Anti-Exfiltration for EC Signatures

Andrew Poelstra
Director, Blockstream Research
May 3, 2023

EC Signatures

Valid EC signatures are linear equations in two secret variables:

$$s = kG + e \cdot xG$$

One equation, two unknowns:

- (permanent) secret key
- (ephemeral) secret nonce

Nonce Reuse

- Reusing an nonce immediately gives "two equations, two unknowns" which can be solved for the secret key
- Even slight deviations from uniform can be solved (Henninger/Breitner 2019)
- Deviations can be hidden so that only a specific attacker can exploit them

It is essential that nonces be generated uniformly at random! But if a hardware wallet is generating the randomness, a user has no way to verify this.

Solutions

- Deterministic nonces (RFC6979) prevent accidental nonce bias/reuse.
- But provide no way for the user to verify whether it was used.

Solutions

- ZKP's could provide assurance that DN was used (NSRW 2020, "Musig-DN")
- But ZKPs are verify expensive to run on limited hardware.
- Also complex, have more room for implementation faults
- And anyway typical ZKPs have their own nonces that could be biased!

Solutions

- Multisigning with the host computer would re-randomize the nonce
- But requires the host manage a key (or user manage a passphrase)
- Needs to be designed with nonce de-biasing in mind
- Implementation complexity

Solutions

- But the multisig idea is basically the right idea
- Suppose the host provides only a nonce contribution, not a key contribution (so not really multisig)
- This contribution can be random and thrown away after use

Anti-Exfiltration

- Our solution is called anti-exfil
- The host provides a random challenge; the HWW tweaks its nonce to commit to the challenge; the host verifies the tweak
- The tweaking completely re-randomizes the nonce, eliminating any bias
- As long as an attacker hasn't compromised the HWW **and** the host, he cannot extract any information

[bonus] Technical Problems

Two-party signature construction schemes need to avoid several pitfalls of naive implementations:

- If host provides randomness first, can the HWW grind its untweaked nonce to bias the final nonce?
- If HWW provides an untweaked nonce first, can the *host* bias the nonce?
- If the HWW goes first, and is deterministic, can the host ask for two signatures with different tweak,s extracting the secret key?
- Can the host verify that the tweaking was done correctly (the whole point of this scheme :))?

[bonus] Technical Solution

These problems are solved by the following protocol:

- The host chooses random data and sends a *commitment* to the HWW.
- The HWW feeds this commitment, with its secret key and message, into a deterministic nonce function to produce an untweaked nonce. It sends this nonce to the host.
- The host sends the actual randomness to the HWW.
- The HWW verifies the randomness matches the commitment, then tweaks its nonce (using $P \mapsto P + H(P \| r)$), and generates a signature.
- The host verifies that the resulting signature uses the correct tweaked nonce.

Thank you

More information, and links to implementations, are at
`https://blog.blockstream.com/anti-exfil-stopping-key-exfiltration/`

A toy implementation/example from 2017 can be seen at
`https://github.com/opentimestamps/python-opentimestamps/pull/14`

I am Andrew Poelstra `andrew@blockstream.com`