to leak the secret key because it chooses the nonce so the algorithm is inherently non-deterministic.
>
> I will give the whole algorithm here because I don't want GCHQ visiting me with a secrecy order.
>
> Goal: NOBUS compliant leakage of secret key of a nondeterministic signature scheme from an HSM with only twice the processing load.
>
> First step is to reduce the private key to manageable size, I will use a deterministic key generation approach from a 128 bit seed.
>
> Let us imagine that the HSM creates and publishes signatures frequently so we can expect to collect tens of thousands of examples. The approach I am going to take is trial and error. The signer chooses a random nonce, creates two signatures signature and then checks to see if the first resulting signature matches a template f(seed, sig) => bool. If the signature matches, the first is returned, otherwise the second.
>
> What this means is that the probability that the signature matches the template is 0.75 instead of 0.5. Which is more than sufficient to leak the value of seed over a few thousand signatures. Not least because all that you need to do is reduce the search space sufficiently to make an exhaustive search of the rest feasible.
>
> The pattern could be quite simple, put the signature value through a digest, take the first byte. Use the low 7 bits to specify the bit position of the seed to be leaked and return true if the top bit matches the corresponding bit in the seed.
>
> For a bit more obfuscation, we could use a MAC with a secret key so that the basis for the statistical variations is hidden.
>
> I guess I should read just Motti's book instead of reinventing it all myself.

> OK, so this seems to make the argument that a signing scheme MUST be deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.

There's a possible (but inefficient) way to address that. Commit to a sequence of random values to be used in order, then use a Zero-knowledge proof to demonstrate you used the correct random value input in the signature. It adds state to be managed, it's definitely a lot slower, but it should work.

From: Tony Arcieri <bascule@gmail.com> via pqc-forum@list.nist.gov
To: Phillip Hallam-Baker <phill@hallambaker.com>
CC: Derek Atkins <datkins@veridify.com>, cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, sfluhrer@cisco.com, cfrg@irtf.org
Subject: Re: [pqc-forum] Covert channel attack means signing should be deterministic
Date: Wednesday, September 28, 2022 02:36:21 PM ET

On Wed, Sep 28, 2022 at 11:08 AM Phillip Hallam-Baker <phill@hallambaker.com> wrote:

> OK, so this seems to make the argument that a signing scheme MUST be deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.

The typical counterargument against deterministic signature schemes is that they enable fault attacks, and lattice-based schemes are no exception to this:

https://tches.iacr.org/index.php/TCHES/article/view/7267

Trying to work around such attacks by adding some sort of post-signature check, such as signing twice and comparing results, or ensuring that the signature verifies correctly, can be defeated with a double fault attack which bypasses whatever branch prevents disclosing the faulty signature.

Furthermore, mandating a deterministic scheme doesn't necessarily close the covert channel if there's no way for a verifier to determine the signature was produced correctly under a prescribed deterministic method.

--
Tony Arcieri

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: Natanael <natanael.l@gmail.com>
CC: Derek Atkins <datkins@veridify.com>, pqc@ietf.org, IRTF CFRG <cfrg@irtf.org>, sfluhrer@cisco.com, pqc-forum@list.nist.gov, cpeikert@alum.mit.edu
Subject: [pqc-forum] Re: [CFRG] Covert channel attack means signing should be deterministic
Date: Wednesday, September 28, 2022 03:11:45 PM ET

On Wed, Sep 28, 2022 at 1:24 PM Natanael <natanael.l@gmail.com> wrote:

> Den ons 28 sep. 2022 19:08Phillip Hallam-Baker <phill@hallambaker.com> skrev:
>
>> Looking at the CPU co-processor implementation, there is a subtle issue here in that we inevitably expose a side channel that would allow the co-processor to leak the secret key because it chooses the nonce so the algorithm is inherently non-deterministic.
>>
>> I will give the whole algorithm here because I don't want GCHQ visiting me with a secrecy order.
>>
>> Goal: NOBUS compliant leakage of secret key of a nondeterministic signature scheme from an HSM with only twice the processing load.
>>
>> First step is to reduce the private key to manageable size, I will use a deterministic key generation approach from a 128 bit seed.
>>
>> Let us imagine that the HSM creates and publishes signatures frequently so we can expect to collect tens of thousands of examples. The approach I am going to take is trial and error. The signer chooses a random nonce, creates two signatures signature and then checks to see if the first resulting signature matches a template f(seed, sig) => bool. If the signature matches, the first is returned, otherwise the second.
>>
>> What this means is that the probability that the signature matches the template is 0.75 instead of 0.5. Which is more than sufficient to leak the value of seed over a few thousand signatures. Not least because all that you need to do is reduce the search space sufficiently to make an exhaustive search of the rest feasible.
>>
>> The pattern could be quite simple, put the signature value through a digest, take the first byte. Use the low 7 bits to specify the bit position of the seed to be leaked and return true if the top bit matches the corresponding bit in the seed.
>>
>> For a bit more obfuscation, we could use a MAC with a secret key so that the basis for the statistical variations is hidden.

> > I guess I should read just Motti's book instead of reinventing it all myself.
> >
> > OK, so this seems to make the argument that a signing scheme MUST be deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.
>
> There's a possible (but inefficient) way to address that. Commit to a sequence of random values to be used in order, then use a Zero-knowledge proof to demonstrate you used the correct random value input in the signature. It adds state to be managed, it's definitely a lot slower, but it should work.

Alternatively, the nonce input must be a seed value that is combined with the manifest data and some secret (e.g. the private key) to derive the output.

That approach would allow us to pull the nonce value out of the HSM envelope so the internals of the HSM are entirely deterministic. But then as Tony Arceri points out, we become vulnerable to fault attacks.

From: Christopher J Peikert <cpeikert@alum.mit.edu> via pqc-forum@list.nist.gov
To: Tony Arcieri <bascule@gmail.com>
CC: Derek Atkins <datkins@veridify.com>, mike.ounsworth@entrust.com, Phillip Hallam-Baker <phill@hallambaker.com>, cfrg@irtf.org, pqc-forum@list.nist.gov, pqc@ietf.org, sfluhrer@cisco.com
Subject: Re: [pqc-forum] Covert channel attack means signing should be deterministic
Date: Wednesday, September 28, 2022 06:42:56 PM ET

On Wed, Sep 28, 2022 at 2:36 PM Tony Arcieri <bascule@gmail.com> wrote:

> On Wed, Sep 28, 2022 at 11:08 AM Phillip Hallam-Baker <phill@hallambaker.com> wrote:
>
> > OK, so this seems to make the argument that a signing scheme MUST be deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.
>
> Furthermore, mandating a deterministic scheme doesn't necessarily close the covert channel if there's no way for a verifier to determine the signature was produced correctly under a prescribed deterministic method.

This is a critically important point. In Falcon and Dilithium (at least), there are many valid signatures for a given message and public key. There is no apparent way to restrict to a unique valid signature, in a verifiable way. So, making the scheme “deterministic” does not prevent covert leakage, because the signer can still misbehave/leak without being caught.

Sincerely yours in cryptography,

Chris

From: Greg Maxwell <gmaxwell@gmail.com> via pqc-forum@list.nist.gov
To: Christopher J Peikert <cpeikert@alum.mit.edu>
CC: Tony Arcieri <bascule@gmail.com>, Derek Atkins <datkins@veridify.com>, mike.ounsworth@entrust.com, Phillip Hallam-Baker <phill@hallambaker.com>, cfrg@irtf.org, pqc-forum@list.nist.gov, pqc@ietf.org, sfluhrer@cisco.com
Subject: Re: [pqc-forum] Covert channel attack means signing should be deterministic
Date: Wednesday, September 28, 2022 07:06:56 PM ET

```
If the signatures are derandomized with a secret key, then an
unusually cautious user could load the same key material into two
independently designed/built/sold devices and continually compare
their output and abort if they disagree.

Now... if they're more likely to mess up handling the secret key
material than a single device is to exfiltrate their data is another
question...

On Wed, Sep 28, 2022 at 10:42 PM Christopher J Peikert
<cpeikert@alum.mit.edu> wrote:
>
> On Wed, Sep 28, 2022 at 2:36 PM Tony Arcieri <bascule@gmail.com> wrote:
>>
>> On Wed, Sep 28, 2022 at 11:08 AM Phillip Hallam-Baker <phill@hallambaker.com>
wrote:
>>>
>>> OK, so this seems to make the argument that a signing scheme MUST be
deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.
>>
>>
>> Furthermore, mandating a deterministic scheme doesn't necessarily close the covert
channel if there's no way for a verifier to determine the signature was produced
correctly under a prescribed deterministic method.
>
>
> This is a critically important point. In Falcon and Dilithium (at least), there are
many valid signatures for a given message and public key. There is no apparent way to
restrict to a unique valid signature, in a verifiable way. So, making the scheme
“deterministic” does not prevent covert leakage, because the signer can still
misbehave/leak without being caught.
```

```
>
> Sincerely yours in cryptography,
> Chris
>
> --
```

--

From: Samuel Lavery <sam.lavery@gmail.com> via pqc-forum@list.nist.gov
To: Tony Arcieri <bascule@gmail.com>
CC: Phillip Hallam-Baker <phill@hallambaker.com>, Derek Atkins <datkins@veridify.com>, cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, sfluhrer@cisco.com, cfrg@irtf.org
Subject: Re: [pqc-forum] Covert channel attack means signing should be deterministic
Date: Thursday, September 29, 2022 12:06:14 AM ET

My understanding is that for properly partitioned signature schemes, the additional entropy is "applied" to the partition that isn't dependent on the message. One also includes this value in the signature for the verifier to use.
This is the Boneh-Shen-Waters transform, it's a way to turn EUF-CMA schemes into sEUF-CMA. SPHINCS+ achieves this via OptRand. I do something similar in mine. It's entropy taken at signing time and shared. The scheme is still deterministic, but gives you 256 or whatever unique signatures per message without leaking any extra secret bits.

Cheers,

Sam

On Sep 28, 2022, at 11:35 AM, Tony Arcieri <bascule@gmail.com> wrote:

On Wed, Sep 28, 2022 at 11:08 AM Phillip Hallam-Baker <phill@hallambaker.com> wrote:

> OK, so this seems to make the argument that a signing scheme MUST be deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.

The typical counterargument against deterministic signature schemes is that they enable fault attacks, and lattice-based schemes are no exception to this:

https://tches.iacr.org/index.php/TCHES/article/view/7267

Trying to work around such attacks by adding some sort of post-signature check, such as signing twice and comparing results, or ensuring that the signature verifies correctly, can be defeated with a double fault attack which bypasses whatever branch prevents disclosing the faulty signature.

Furthermore, mandating a deterministic scheme doesn't necessarily close the covert channel if there's no way for a verifier to determine the signature was produced correctly under a prescribed deterministic method.

--
Tony Arcieri

From: Phillip Hallam-Baker <phill@hallambaker.com> via pqc-forum@list.nist.gov
To: Samuel Lavery <sam.lavery@gmail.com>
CC: Tony Arcieri <bascule@gmail.com>, Derek Atkins <datkins@veridify.com>, cpeikert@alum.mit.edu, mike.ounsworth@entrust.com, pqc-forum@list.nist.gov, pqc@ietf.org, sfluhrer@cisco.com, cfrg@irtf.org
Subject: Re: [pqc-forum] Covert channel attack means signing should be deterministic
Date: Friday, September 30, 2022 06:25:44 PM ET

My top level point is that this particular concern is something we should address once in standards space and apply to all the algorithms we accept rather than accepting whatever approach the individual algorithms picked.

If that results in someone saying 'hey I need X because of Y', well, that is a valuable piece of data we just picked up. If we are considering HOTDOG vs BURGER and they can't use a common module, hey I want to know that and I want to know the reason.

My secondary point is about modularization of systems. As Tony reminds us, it was the fault attack issue that drove us to non-deterministic signing in the first place. I am not disputing that constraint but I also know that state actors have run bogus crypto HSM companies in the past and next time it might not be a friendly govt. that does it. So the covert channel issue is an equal concern for me.

What this comes down to is that for my most security sensitive applications, I am going to be looking for devices that are modularized in such a fashion that I can buy 10 devices, pick two at random, qualify them and use the other 8 for production. The qualification process is going to be destructive insofar as I am not going to be able to use them in FIPS mode any more.

For Ed448, I can achieve the same result non destructively by applying threshold. Instead of relying on one vendor, I pick two and now Mallet has to compromize both.

On Thu, Sep 29, 2022 at 12:05 AM Samuel Lavery <sam.lavery@gmail.com> wrote:

> My understanding is that for properly partitioned signature schemes, the additional entropy is “applied” to the partition that isn’t dependent on the message. One also includes this value in the signature for the verifier to use.
> This is the Boneh-Shen-Waters transform, it’s a way to turn EUF-CMA schemes into sEUF-CMA. SPHINCS+ achieves this via OptRand. I do something similar in mine. It’s entropy taken at signing time and shared. The scheme is still deterministic, but gives you 256 or whatever unique signatures per message without leaking any extra secret bits.
>
> Cheers,

Sam

On Sep 28, 2022, at 11:35 AM, Tony Arcieri <bascule@gmail.com> wrote:

On Wed, Sep 28, 2022 at 11:08 AM Phillip Hallam-Baker <phill@hallambaker.com> wrote:

> OK, so this seems to make the argument that a signing scheme MUST be deterministic. Otherwise, we can never trust the HSMs not to be leaking their keys.

The typical counterargument against deterministic signature schemes is that they enable fault attacks, and lattice-based schemes are no exception to this:

https://tches.iacr.org/index.php/TCHES/article/view/7267

Trying to work around such attacks by adding some sort of post-signature check, such as signing twice and comparing results, or ensuring that the signature verifies correctly, can be defeated with a double fault attack which bypasses whatever branch prevents disclosing the faulty signature.

Furthermore, mandating a deterministic scheme doesn't necessarily close the covert channel if there's no way for a verifier to determine the signature was produced correctly under a prescribed deterministic method.

--
Tony Arcieri