From: Blumenthal, Uri - 0553 - MITLL <uri@ll.mit.edu> via pqc-forum@list.nist.gov
To: Mike Ounsworth <mike.ounsworth=40entrust.com@dmarc.ietf.org>, Mike Ounsworth <mike.ounsworth@entrust.com>, pqc-forum <pqc-forum@list.nist.gov>
CC: 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 05:53:25 PM ET
Attachments: smime.p7m

My read:

- Option 0: no-go in 99% of the cases;
- Option 1: should be acceptable in 95+% of the cases;
- Option 2: absolutely no-go;
- Option 4: an "accessorized" version of (1), probably the best, as each protocol can decide what "accessories" it wants for the "envelope".

TNX

--

V/R,

Uri

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

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

- C. A. R. Hoare

From: CFRG on behalf of Mike Ounsworth
Date: Monday, September 19, 2022 at 17:18
To: Mike Ounsworth , pqc-forum
Cc: "pqc@ietf.org" , CFRG
Subject: Re: [CFRG] Design rationale for keyed message digests in SPHINCS+, Dilithium, FALCON?

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
Sent: September 18, 2022 2:42 PM
To: pqc-forum
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