On the security of ECDSA with additive key derivation and presignatures

Jens Groth
Victor Shoup

(DFINITY)

Context

Context

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

Context

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)

Context

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**

Context

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

Context

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

Context

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

Companion paper:

- Groth, Shoup 2022: Design and analysis of a distributed ECDSA signing service (eprint)

ECDSA signatures

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

ECDSA signatures

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

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

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

ECDSA signatures

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 $(s, t) \in \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

ECDSA signatures

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

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

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

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

Precomputation

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

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

Precomputation

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

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 $u \leftarrow_R \mathbb{Z}_q$

Precomputation

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

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

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

Precomputation

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

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 (AKD)

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

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

Additive Key Derivation (AKD)

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

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

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

Additive Key Derivation (AKD)

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

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

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

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

Additive Key Derivation (AKD)

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

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

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

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

In a threshold implementation:

Additive Key Derivation (AKD)

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

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

$e \in \mathbb{Z}_q$ is a *public* "additive tweak" derived by hashing some 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)

Additive Key Derivation (AKD)

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

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

$e \in \mathbb{Z}_q$ is a *public* "additive tweak" derived by hashing some 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:

Additive Key Derivation (AKD)

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

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

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

Additive Key Derivation (AKD)

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

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

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

Additive Key Derivation (AKD)

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

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

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

Additive Key Derivation (AKD)

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

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

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

Additive Key Derivation (AKD)

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

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

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

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

Security of threshold ECDSA reduces to security of ECDSA
(typically)

ECDSA security

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

Security of threshold ECDSA reduces to security of ECDSA (typically)

But . . . if we use presigs and/or AKD, need to adjust attack game accordingly

ECDSA security

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

Security of threshold ECDSA reduces to security of ECDSA (typically)

But . . . if we use presigs and/or AKD, need to adjust attack game accordingly

What's previously known:

ECDSA security

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

Security of threshold ECDSA reduces to security of ECDSA (typically)

But . . . if we use presigs and/or 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 [Brown 2002]

ECDSA security

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

Security of threshold ECDSA reduces to security of ECDSA (typically)

But . . . if we use presigs and/or 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 [Brown 2002]
- **ECDSA with presigs:** secure in GGM plus Random Oracle Model (ROM) [Canetti, Makriyannis, Peled 2020]

ECDSA security

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

Security of threshold ECDSA reduces to security of ECDSA (typically)

But . . . if we use presigs and/or 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 [Brown 2002]
- **ECDSA with presigs:** secure in GGM plus Random Oracle Model (ROM) [Canetti, Makriyannis, Peled 2020]
- **ECDSA with AKD:** no general results (but some results with various restrictions on attack)

ECDSA security

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

Security of threshold ECDSA reduces to security of ECDSA (typically)

But . . . if we use presigs and/or 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 [Brown 2002]
- **ECDSA with presigs:** secure in GGM plus Random Oracle Model (ROM) [Canetti, Makriyannis, Peled 2020]
- **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)

Presigs + AKD: an attack!

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

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

Presigs + AKD: an attack!

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

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

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

Presigs + AKD: an attack!

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

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

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

Presigs + AKD: an attack!

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

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^*

Presigs + AKD: an attack!

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

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*

Presigs + AKD: an attack!

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

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 $O(q^{1/3})$

Presigs + AKD: an attack!

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

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

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

- **Re-randomized presigs**

Presigs + AKD: mitigations

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

- **Re-randomized presigs**

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

Presigs + AKD: mitigations

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

- **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*

Presigs + AKD: mitigations

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

- **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:

Presigs + AKD: mitigations

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

- **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)

Presigs + AKD: mitigations

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

- **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

Presigs + AKD: mitigations

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

- **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)

Presigs + AKD: mitigations

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

- **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**

Presigs + AKD: mitigations

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

- **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: $(d, d') \in \mathbb{Z}_q \times \mathbb{Z}_q$

Presigs + AKD: mitigations

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

- **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: $(\mathcal{D}, \mathcal{D}') := (d\mathcal{G}, d'\mathcal{G})$

Presigs + AKD: mitigations

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

- **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: $\mathcal{D} + e\mathcal{D}'$

Presigs + AKD: mitigations

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

- **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

Presigs + AKD: mitigations

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

- **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

Presigs + AKD: mitigations

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

- **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**

	no presigs	presigs	re-randomized presigs

	no presigs	presigs	re-randomized presigs
no derivation			

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$		

- $N =$ # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	

- $N = \#$ of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- $U = \#$ of "unused" presigs
- ⊠: insecure with "raw" signing queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation			

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$		

- $N =$ # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- $U =$ # of "unused" presigs
- ⊠: insecure with "raw" signing queries
- $\mathfrak{E}$: set of allowed tweaks (viewing tweak derivation as Random Oracle)

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + N_{\text{psig}}\,\mathcal{E}_{\text{4sum1}} + N\mathcal{E}_{\text{4sum2}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries
- $\mathfrak{E}$: set of allowed tweaks (viewing tweak derivation as Random Oracle)
- $\mathcal{E}_{\text{4sum1}}, \mathcal{E}_{\text{4sum2}}$: specialized 4-sum problems
- N_{psig} = # of presig queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + N_{\text{psig}}\,\mathcal{E}_{\text{4sum1}} + N\mathcal{E}_{\text{4sum2}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries
- $\mathfrak{E}$: set of allowed tweaks (viewing tweak derivation as Random Oracle)
- $\mathcal{E}_{\text{4sum1}}, \mathcal{E}_{\text{4sum2}}$: specialized 4-sum problems
- N_{psig} = # of presig queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + N_{\text{psig}}\,\mathcal{E}_{\text{4sum1}} + N\mathcal{E}_{\text{4sum2}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
homogeneous derivation			

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries
- $\mathfrak{E}$: set of allowed tweaks (viewing tweak derivation as Random Oracle)
- $\mathcal{E}_{\text{4sum1}}, \mathcal{E}_{\text{4sum2}}$: specialized 4-sum problems
- N_{psig} = # of presig queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + N_{\text{psig}}\,\mathcal{E}_{\text{4sum1}} + N\mathcal{E}_{\text{4sum2}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
homogeneous derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$		

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries
- $\mathfrak{E}$: set of allowed tweaks (viewing tweak derivation as Random Oracle)
- $\mathcal{E}_{\text{4sum1}}, \mathcal{E}_{\text{4sum2}}$: specialized 4-sum problems
- N_{psig} = # of presig queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + N_{\text{psig}}\,\mathcal{E}_{\text{4sum1}} + N\mathcal{E}_{\text{4sum2}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
homogeneous derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries
- $\mathfrak{E}$: set of allowed tweaks (viewing tweak derivation as Random Oracle)
- $\mathcal{E}_{\text{4sum1}}, \mathcal{E}_{\text{4sum2}}$: specialized 4-sum problems
- N_{psig} = # of presig queries

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive derivation	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + N_{\text{psig}}\,\mathcal{E}_{\text{4sum1}} + N\mathcal{E}_{\text{4sum2}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
homogeneous derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ ⊠	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$

- N = # of queries (group, signing, etc)
- $\mathcal{E}_{\text{cr}}$: collision resistance
- $\mathcal{E}_{\text{rpr}}$: random preimage resistance (given random ρ, find m: $Hash(m) = \rho$)
- $\mathcal{E}_{\text{zpr}}$: zero preimage resistance (find m: $Hash(m) = 0$)
- $\mathcal{E}_{\text{rr}}$: ratio resistance (given random ρ, find m, m': $Hash(m)/Hash(m') = \rho$)
- U = # of "unused" presigs
- ⊠: insecure with "raw" signing queries
- $\mathfrak{E}$: set of allowed tweaks (viewing tweak derivation as Random Oracle)
- $\mathcal{E}_{\text{4sum1}}, \mathcal{E}_{\text{4sum2}}$: specialized 4-sum problems
- N_{psig} = # of presig queries

Other features of our analysis

- We explicitly model the “encoding space” in GGM as an elliptic curve of prime order q — the EC-GGM model

Other features of our analysis

- We explicitly model the "encoding space" in GGM as an elliptic curve of prime order q — the EC-GGM model
- The encoding is random, except that it respects the inverse law $-(x, y) = (x, -y)$

Other features of our analysis

- We explicitly model the "encoding space" in GGM as an elliptic curve of prime order q — the EC-GGM model
- The encoding is random, except that it respects the inverse law $-(x, y) = (x, -y)$
- This allows us to model ECDSA "quirks" such as "malleability": *if* (s, t) *is a signature on* m*, then so is* $(-s, t)$

Other features of our analysis

- We explicitly model the "encoding space" in GGM as an elliptic curve of prime order q — the EC-GGM model
- The encoding is random, except that it respects the inverse law $-(x, y) = (x, -y)$
- This allows us to model ECDSA "quirks" such as "malleability": *if* (s, t) *is a signature on* m*, then so is* $(-s, t)$
- We prove that ECDSA is strongly unforgeable "up to sign"

Other features of our analysis

- We explicitly model the "encoding space" in GGM as an elliptic curve of prime order q — the EC-GGM model
- The encoding is random, except that it respects the inverse law $-(x, y) = (x, -y)$
- This allows us to model ECDSA "quirks" such as "malleability": *if* (s, t) *is a signature on* m*, then so is* $(-s, t)$
- We prove that ECDSA is strongly unforgeable "up to sign"
- We analyze the BIP32 construction and show that it is indifferentiable from a "public" Random Oracle [Dodis, Ristenpart, Shrimpton 2009]

Other features of our analysis

- We explicitly model the "encoding space" in GGM as an elliptic curve of prime order q — the EC-GGM model
- The encoding is random, except that it respects the inverse law $-(x, y) = (x, -y)$
- This allows us to model ECDSA "quirks" such as "malleability": *if* (s, t) *is a signature on* m*, then so is* $(-s, t)$
- We prove that ECDSA is strongly unforgeable "up to sign"
- We analyze the BIP32 construction and show that it is indifferentiable from a "public" Random Oracle [Dodis, Ristenpart, Shrimpton 2009]
- We also analyze AKD under concrete security assumptions (rather than using Random Oracles)

Thank you!

Thank you!

And come join as at DFINITY!!

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