From:	Davidson, Michael S. (Fed)
To:	Kelsey, John M. (Fed); Dworkin, Morris J. (Fed)
Cc:	Cooper, David (Fed); Dang, Quynh H. (Fed); Miller, Carl A. (Fed); Apon, Daniel C. (Fed)
Subject:	RE: Preventing reuse of one-time signing keys in stateful hash-based signatures
Date:	Wednesday, February 27, 2019 12:49:31 PM

Yeah, I'm not sure there's a good way of proving malfeasance, but if the signer can in some sense commit to the next pi value to use in signing 0, and then on signing 1 the helper passes back to the signer some proof of the signer's own intentions….well, "thinking aloud", I'm not sure how much this even helps, since the "next" pi might have already been used, and the helper can probably replay old "proofs" anyways. Never mind!

I can't help but feel like this would be a good use for a blockchain: updating the private key requires an on-chain state-change that logs pi, so if users perform backups of their private key, they can re-scan the chain to validate the proper index to use.

-Michael

From: Kelsey, John M. (Fed)
Sent: Wednesday, February 27, 2019 12:24 PM
To: Davidson, Michael S. (Fed) <michael.davidson@nist.gov>; Dworkin, Morris J. (Fed) <morris.dworkin@nist.gov>
Cc: Cooper, David A. (Fed) <david.cooper@nist.gov>; Dang, Quynh (Fed) <quynh.dang@nist.gov>; Miller, Carl A. (Fed) <carl.miller@nist.gov>; Apon, Daniel C. (Fed) <daniel.apon@nist.gov>
Subject: Re: Preventing reuse of one-time signing keys in stateful hash-based signatures

I can see how to prove that keys up until pi have been used (just show the signature for pi), but I don't see how to prove that keys past some pi haven't been used.

Imagine I'm a helper and I know that key pi + 1 has been used. I could just make a simulated version of myself that has the state I had when it hadn't yet been used, and respond to any query by giving that simulated version's response. So I don't think there's any way to do this directly.

Could we force the helper to send something that would be evidence of malfeasance? That is, I have to send something that can be used later to prove that I've intentionally forced you to reuse a key? Again, it seems like the problem is that we know that errors can happen, so the helper could make an error (say by having its state reset to some earlier state by the cloud provider).

--John

From: "Davidson, Michael S. (Fed)" <michael.davidson@nist.gov>
Date: Wednesday, February 27, 2019 at 12:07
To: "Kelsey, John M. (Fed)" <john.kelsey@nist.gov>, "Dworkin, Morris J. (Fed)" <morris.dworkin@nist.gov>
Cc: "Cooper, David A. (Fed)" <david.cooper@nist.gov>, "Dang, Quynh (Fed)"

<quynh.dang@nist.gov>, "Miller, Carl A. (Fed)" <carl.miller@nist.gov>, "Apon, Daniel C. (Fed)" <daniel.apon@nist.gov>
Subject: RE: Preventing reuse of one-time signing keys in stateful hash-based signatures

Interesting, thanks for the corrections!

>> Just establish two shared symmetric keys offline

It would seem this largely reduces the situations where it is possible to ones where the signer/helper have a preexisting relationship or physical proximity. Despite being a limitation, this actually makes me feel better about the scheme, because these situations are where it is least risky anyways.

>> The helper can't force key reuse—the signer chooses which key to use, and the helper just approves or disapproves it and sends along the tree path.

The helper can't force key reuse ***arbitrarily***, but in the situations where the signer gets pi wrong (precisely the situations that we are concerned about), the helper can say the one-time key is kosher and supply the valid path, "forcing" key reuse in this situation. That's why I would view the helper as a trusted party that can compromise the security of the scheme. That said, this isn't any worse than where we are without the helper, except for 1) potentially creating a false sense of security, and 2) compromise of the helper w/o a corresponding compromise of the signer. This seems to be an improvement overall, if the helper is considered trustworthy and is well-protected.

Can we come up with a proof scheme where the helper can prove to the signer that a one-time key is safe to use? The following doesn't work well for a few reasons, but what I have in mind is that the signer could sign the next pi value they intend to use, pass the signature to the helper, who simply passes it back unaltered with the next query. The signer could then verify his own intentions.

Regards,
Michael

From: Kelsey, John M. (Fed)
Sent: Wednesday, February 27, 2019 11:38 AM
To: Davidson, Michael S. (Fed) <michael.davidson@nist.gov>; Dworkin, Morris J. (Fed) <morris.dworkin@nist.gov>
Cc: Cooper, David A. (Fed) <david.cooper@nist.gov>; Dang, Quynh (Fed) <quynh.dang@nist.gov>; Miller, Carl A. (Fed) <carl.miller@nist.gov>; Apon, Daniel C. (Fed) <daniel.apon@nist.gov>
Subject: Re: Preventing reuse of one-time signing keys in stateful hash-based signatures

Michael,

a. We can trivially do the secure connection between the helper and signer using only symmetric crypto, which is quantum safe as far as anyone knows. Just establish two shared symmetric keys offline, and then use GCM with different keys for messages going different directions.

b. The helper can't force key reuse—the signer chooses which key to use, and the helper just approves or disapproves it and sends along the tree path. If the signer has a glitch and tries to reuse a key, then the helper can save the day, but nothing the helper does can force the signer to reuse a key.
c. It's possible to set the helper up so that it never learns ***anything*** about the message being signed, or which user is signing it. That's a more complicated protocol, though.

--John

From: "Davidson, Michael S. (Fed)" <michael.davidson@nist.gov>
Date: Wednesday, February 27, 2019 at 10:26
To: "Dworkin, Morris J. (Fed)" <morris.dworkin@nist.gov>, "Kelsey, John M. (Fed)" <john.kelsey@nist.gov>
Cc: "Cooper, David A. (Fed)" <david.cooper@nist.gov>, "Dang, Quynh (Fed)" <quynh.dang@nist.gov>, "Miller, Carl A. (Fed)" <carl.miller@nist.gov>, "Apon, Daniel C. (Fed)" <daniel.apon@nist.gov>
Subject: RE: Preventing reuse of one-time signing keys in stateful hash-based signatures

Hi John,

It seems like this requires the ability to make secure connections to some outside party, and we don't have any quantum-secure, standardized way of doing that (and probably won't for another few years). I'm not sure to what extent that matters, but if describing this mitigation in our standard, we would probably want to be extremely careful with wording for this reason. Presumably, we would need a post-quantum PKI for this to be secure, and it would be self-referential if the PKI were based on stateful-HBS itself

I personally don't much like the idea of adding a trusted third party to the protocol. A malicious third party, or a compromised-but-otherwise-honest third party could force the signer to reuse a key. I'm more likely to get behind the idea if the "helper" is more like a module separate from the "signer", but where both the helper and signer are controlled by the same entity. If the signing module polls the helper module that is on a server protected within my internal network, I'm far more comfortable with this kind of risk.

Regards,
Michael

From: Dworkin, Morris J. (Fed)
Sent: Wednesday, February 27, 2019 9:34 AM
To: Kelsey, John M. (Fed) <john.kelsey@nist.gov>
Cc: Cooper, David A. (Fed) <david.cooper@nist.gov>; Dang, Quynh (Fed) <quynh.dang@nist.gov>; Miller, Carl A. (Fed) <carl.miller@nist.gov>; Davidson, Michael S. (Fed) <michael.davidson@nist.gov>; Apon, Daniel C. (Fed) <daniel.apon@nist.gov>
Subject: Re: Preventing reuse of one-time signing keys in stateful hash-based signatures

Thanks very much, John; looks interesting. I'm copying the HBS working group on this message so they can weigh in too. We'll consider this for the special pub.

Morrie

> On Feb 26, 2019, at 11:22 AM, Kelsey, John M. (Fed) <john.kelsey@nist.gov> wrote:
>
> Morrie,
>
> I wanted to give you some feedback about my ideas for preventing accidental or intentional reuse of one-time keys in a stateful hash-based signature scheme. My basic idea is to have an online
> "helper" which has to interact with the signer to get a final one-time signature, and then use that helper to prevent accidental or even intentional reuse of a key.
>
> The interesting problems here are:
>
> a. How do we minimize the power/knowledge of the helper?
> b. How do we minimize the memory and other resources needed by both the helper and the signer?
> c. How do we deal with the signer or helper being offline?
>
> The simplest way I can see to do this works like this:
>
> For an HBS scheme, you need a unique identifier for each one-time key. In LMS, this consists of a "tree ID" (the ID of the whole big composite key), and a "key ID" (the ID of the one-time signature). I'm going to call the unique identifier per one-time key pi, and assume that the low 64 bits of pi is the counter for keys in this tree. The signer keeps track of pi, and increments it each time he signs something. The risk is that the signer may either accidentally or intentionally reuse a value of pi.
>
> HBS schemes have a lot of secret values. To avoid having crazy storage requirements, those are usually all generated from a single secret key, using some kind of PRF. I'm going to assume we have a PRF so that PRF(K,input) gives me a 256-bit output that works the way we'd expect a PRF to work.
>
> A big practical problem with implementing an HBS is that for each signature, you need to construct a tree path for the one-time key used. If you don't want to store the whole set of hashes of one-time keys, this requires a fair bit of extra complexity in the implementation—you end up either having a lot of computation to do for some signatures (where you have to regenerate half the tree, say), or having some clever algorithm for doing precomputation to spread that work out among many signatures.

Let PATH(pi) be the path to one-time key pi. Let K = the PRF key held by the signer.

So, here's the simplest scheme I can think of:

The helper holds a database indexed by each possible value of pi, and stores PATH(pi) and whether or not this key has been used before.

To sign, the signer does these steps:

a. Determine the next value of pi.
b. Send pi to the helper.
c. Get back either "FAIL" (if the key is being reused) or PATH(pi) (if the key isn't being reused).
d. Update the stored value of pi.
e. Use K to rederive the secret values of the one-time key.
f. Construct the signature.
g. Produce the signature along with PATH(pi).

The interesting properties here are:

1. The helper never learns what is being signed, so while it can deny service to the signer, it can't veto a particular thing being signed.
2. The signer can't sign anything without the helper's help unless it does some extra computation and stores some extra data.
3. The signer needs almost no memory and doesn't have to rederive the tree.
4. The helper needs a lot of memory.

It would be easy to split this among two helpers—one handles even-numbered versions of pi, the other handles odd-numbered versions. This would let the signer keep signing even if one of the helpers was unavailable.

There are more complicated versions of this idea that will prevent even a signer who wants to reuse keys from reusing a key.

Comments?

--John