A threshold ECDSA protocol: its design and implementation

Victor Shoup

(DFINITY)

Overview

The **Internet Computer** is a distributed platform for secure execution of smart contracts developed by **DFINITY**

The Internet Computer provides excellent **scalability** and a **complete protocol stack** for distributed apps (both front-end and back-end)

We are currently implementing a **new threshold ECDSA signing protocol**

This protocol is being integrated into the architecture of the Internet Computer

This enables smart contracts running on the Internet Computer to securely hold and spend Bitcoin and other cryptocurrencies

Two papers:

- Groth, Shoup 2021: On the security of ECDSA with additive key derivation and presignatures
- Groth, Shoup 2022: Design and analysis of a distributed ECDSA signing service

ECDSA signatures

$\boxed{s\mathcal{R} = h\mathcal{G} + t\mathcal{D}}$

E is an elliptic curve of order q generated by $\mathcal{G} \in E$

Secret key: $d \leftarrow_R \mathbb{Z}_q^*$

Public key: $\mathcal{D} \leftarrow d\mathcal{G} \in E^*$

Sign message m:

```
$h \leftarrow Hash(m) \in \mathbb{Z}_q$
$r \leftarrow_R \mathbb{Z}_q^*$,  $\mathcal{R} \leftarrow r\mathcal{G} \in E$,  $t \leftarrow C(\mathcal{R}) \in \mathbb{Z}_q$
if $t = 0$ or $h + td = 0$ then return *fail*
$s \leftarrow r^{-1}(h + td)$
return the signature $(s, t) \in \mathbb{Z}_q^* \times \mathbb{Z}_q^*$
```

Verify signature $(s, t) \in \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ on m:

```
$h \leftarrow Hash(m) \in \mathbb{Z}_q$
$\mathcal{R} \leftarrow s^{-1}h\mathcal{G} + s^{-1}t\mathcal{D}$
check that $\mathcal{R} \neq \mathcal{O}$ and $C(\mathcal{R}) = t$
```

Precomputation

$\boxed{s\mathcal{R} = h\mathcal{G} + t\mathcal{D}}$

The computation

$$r \leftarrow_R \mathbb{Z}_q^*, \ \mathcal{R} \leftarrow r\mathcal{G} \in E, \ t \leftarrow C(\mathcal{R}) \in \mathbb{Z}_q$$

does not depend on m and can be precomputed

The value $\mathcal{R} = r\mathcal{G}$ is called a presignature

In a threshold implementation, we can also precompute "sharings" $[r]$, $[u]$, $[r'] = [ru]$, $[d'] = [du]$, where $u \leftarrow_R \mathbb{Z}_q$

To sign a given m, we only need to locally compute

$$h \leftarrow Hash(m), \quad [v] \leftarrow h[u] + t[d']$$

and then "open" $[v]$ and $[r']$ to compute

$$s = \frac{v}{r'} = \frac{hu + tdu}{ru} = \frac{h + td}{r}$$

Given the presignature plus preshared data, latency for a signature is just *one communication round*

Additive key derivation

$\boxed{s\mathcal{R} = h\mathcal{G} + t\mathcal{D}}$

Idea: replace secret key d by $d + e$ and public key by $\mathcal{D} + e\mathcal{G}$

$e \in \mathbb{Z}_q$ is an "additive tweak" derived by hashing some public ID

BIP32: a special case of this, used by bitcoin and other cryptocurrencies to implement "hierarchical deterministic wallets"

In a threshold implementation:

- easy to *efficiently implement* (due to linearity)
- useful to keep *key maintenance costs* to a minimum:
  - each signing key must be shared among all parties
  - each key must be reshared whenever
    - the network membership changes, or
    - a proactive security refresh occurs
  - each key may need to be backed up

New verification equation: $s\mathcal{R} = (h + te)\mathcal{G} + t\mathcal{D}$

ECDSA security

$\boxed{s\mathcal{R} = (h + te)\mathcal{G} + t\mathcal{D}}$

Security of threshold ECDSA reduces to security of ECDSA

But . . . if we use presigs and/or additive key derivation (AKD), need to adjust attack game accordingly

What’s previously known:

- **ECDSA:** secure in Generic Group Model (GGM) assuming *Hash* is collision resistant and random preimage resistant [Brown02]
- **ECDSA with presigs:** secure in GGM plus Random Oracle Model (ROM) [CMP20]
- **ECDSA with AKD:** no general results (but some results with various restrictions on attack)
- **ECDSA with presigs *and* AKD:** never really looked at (even though this mode of operation has been advocated for)

Presigs + AKD: an attack!

$\boxed{s\mathcal{R} = (h + te)\mathcal{G} + t\mathcal{D}}$

1. Make one presig query to get $\mathcal{R}$ and let $t := C(\mathcal{R})$.
2. Find m, e, m^*, e^* such that
$$h + te = h^* + te^*,$$
where $e \neq e^*$ and $h := Hash(m)$ and $h^* := Hash(m^*)$
3. Ask for a signature (s, t) using this presignature on message m with tweak e

Then
$$s\mathcal{R} = (h + te)\mathcal{G} + t\mathcal{D} = (h^* + te^*)\mathcal{G} + t\mathcal{D}$$
which means that (s, t) is also a valid signature on m^* with respect to e^*

Step 2 is essentially a *4-sum problem*

Using Wagner's 4-sum algorithm [Wagner02], attack takes time $O(q^{1/3})$

This beats the $O(q^{1/2})$ time needed to break ECDSA generically

Presigs + AKD: mitigations

$\boxed{s\mathcal{R} = (h + te)\mathcal{G} + t\mathcal{D}}$

- **Re-randomized presigs**

  A "base presignature" $\mathcal{R}' := r'\mathcal{G}$ precomputed as before

  When signing request is made, $\mathcal{R}'$ is replaced by $\mathcal{R} := \mathcal{R}' + \delta\mathcal{G}$, where $\delta \in \mathbb{Z}_q$ is a *public* value that is pseudo-randomly generated *at that time*

  In a threshold implemention:

  - easy to efficiently implement (due to linearity)
  - *may* introduce some additional latency
  - on the Internet Computer, it *does not* (by the time the signing request passes through consensus, δ is already available)

- **Homogeneous key derivation**

  master secret key: $(d, d') \in \mathbb{Z}_q \times \mathbb{Z}_q$ / master public key: $(\mathcal{D}, \mathcal{D}') := (d\mathcal{G}, d'\mathcal{G})$

  For tweak $e \in \mathbb{Z}_q$, derived secret key: $d + ed'$ / derived public key: $\mathcal{D} + e\mathcal{D}'$

  Disadvantage: not compatible with BIP32

  Not (currently) implemented on Internet Computer

- **Re-randomized presigs** + **homogeneous key derivation**
- **Analysis:** Our paper [Groth, Shoup 2021] analyzes ECDSA and these variants (all 9 of them!) in GGM under concrete assumptions on *Hash*

A threshold ECDSA protocol

Our paper [Groth, Shoup 2022] presents a concrete threshold ECDSA protocol with these properties:

- it works in an **asynchronous** communication model
- it works with n parties with up to $f < n/3$ **Byzantine corruptions**
- it provides **guaranteed output delivery**
- it provides a very efficient, **non-interactive online signing phase**
- it supports **BIP32-style additive key derivation**

Other properties:

- It uses re-randomized presigs + AKD
- Every smart contract on the Internet Computer effectively gets its own signing key (via AKD)
- It employs a new *Asynchronous Verifiable Secret Sharing (AVSS)* scheme
- It relies on the Internet Computer’s consensus mechanism (but any consensus mechanism could be used)
- It relies on the Internet Computer’s “Random Beacon” to implement re-randomized presigs

Comparison to other recent works

There has been a flurry of threshold ECDSA protocols in the past few years

Essentially all of these:

- assume **synchronous communication**, and
- are **not fault tolerant**: *single node crashes* $\implies$ *no signatures*

Neither of these properties are compatible with the Internet Computer:

- Synchronous communication is an unrealistic assumption for a globally distributed network
  - unless extremely pessimistic and impractical time-outs are used
- Fault tolerance is essential
  - smart contracts must be “unstoppable”

Thank you!

And come join as at DFINITY!!

. . . we are looking for good people who want to work at the intersection of theory and practice

Reach out to me or Jens Groth who is at RWC IRL