Threshold ECDSA
from ECDSA assumptions:
the multiparty case

Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat

ej@ckdoerner.net ykondi@ccs.neu.edu eysa@ccs.neu.edu abhi@neu.edu

Northeastern University
Traditional Signature

pk

sk
Threshold Signature

\{ sk_A, sk_B, sk_C \} \leftarrow \text{Share}(sk)

INDISTINGUISHABLE FROM ORDINARY SIGNATURE

pk
3-of-n Signature Scheme

\[ \text{pk} \]

\[ \text{sk}_A, \text{sk}_B, \text{sk}_C, \text{sk}_D, \text{sk}_E, \text{sk}_F \]
3-of-n Signature Scheme

Diagram showing five individuals with private keys $sk_A$, $sk_B$, $sk_C$, $sk_D$, and $sk_E$ sharing a public key $pk$. The diagram includes a checkmark indicating that the signature is valid when at least three out of five private keys are used.
3-of-n Signature Scheme
3-of-n Signature Scheme

\(sk_A, sk_B, sk_C, sk_D, sk_E\)

\(pk\)

\(\checkmark\)
3-of-n Signature Scheme

\[ \text{sk}_A, \text{sk}_B, \text{sk}_C, \text{sk}_D, \text{sk}_E \]

\[ \text{pk} \]

\[ \text{x} \]
3-of-n Signature Scheme

\[ \text{sk}_A, \text{pk}, \text{sk}_B, \text{sk}_C, \text{sk}_D, \text{sk}_E \]
Full Threshold

- Scheme can be instantiated with any $t \leq n$

- Adversary corrupts up to $t-1$ parties
ECDSA

- **Elliptic Curve Digital Signature Algorithm**
- Devised by David Kravitz, standardized by NIST
- Widespread adoption across the internet
Notation

Elliptic curve parameters: $G$, $q$

Secret values: $sk$, $k$

Public values: $pk$, $R$
ECDSA Recap

\[ R = k \cdot G \]

\[ \text{sign}(m, \text{sk}, k) = H(m) + \text{sk} \cdot r_x \]

x-coordinate of \( R \)

Non-linearity makes ‘thresholdization’ difficult
Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]

- Practical key generation and efficient signing (full threshold):
  - [Gennaro Goldfeder 18]: Paillier-based
  - [Lindell Nof Ranellucci 18]: El-Gamal based

- **Our work last year [DKLs18]:** 2-of-n ECDSA under native assumptions

- **This work:** Full-Threshold ECDSA under native assumptions
Our Approach

- 2-party multipliers: Oblivious Transfer in ECDSA curve

  - Pros:
    - With OT Extension (no extra assumptions) just a few milliseconds
    - Native assumptions (CDH in the same curve)

  - Con: Higher bandwidth (100s of KB/party)
Our Approach

• OT-MUL secure up to choice of inputs

• Light consistency check *(unique to our protocol)*:
  - Verify shares in the exponent before reveal
  - Costs 5 exponentiations + curve points/party
  - Subverting checks implies solving CDH in the same curve
Tradeoffs

• Our work avoids expensive zero-knowledge proofs and assumptions foreign to ECDSA itself, required by other works in the area.

• Using OT-MUL is very light on computation, but more demanding of bandwidth than alternative approaches; we argue this is not an issue for most applications.

• Our wall clock times (even WAN) are an order of magnitude better than the next best concurrent work.
Our Model

• **Universal Composability** [Canetti ’01] (static adv., local RO)

• **Functionality (trusted third party emulated by protocol):**
  - Store secret key
  - Compute ECDSA signature when enough parties ask

• **Assumption:** CDH is hard in the ECDSA curve

• **Network:** Synchronous, broadcast

• **Security with abort**
Our Approach

• **Setup**: MUL setup, VSS for [sk]

• **Signing**:

1. Get candidate shares [k], [1/k], and \( R=k \cdot G \)
2. Compute \([sk/k] = \text{MUL}([1/k], [sk])\)
3. Check relations in exponent
4. Reconstruct \( sig = [1/k] \cdot H(m) + [sk/k] \)
Setup

• Fully distributed

• **MUL setup:** Pairwise among parties (128 OTs)

• **Key generation:** (Pedersen-style)
  - Every party Shamir-shares a random secret
  - Secret key is sum of parties’ contributions
  - Verify in the exponent that parties’ shares are on the same polynomial
Our Approach

• **Setup**: MUL setup, VSS for \([sk]\)

• **Signing**: 
  1. Get candidate shares \([k]\), \([1/k]\), and \(R=k\cdot G\)
  2. Compute \([sk/k] = \text{MUL}([1/k], [sk])\)
  3. Check relations in exponent
  4. Reconstruct \(\text{sig} = [1/k] \cdot H(m) + [sk/k]\)
Obtaining Candidate Shares

• **Building Block**: Two party MUL with full security [DKLs18]

• **One approach** (implemented):
  
  - Each party starts with multiplicative shares of $k$ and $1/k$
  
  - Multiplicative to additive shares: $\log(t)+c$ rounds

• **Alternative**: [Bar-Ilan&Beaver ’89] approach yields constant round protocol (work in progress)
Our Approach

• **Setup:** MUL setup, VSS for [sk]

• **Signing:**

1. Get candidate shares \([k], [1/k], \) and \(R=k\cdot G\)

2. Compute \([sk/k] = \text{MUL}([1/k], [sk])\)

3. Check relations in exponent

4. Reconstruct \(\text{sig} = [1/k] \cdot H(m) + [sk/k]\)
Our Approach

**Setup:** MUL setup, VSS for [sk]

**Signing:**

1. Get candidate shares \([k], [1/k],\) and \(R=k \cdot G\)
2. Compute \([sk/k] = \text{MUL}([1/k], [sk]) \Rightarrow \text{Standard GMW}\)
3. Check relations in exponent
4. Reconstruct \(\text{sig} = [1/k] \cdot H(m) + [sk/k]\)
Our Approach

• **Setup**: MUL setup, VSS for [sk]

• **Signing**:

1. Get candidate shares \([k], \left[1/k\right], \text{ and } R=k\cdot G\)
2. Compute \([sk/k] = \text{MUL([1/k, [sk]])}\)
3. Check relations in exponent
4. Reconstruct \(\text{sig} = [1/k]\cdot H(m) + [sk/k]\)
Major challenges from 2 to Multi-party

2-party check does not obviously generalize [LNR18]

Can’t use Diffie-Hellman Exchange for R
Check in Exponent

• There are **three** relations that have to be verified

\[
\begin{bmatrix} k \\ \frac{1}{k} \end{bmatrix} \quad \begin{bmatrix} k \end{bmatrix} \quad \begin{bmatrix} sk \\ \frac{1}{k} \end{bmatrix}
\]
Check in Exponent

\[
\begin{bmatrix}
    k \\
\end{bmatrix}
\begin{bmatrix}
    1 \\
    \frac{1}{k} \\
    \frac{sk}{k}
\end{bmatrix}
\]

- **Technique**: Each equation is verified in the exponent, using ‘auxiliary’ information that’s already available.

- **Cost**: 5 exponentiations, 5 group elements per party, independent of party count, and no ZK proofs.
Check in Exponent

- **Task:** verify relationship between \([k]\) and \([1/k]\)

- **Idea:** verify \(\left[\frac{1}{k}\right][k] = 1\) by verifying \(\left[\frac{1}{k}\right][k] \cdot G = G\)
Check in Exponent

Attempt at a solution:

Public

\[ R \]

Broadcast

\[ \Gamma_i = \left[ \frac{1}{k} \right]_i \cdot R \]

Verify

\[ \sum_{i \in [n]} \Gamma_i = G \]
Check in Exponent

Attempt at a solution:

Public

\[ R = k_A k_h \cdot G \]

Broadcast

\[ \Gamma_i = \left[ \begin{array}{cc} 1 & 1 \\ k_A & k_h \end{array} \right] \cdot R \]

Verify

\[ \sum_{i \in [n]} \Gamma_i = G \]
Check in Exponent

Attempt at a solution:

Public

\[ R = k_A k_h \cdot G \]

Broadcast

\[ \Gamma_i = \left[ \left( \frac{1}{k_A} + \epsilon \right) \frac{1}{k_h} \right] \cdot R \]

Verify

\[ \sum_{i \in [n]} \Gamma_i = G + \epsilon k_A \cdot G \]

Adversary's contribution

Honest Party's contribution

Easy for Adv. to offset
Idea: Randomize Target

- Currently we expect $\sum \Gamma_i$ to hit a fixed target $G$

- **Idea**: randomize the multiplication so target is unpredictable

- Compute $\left[ \frac{\phi}{k} \right]$ instead of $\left[ \frac{1}{k} \right]$

- Reveal $\phi$ only after every other value is committed
Check in Exponent

Attempt at a solution:

Public

\[ R = k_A k_h \cdot G \]

Broadcast

\[ \Gamma_i = \left[ \begin{array}{cc} 1 & 1 \\ k_A & k_h \end{array} \right] \cdot R \]
Check in Exponent

Attempt at a solution:

Public

$$R = k_A k_h \cdot G$$

Broadcast

$$\Gamma_i = \begin{bmatrix} \phi_A & \phi_h \\ k_A & k_h \end{bmatrix}_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma_i = \phi_A \phi_h \cdot G$$
Check in Exponent

Attempt at a solution:

\[
R = k_A k_h \cdot G
\]

\[
\Gamma_i = \left[ \frac{\phi_A}{k_A}, \frac{\phi_h}{k_h} \right] \cdot R
\]

Verify

\[
\sum_{i \in [n]} \Gamma_i = \Phi
\]
Attempt at a solution:

Public

\[ R = k_A k_h \cdot G \]

Broadcast

\[ \Gamma_i = \left( \frac{\phi_A}{k_A} + \epsilon \right) \frac{\phi_h}{k_h} \cdot R \]

Verify

\[ \sum_{i \in [n]} \Gamma_i = \Phi + \epsilon \phi_h k_A \cdot G \]

Completely unpredictable
There are three relations that have to be verified:

\[
\begin{bmatrix} k \end{bmatrix} \quad \begin{bmatrix} 1 \end{bmatrix} \quad \begin{bmatrix} sk \end{bmatrix}
\]

Each costs, per party:
- 2 exponentiations
- 2 field elements

Two broadcast rounds
Our Approach

- **Setup**: MUL setup, VSS for $[sk]$  

- **Signing**:
  1. Get candidate shares $[k]$, $[1/k]$, and $R = k \cdot G$
  2. Compute $[sk/k] = \text{MUL}([1/k], [sk])$
  3. Check relations in exponent
  4. Reconstruct $\text{sig} = [1/k] \cdot H(m) + [sk/k]$  

Broadcast linear combination of shares
# Dominant Costs

<table>
<thead>
<tr>
<th></th>
<th>Rounds</th>
<th>Public Key</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Setup</strong></td>
<td>5</td>
<td>520n</td>
<td>21$n$ KB</td>
</tr>
<tr>
<td><strong>Signing</strong></td>
<td>$\log(t)+6$</td>
<td>5</td>
<td>$&lt;100t$ KB</td>
</tr>
</tbody>
</table>

Journal version (in progress): **8 round signing**

(à la [Bar-Ilan Beaver 89])
Benchmarks

- Implementation in **Rust**
- Ran benchmarks on Google Cloud
- One node per party
- **LAN** and **WAN** tests (up to 16 zones)
- **Low Power Friendliness**: Raspberry Pi (~93ms for 3-of-3)
LAN Setup

Broadcast PoK (DLog), **Pairwise**: 128 OTs
LAN Setup

Broadcast PoK (DLog), **Pairwise**: 128 OTs
LAN Setup

Broadcast PoK (DLog), **Pairwise**: 128 OTs
LAN Signing

![Graph showing execution time vs number of parties (t)]
LAN Signing
Among these, the longest leg was between Oregon and South Carolina, with a round-trip latency of 66.5 ms and bandwidth of 53.4 MBits/sec. We tested two configurations: one with only the five US datacenters participating, and another with all 16. For the longest leg, the average of 5 trials was 13.6 ms and 67.9 ms, respectively. Benchmarks involving only a single zone are LAN benchmarks, for comparison.

Table IV:

<table>
<thead>
<tr>
<th>Parties/Zones</th>
<th>Signing Rounds</th>
<th>Signing Time</th>
<th>Setup Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/1</td>
<td>9</td>
<td>13.6</td>
<td>67.9</td>
</tr>
<tr>
<td>5/5</td>
<td>9</td>
<td>288</td>
<td>328</td>
</tr>
<tr>
<td>16/1</td>
<td>10</td>
<td>26.3</td>
<td>181</td>
</tr>
<tr>
<td>16/16</td>
<td>10</td>
<td>3045</td>
<td>1676</td>
</tr>
<tr>
<td>40/1</td>
<td>12</td>
<td>60.8</td>
<td>539</td>
</tr>
<tr>
<td>40/5</td>
<td>12</td>
<td>592</td>
<td>743</td>
</tr>
<tr>
<td>128/1</td>
<td>13</td>
<td>193.2</td>
<td>2300</td>
</tr>
<tr>
<td>128/16</td>
<td>13</td>
<td>4118</td>
<td>3424</td>
</tr>
</tbody>
</table>

All time values in milliseconds

WAN Benchmarks

It is worth noting that Wang et al. [33] recently made the claim that they performed the largest-scale demonstration of multiparty computation to date. Their benchmark involves 128 parties split among 16 datacenters, using a multiparty garbling protocol that they developed. Our WAN benchmarks employ protocols of Doerner et al. [1] without modification, and when the setup and signing are considered to have a much lower circuit complexity than ECDSA; this is reflected in the significantly lower computation costs are combined for our protocol, it requires 2.5 seconds; whereas their protocol requires 2.5 minutes. When the setup and signing among the group of Pis, with current-generation CPUs; these are located on embedded devices such as hardware tokens or smartwatches, we used the protocols presented in this paper. For setup, fixing $n=2$ and $t=3$, we used the slightly more efficient algorithms of the authors of this paper. We also note that the in the clear setting, AES is generally considered to have a much lower circuit complexity than ECDSA; this is reflected in the significantly lower computation time for a single AES operation as compared to signing a signature with ECDSA. It is counterintuitive, but unsurprising in light of the fact that SHA-256 (except where required by ECDSA) in favor of the algebraically structured nature of ECDSA allows custom preprocessing, and in the global WAN setting with 128 parties, employing, they report a 17-second wall clock time, including as opposed to the blind use of generic MPC for all tasks.

For example, in the clear setting, AES is generally considered to have a much lower circuit complexity than ECDSA; this is reflected in the significantly lower computation time for a single AES operation as compared to signing a signature with ECDSA. It is counterintuitive, but unsurprising in light of the fact that SHA-256 (except where required by ECDSA) in favor of the algebraically structured nature of ECDSA allows custom preprocessing, and in the global WAN setting with 128 parties, employing, they report a 17-second wall clock time, including as opposed to the blind use of generic MPC for all tasks.
Among these, the longest leg was between Oregon and South Taiwan, Tokyo, Mumbai, and Singapore. Among the complete benchmark involves 128 parties split among 16 datacenters, multiparty computation to date. Their benchmark involves a round-trip latency of 348 ms and a bandwidth of 53.4 Mbits/sec. The remaining 11 were located in Montreál, Carolina, with a round-trip latency of 66.5 ms and bandwidth of 353 Mbits/sec. The subgroup of five zones inside the US are highlighted in red.

FIG. 4: Map of Datacenter Locations used for WAN Benchmarks

TABLE IV: Wall-clock Times in Milliseconds over WAN

<table>
<thead>
<tr>
<th>Parties/Zones</th>
<th>Signing Rounds</th>
<th>Signing Time (ms)</th>
<th>Setup Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/1</td>
<td>9</td>
<td>13.6</td>
<td>67.9</td>
</tr>
<tr>
<td>5/5</td>
<td>9</td>
<td>288</td>
<td>328</td>
</tr>
<tr>
<td>16/1</td>
<td>10</td>
<td>26.3</td>
<td>181</td>
</tr>
<tr>
<td><strong>16/16</strong></td>
<td><strong>10</strong></td>
<td><strong>3045</strong></td>
<td><strong>1676</strong></td>
</tr>
<tr>
<td>40/1</td>
<td>12</td>
<td>60.8</td>
<td>539</td>
</tr>
<tr>
<td>40/5</td>
<td>12</td>
<td>592</td>
<td>743</td>
</tr>
<tr>
<td>128/1</td>
<td>13</td>
<td>193.2</td>
<td>2300</td>
</tr>
<tr>
<td>128/16</td>
<td>13</td>
<td>4118</td>
<td>3424</td>
</tr>
</tbody>
</table>

Finally, we performed a set of benchmarks on a group of five zones inside the US, wherein an embedded device signs with a more powerful one, as opposed to the blind use of generic MPC for all tasks. For setup, we collected 50 samples, and for signing, we collected 250. As an optimization for the embedded setting, we abandoned protocols of Doerner et al. \cite{1} without modification, and when we used a 2013 15" Macbook Pro running Mac OS 10.13 (i.e. one author’s laptop). This machine was engaged in other tasks at the time of benchmarking, and no attempt was made to employ), they report a 17-second wall clock time, including preprocessing, and in the global WAN setting with 128 parties, their protocol requires 2.5 minutes. When the setup and signing costs are combined for our protocol, it requires 2.5 seconds as both ECDSA; this is reflected in the significantly lower computation time for a single AES operation as compared to signing a three Raspberry Pi model 3B+ single-board computers in order to demonstrate the feasibility of evaluating our protocol (and the three Raspberry Pi model 3B+ single-board computers in order to demonstrate the feasibility of evaluating our protocol (and the algebraically structured nature of ECDSA allows custom protocols to be computationally efficient enough to run even on smaller, low-powered devices such as hardware tokens or smartwatches, as party count and geographic distribution are concerned. As party count and geographic distribution are concerned. We also note that the in the clear setting, AES is generally considered to have a much lower circuit complexity than BLAKE2 hash function \cite{34}, using assembly implementations.

We believe that this consideration allows our protocol to outperform Wang et al.'s realization of ECDSA; this is reflected in the significantly lower computation time for a single AES operation as compared to signing a three Raspberry Pi model 3B+ single-board computers in order to demonstrate the feasibility of evaluating our protocol (and the algebraically structured nature of ECDSA allows custom protocols to be computationally efficient enough to run even on smaller, low-powered devices such as hardware tokens or smartwatches, as party count and geographic distribution are concerned.

BLAKE2 hash function \cite{34}, using assembly implementations.

We believe that this consideration allows our protocol to outperform Wang et al.'s realization of ECDSA; this is reflected in the significantly lower computation time for a single AES operation as compared to signing a three Raspberry Pi model 3B+ single-board computers in order to demonstrate the feasibility of evaluating our protocol (and the algebraically structured nature of ECDSA allows custom protocols to be computationally efficient enough to run even on smaller, low-powered devices such as hardware tokens or smartwatches, as party count and geographic distribution are concerned.
### WAN Benchmarks

All time values in milliseconds

<table>
<thead>
<tr>
<th>Parties/Zones</th>
<th>Signing Rounds</th>
<th>Signing Time</th>
<th>Setup Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/1</td>
<td>9</td>
<td>13.6</td>
<td>67.9</td>
</tr>
<tr>
<td>5/5</td>
<td>9</td>
<td>288</td>
<td>328</td>
</tr>
<tr>
<td>16/1</td>
<td>10</td>
<td>26.3</td>
<td>181</td>
</tr>
<tr>
<td>16/16</td>
<td>10</td>
<td>3045</td>
<td>1676</td>
</tr>
<tr>
<td>40/1</td>
<td>12</td>
<td>60.8</td>
<td>539</td>
</tr>
<tr>
<td>40/5</td>
<td>12</td>
<td>592</td>
<td>743</td>
</tr>
<tr>
<td>128/1</td>
<td>13</td>
<td>193.2</td>
<td>2300</td>
</tr>
<tr>
<td><strong>128/16</strong></td>
<td><strong>13</strong></td>
<td><strong>4118</strong></td>
<td><strong>3424</strong></td>
</tr>
</tbody>
</table>
## Comparison

All time figures in milliseconds

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Signing ( t = 2 )</th>
<th>Signing ( t = 20 )</th>
<th>Setup ( n = 2 )</th>
<th>Setup ( n = 20 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Work</td>
<td>9.5</td>
<td>31.6</td>
<td>45.6</td>
<td>232</td>
</tr>
<tr>
<td>GG18</td>
<td>77</td>
<td>509</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LNR18</td>
<td>304</td>
<td>5194</td>
<td>~11000</td>
<td>~28000</td>
</tr>
</tbody>
</table>

Note: Our figures are wall-clock times; includes network costs.
Comparison

All time figures in milliseconds

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Signing $t = 2$</th>
<th>Signing $t = 20$</th>
<th>Setup $n = 2$</th>
<th>Setup $n = 20$</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Work</td>
<td>9.5</td>
<td>31.6</td>
<td>45.6</td>
<td>232</td>
</tr>
<tr>
<td>GG18</td>
<td>77</td>
<td>509</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LNR18</td>
<td>304</td>
<td>5194</td>
<td>~11000</td>
<td>~28000</td>
</tr>
</tbody>
</table>

Note: Our figures are wall-clock times; includes network costs.
Is communication the bottleneck?

- Mobile applications (human-initiated):
  - eg. t=4, <4Mb transmitted per party
  - Well within LTE envelope for responsivity
Is communication the bottleneck?

• Large-scale automated distributed signing:
  - Threshold 2: 3.8ms/sig  <= ~263 sig/second
  - Threshold 20: 31.6ms/sig  <= ~31 sig/second

• Both settings need <500Mb bandwidth
Conclusion

• Efficient full-threshold ECDSA with fully distributed keygen

• **Paradigm**: ‘produce candidate shares, verify by exponent check’
  costs **5 exponentiations** (+ many hashes) to sign, **no ZK online**

• **Instantiation**: Cryptographic assumptions native to ECDSA itself
  (**CDH** in the same curve)

• Lightweight computation **but communication well within practical range** (<100t KB/party)

• **Wall-clock times**: Practical in realistic scenarios
Thank you!
eprint.iacr.org/2019/523