Standardizing Protocols for Threshold ECDSA

Jonathan Katz
Chief Scientist, Dfns

Sept. 26, 2023
MPTS 2023: NIST Workshop on Multi-Party Threshold Schemes 2023

Thanks to Denis Varlakov, Nik Sorokovikov, and Antoine Urban for helpful discussions

Overview of the talk

- Threshold cryptography (signing) in a “key-management network”
  - Applies to schemes beyond ECDSA

Overview of the talk

- Threshold cryptography (signing) in a “key-management network”
  - Applies to schemes beyond ECDSA
- Standardizing threshold ECDSA protocols
  - No-honest-majority setting
  - Honest-majority setting

Key-management network

- Most (all?) treatments of threshold cryptography in the literature assume a single user distributing a key among n parties
  - Users act independently, and may choose different sets of parties
  - Even if users choose (some of) the same parties, protocol executions for different users' keys are considered in isolation

Key-management network

- Most (all?) treatments of threshold cryptography in the literature assume a single user distributing a key among n parties
  - Users act independently, and may choose different sets of parties
  - Even if users choose (some of) the same parties, protocol executions for different users' keys are considered in isolation
- Key-management network: a dedicated set of n parties holding shares of multiple keys on behalf of multiple users

Key-management network

- Most (all?) treatments of threshold cryptography in the literature assume a single user distributing a key among n parties
  - Users act independently, and may choose different sets of parties
  - Even if users choose (some of) the same parties, protocol executions for different users' keys are considered in isolation
- Key-management network: a dedicated set of n parties holding shares of multiple keys on behalf of multiple users
- Technical advantages:
  - Each party's state can be shared across protocol executions involving different keys
  - Possibility of parallelization/batch processing across keys

Example

The robust KeyGen protocol I described previously

Suggestions

- Proposers should be encouraged to highlight potential optimizations of their protocols when run in a key-management network
- Schemes should be evaluated (among other factors) based on their performance in a "key-management network" setting

(Threshold) ECDSA

ECDSA

- $\mathbb{G}$ is a cyclic group of prime order q, with generator g
- Private key $x \in \mathbb{Z}_q$; public key $y = g^x$
- To sign a (hashed) message m:
  - Choose $k \leftarrow \mathbb{Z}_q$; compute $R := g^k$ and $r := F(R)$
  - Compute $s := k^{-1} \cdot (m + rx)$

(Threshold) ECDSA

ECDSA

- $\mathbb{G}$ is a cyclic group of prime order q, with generator g
- Private key $x \in \mathbb{Z}_q$; public key $y = g^x$
- To sign a (hashed) message m:
  - Choose $k \leftarrow \mathbb{Z}_q$; compute $R := g^k$ and $r := F(R)$
  - Compute $s := k^{-1} \cdot (m + rx)$

Threshold ECDSA

- n is the total number of parties
- t is an upper bound on the number of corrupted parties
- Honest majority: $t < n/2$; no-honest majority: $n/2 \leq t < n$

Threshold ECDSA in the no-honest-majority setting

Will focus on the CGGMP protocol

- Goal is *not* to present the protocol in detail
- Will highlight some optimizations/issues that arise in a key-management network setting

We would be interested in collaborating on a submission to NIST

- Is it possible to merge with a DKLS submission?

CGGMP protocol

CGGMP protocol offers

- Support for any $t < n$
- Presigning + one-round online signing
- Universally composable
- Security for adaptive adversaries
- Can incorporate identifiable abort

CGGMP protocol (high level)

Key generation and provisioning

- Run DKG protocol to generate shares of a private key (denoted $[x]_t$)
- Each party P_i generates a Paillier key N_i, values $s_i, t_i \in \mathbb{Z}^*_{N_i}$, and ZK proofs of various properties of those parameters

CGGMP protocol (high level)

Key generation and provisioning

- Run DKG protocol to generate shares of a private key (denoted $[x]_t$)
- Each party P_i generates a Paillier key N_i, values $s_i, t_i \in \mathbb{Z}^*_{N_i}$, and ZK proofs of various properties of those parameters

Signing

- Generate random $[k^{-1}]_t$, $[a]_t$
- Compute $[ak^{-1}]_t$ and $[xk^{-1}]_t$ using a multiplication protocol
- Reconstruct g^a and ak^{-1}; use these to compute g^k and $r := F(g^k)$
- Locally compute $m \cdot [k^{-1}]_t + r \cdot [xk^{-1}]_t = [k^{-1} \cdot (m + rx)]_t$

Provisioning

Provisioning is somewhat slow. . .

Provisioning

Provisioning is somewhat slow. . .

Observation: provisioning can be done *once* for a given network of parties (rather than on a per-key basis)

Provisioning

Provisioning is somewhat slow. . .

Observation: provisioning can be done *once* for a given network of parties (rather than on a per-key basis)

Of course, need to prove that this does not affect security

Using precomputation to optimize signing

The signing protocol involves many ZK proofs

One bottleneck: $\approx 20t$ computations of the form $s_j^x t_j^y \bmod N_j$, where $\|x\| \approx 500$, $\|y\| \approx 3500$, and $\|N_i\| \approx 3000$

Using precomputation to optimize signing

The signing protocol involves many ZK proofs

One bottleneck: $\approx 20t$ computations of the form $s_j^x t_j^y \bmod N_j$, where $\|x\| \approx 500$, $\|y\| \approx 3500$, and $\|N_i\| \approx 3000$

Observation: do precomputation during provisioning to speed up fixed-base multi-exponentiations

- E.g., for parameters above, $\approx 8\times$ speedup by storing $\approx 300KB$

(Key-dependent) presigning

CGGMP presigning computes g^k, $[k^{-1}]_t$, and $[xk^{-1}]_t$

- Given this information and m, can sign in one round

(Key-dependent) presigning

CGGMP presigning computes g^k, $[k^{-1}]_t$, and $[xk^{-1}]_t$

- Given this information and m, can sign in one round

Note that presigning is *key-dependent*

- Key-dependent presigning is not great in practice

(Key-dependent) presigning

CGGMP presigning computes g^k, $[k^{-1}]_t$, and $[xk^{-1}]_t$

- Given this information and m, can sign in one round

Note that presigning is *key-dependent*

- Key-dependent presigning is not great in practice

Question: is (efficient) key-*independent* presigning (with one-round online signing) possible in the no-honest-majority setting?

Honest-majority ECDSA

Honest-majority ECDSA

We see value in honest-majority ECDSA protocols

- Can be more efficient, while offering "equivalent" security for some applications
- Can offer better availability
- Can offer security properties (e.g., robustness) not achievable otherwise

We would be interested in collaborating on a submission to NIST

Honest-majority ECDSA

Note

In the honest-majority setting, the number of parties running the protocol is (at least) $2t+1$

Honest-majority ECDSA

Note

In the honest-majority setting, the number of parties running the protocol is (at least) $2t + 1$

Damgård et al. (2020) show an efficient honest-majority ECDSA protocol

- Appears covered by US Patent 11,757,657 assigned to Sepior APS

Honest-majority ECDSA

Note

In the honest-majority setting, the number of parties running the protocol is (at least) $2t + 1$

Damgård et al. (2020) show an efficient honest-majority ECDSA protocol

- Appears covered by US Patent 11,757,657 assigned to Sepior APS

Will sketch an alternate approach

- One possibility...

Honest-majority ECDSA (high level)

Provisioning and key generation

- Provision parties with setup for PRSS (cf. DKG talk)
- Honest-majority DKG to generate $[x]_t$

Presigning

- Generate random $[k^{-1}]_t, [a]_t$
- Compute $[ak^{-1}]_t$ using a multiplication protocol
- Reconstruct ak^{-1}; compute $[k]_t$, g^k, and $r := F(g^k)$

Signing

- Compute $m \cdot [k^{-1}]_t + r \cdot [k^{-1}]_t \cdot [x]_t = [k^{-1} \cdot (m + rx)]_{2t}$

Honest-majority ECDSA (high level)

Provisioning and key generation

- Provision parties with setup for PRSS (cf. DKG talk)
- Honest-majority DKG to generate $[x]_t$

Key-independent presigning

- Generate random $[k^{-1}]_t, [a]_t$
- Compute $[ak^{-1}]_t$ using a multiplication protocol
- Reconstruct ak^{-1}; compute $[k]_t$, g^k, and $r := F(g^k)$

Signing

- Compute $m \cdot [k^{-1}]_t + r \cdot [k^{-1}]_t \cdot [x]_t = [k^{-1} \cdot (m + rx)]_{2t}$

Batch presigning

Presigning needs a multiplication protocol resilient to malicious behavior

Batch presigning

Presigning needs a multiplication protocol resilient to malicious behavior

Can amortize cost of multiplication by doing *batch* presigning

- This becomes practical when presigning is key-independent!

Batch multiplication [Nordholt-Veeningen (2018)]

Given $\{[a_i]_t\}_{i=1}^{m+1}$, $\{[b_i]_t\}_{i=1}^{m+1}$

Batch multiplication [Nordholt-Veeningen (2018)]

Given $\{[a_i]_t\}_{i=1}^{m+1}$, $\{[b_i]_t\}_{i=1}^{m+1}$

Let F, G be degree-m polynomials with $F(i) = a_i$, $G(i) = b_i$ for $i \in [m]$; locally compute $\{[a_j = F(j)]_t\}_{j=m+2}^{2m+1}$ and $\{[b_j = G(j)]_t\}_{j=m+2}^{2m+1}$

Batch multiplication [Nordholt-Veeningen (2018)]

Given $\{[a_i]_t\}_{i=1}^{m+1}$, $\{[b_i]_t\}_{i=1}^{m+1}$

Let F, G be degree-m polynomials with $F(i) = a_i$, $G(i) = b_i$ for $i \in [m]$; locally compute $\{[a_j = F(j)]_t\}_{j=m+2}^{2m+1}$ and $\{[b_j = G(j)]_t\}_{j=m+2}^{2m+1}$

For $i \in [2m+1]$, use "passively secure" multiplication to get $\{[c_i]_t\}_{i=1}^{2m+1}$

Batch multiplication [Nordholt-Veeningen (2018)]

Given $\{[a_i]_t\}_{i=1}^{m+1}$, $\{[b_i]_t\}_{i=1}^{m+1}$

Let F, G be degree-m polynomials with $F(i) = a_i$, $G(i) = b_i$ for $i \in [m]$; locally compute $\{[a_j = F(j)]_t\}_{j=m+2}^{2m+1}$ and $\{[b_j = G(j)]_t\}_{j=m+2}^{2m+1}$

For $i \in [2m+1]$, use "passively secure" multiplication to get $\{[c_i]_t\}_{i=1}^{2m+1}$

Let H be degree-$2m$ polynomial with $H(i) = c_i$ for $i \in [2m+1]$

- If everyone was honest, then $H(X) = F(X) \cdot G(X)$

Batch multiplication [Nordholt-Veeningen (2018)]

Given $\{[a_i]_t\}_{i=1}^{m+1}$, $\{[b_i]_t\}_{i=1}^{m+1}$

Let F, G be degree-m polynomials with $F(i) = a_i$, $G(i) = b_i$ for $i \in [m]$; locally compute $\{[a_j = F(j)]_t\}_{j=m+2}^{2m+1}$ and $\{[b_j = G(j)]_t\}_{j=m+2}^{2m+1}$

For $i \in [2m+1]$, use "passively secure" multiplication to get $\{[c_i]_t\}_{i=1}^{2m+1}$

Let H be degree-$2m$ polynomial with $H(i) = c_i$ for $i \in [2m+1]$

- If everyone was honest, then $H(X) = F(X) \cdot G(X)$

Choose $\alpha \leftarrow \mathbb{Z}_q$; reconstruct $F(\alpha)$, $G(\alpha)$, $H(\alpha)$ and check correctness

Batch presigning

Note

Measuring performance for threshold signing of a single message is not indicative of the amortized performance when batch presigning is used

Summary

Highlighted some (technical) considerations for threshold cryptography in "key-management networks"

- Should be taken into account in submissions/evaluation

Interest in standardizing CGGMP no-honest-majority protocol + honest-majority ECDSA protocol

Thank you!