Secure Two-party Threshold ECDSA from ECDSA Assumptions

Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat
Northeastern University
Elliptic Curve Digital Signature Algorithm

- Digital Signature Algorithm with elliptic curves
  - Smaller signature (512 bits) and key sizes (256-bit)
  - Security proof in “generic group model”

- Used pervasively in:
  - TLS
  - DNSSEC
  - Cryptocurrencies (Bitcoin, Ethereum, …)
Why Threshold Signatures?
Single Point of Failure for Signer
Distribute Signing Key Among Many Devices
Multi-Signature

$n$ parties

Each party has their own key pair

To sign a message, each party produces a signature under their public key

Signature: $\sigma_1, \sigma_2, \ldots, \sigma_n$
Why not Multi-Signatures?

• High bandwidth
  • Need to produce $n$ signatures
  • Major bugs in implementations trying to reduce bandwidth
• Participating signers publicly known
$t$-of-$n$ Threshold Signature Scheme

$n$ parties

Jointly compute a *single* public key

Each party has a share of the secret key

$t$ parties needed to generate new signatures

Signature: $\sigma$
2-of-$n$ Threshold Signature Scheme
2-of-$n$ Threshold Signature Scheme

Participation of 2 parties needed to generate new signatures
2-of-\( n \) Threshold Signature Scheme

Single users cannot forge a signature
2-of-$n$ Threshold Signature Scheme

Single users cannot forge a signature
2-of-\(n\) Threshold Signature Scheme

Single users cannot forge a signature
Handling Corruptions
Handling Corruptions

Adversary can interact with parties
Handling Corruptions

Adversary can interact with parties
Handling Corruptions

Adversary can interact with parties
Handling Corruptions

Adversary still shouldn’t be able to forge a signature
Security Model

Real

Ideal

Any Adv in the real world can be mapped to one in the ideal world
ECDSA Functionality

Note: Our functionality concretely implements the ECDSA algorithm and is not a signature algorithm.
ECDSA Functionality
ECDSA Functionality
ECDSA Functionality
ECDSA Functionality
ECDSA Functionality
(Preview) Prior Works on Threshold ECDSA

• Some not proven via real/ideal
• Some have long complex, setup (several minutes), semi-honest
• All need additional assumptions
This Work

• Maliciously secure threshold ECDSA
  • 2-round with relaxed definition
  • Maliciously secure multiplication with external checks
• No additional assumptions
  • Threshold ECDSA scheme from only ECDSA
• Improved efficiency
  • ~3 ms to sign
• Open source implementation in Rust
This Talk

• 2-of-2 Threshold ECDSA
  • Extended to 2-of-n in paper

• Optimizations
Threshold Schnorr

SchnorrSign(sk, m):
    Sample instance key $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \parallel m)$
    $\sigma = k - sk \cdot e$
Output $(\sigma, e)$
Threshold Schnorr

SchnorrSign($sk, m$):

→ Sample instance key $k \leftarrow \mathbb{Z}_q$
→ $R = k \cdot G$
→ $e = H(R \parallel m)$
→ $\sigma = k - sk \cdot e$

Output $(\sigma, e)$

$sk = sk_a + sk_b$

$k = k_a + k_b$

$R = k_a \cdot G + k_b \cdot G$

$\sigma = k - sk \cdot e$

$k_a + k_b - (sk_a + sk_b) \cdot e$
Threshold Schnorr

SchnorrSign(sk, m):
Sample instance key $k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(R \parallel m)$

$\rightarrow \sigma = k - sk \cdot e$

Output $(\sigma, e)$
Threshold Schnorr

SchnorrSign$(sk, m)$:
Sample instance key $k \leftarrow \mathbb{Z}_q$
$R = k \cdot G$
$e = H(R \parallel m)$
$\rightarrow \sigma = k - sk \cdot e$
Output $(\sigma, e)$
Threshold Schnorr

SchnorrSign(sk, m):
Sample instance key $k \leftarrow \mathbb{Z}_q$
$R = k \cdot G$
$e = H(R \parallel m)$
$\sigma = k - sk \cdot e$
→ Output $(\sigma, e)$
What makes ECDSA difficult?

SchnorrSign\((sk, m)\):
- Sample instance key \( k \leftarrow \mathbb{Z}_q \)
- \( R = k \cdot G \)
- \( e = H(R \ || \ m) \)
- \( \sigma = k - sk \cdot e \)
- Output \((\sigma, e)\)

ECDSASign\((sk, m)\):
- Sample instance key \( k \leftarrow \mathbb{Z}_q^* \)
- \( R = k \cdot G \)
- \( e = H(m) \)
- \( \sigma = e \cdot \frac{sk}{k} \cdot r_x \)
- Output \((\sigma, r_x)\)
What makes ECDSA difficult?

SchnorrSign$(sk, m)$:
Sample instance key $k \leftarrow \mathbb{Z}_q$
$R = k \cdot G$
$e = H(R \parallel m)$
$\sigma = k - sk \cdot e$
Output $(\sigma, e)$

ECDSASign$(sk, m)$:
Sample instance key $k \leftarrow \mathbb{Z}_q^*$
$R = k \cdot G$
$e = H(m)$
$\sigma = \frac{e}{k} + \frac{sk}{k} \cdot r_x$
Output $(\sigma, r_x)$
What makes ECDSA difficult?

SchnorrSign\((sk, m)\):
- Sample instance key \(k \leftarrow \mathbb{Z}_q\)
- \(R = k \cdot G\)
- \(e = H(R \parallel m)\)
- \(\sigma = k - sk \cdot e\)
- Output \((\sigma, e)\)

ECDSASign\((sk, m)\):
- Sample instance key \(k \leftarrow \mathbb{Z}_q^*\)
- \(R = k \cdot G\)
- \(e = H(m)\)
- \(\sigma = \frac{e}{k} + \frac{sk}{k} \cdot r_x\)
- Output \((\sigma, r_x)\)

Need shares of \(k\) and \(k^{-1}\)
Prior Approaches

Gennaro-Goldfeder-Narayanan16
Lindell17
Boneh-Gennaro-Goldfeder17

1. Multiplicative shares of the secret and instance keys
Prior Approaches

Gennaro-Goldfeder-Narayanan\textsuperscript{16}

Lindell\textsuperscript{17}

Boneh-Gennaro-Goldfeder\textsuperscript{17}

1. Multiplicative shares of the secret and instance keys

2. Use additively homomorphic Paillier encryption

\[ k = k_a \cdot k_b \]
\[ \text{sk} = \text{sk}_a \cdot \text{sk}_b \]
\[ \frac{1}{k} \cdot (H(m) + \text{sk} \cdot r_x) \]
\[ \frac{1}{k_a} \cdot \left( \frac{1}{k_b} \cdot H(m) + \frac{\text{sk}}{k_b} \cdot r_x \right) \]

Paillier encryption
Prior Approaches
GGN16, BGG17
• t-of-n, 4 rounds (reduced from 6 rounds)
• Expensive setup; not implemented or not reported
  • Additional assumptions:
    • Decisional Composite Residuosity
    • Strong RSA

Lindell17
• Only 2-of-2, 4 rounds
  • Additional assumptions:
    • Decisional Composite Residuosity
    • Paillier-EC (new, construction-specific)
Our Approach to Threshold Signing

\[ pk = sk \cdot G \]
\[ sk = sk_a \cdot sk_b \]
Our Approach to Threshold Signing

\[ R = k \cdot G \]
\[ k = k_a \cdot k_b \]
Our Approach to Threshold Signing

\[ \frac{1}{k} \cdot H(m) + \frac{sk}{k} \cdot r_x \]

\[ \frac{1}{k_a} \cdot \frac{1}{k_b} \cdot H(m) + \frac{sk_a}{k_a} \cdot \frac{sk_b}{k_b} \cdot r_x \]
Our Approach to Threshold Signing

\[
\frac{1}{k} \cdot H(m) + \frac{sk}{k} \cdot r_x
\]

\[
\frac{1}{k_a} \cdot \frac{1}{k_b} \cdot H(m) + \frac{sk_a}{k_a} \cdot \frac{sk_b}{k_b} \cdot r_x
\]
Our Approach to Threshold Signing

\[
\frac{1}{k} \cdot H(m) + \frac{sk}{k} \cdot r_x
\]

\[
\frac{1}{k_a} \cdot \frac{1}{k_b} \cdot H(m) + \frac{sk_a}{k_a} \cdot \frac{sk_b}{k_b} \cdot r_x
\]
Our Approach to Threshold Signing

\[
\frac{1}{k_a} \cdot \frac{1}{k_b} \cdot H(m) + \frac{sk_a}{k_a} \cdot \frac{sk_b}{k_b} \cdot r_x
\]

\[t_a + t_b = \alpha \cdot \beta\]
Our Approach to Threshold Signing

\[
\frac{1}{k_a} \cdot \frac{1}{k_b} \cdot H(m) + \frac{sk_a}{k_a} \cdot \frac{sk_b}{k_b} \cdot r_x
\]
Our Approach to Threshold Signing

\[
\frac{t_a^{(1)}}{k_a} + \frac{t_b^{(1)}}{k_b} \cdot H(m) + \frac{sk_a}{k_a} \cdot \frac{sk_b}{k_b} \cdot r_x
\]
Our Approach to Threshold Signing

\[
(t_a^{(1)} + t_b^{(1)}) \cdot H(m) + \frac{sk_a}{k_a} \cdot \frac{sk_b}{k_b} \cdot r_x
\]
Our Approach to Threshold Signing

\[
(t_{a}^{(1)} + t_{b}^{(1)}) \cdot H(m) + (t_{a}^{(2)} + t_{b}^{(2)}) \cdot r_{x}
\]
Our Approach to Threshold Signing

\[
\left( t_a^{(1)} + t_b^{(1)} \right) \cdot H(m) + \left( t_a^{(2)} + t_b^{(2)} \right) \cdot r_x
\]
(Semi-honest)

[Gilboa99] Multiplication by Oblivious Transfer

# of OTs proportional to security parameter
Efficient with OT extension (symmetric key operations)
Skeleton Protocol

\[ pk = sk_a \cdot sk_b \cdot G \]
\[ R = k_a \cdot k_b \cdot G \]

\[ \sigma_a = t_a^{(1)} \cdot H(m) + t_a^{(2)} \cdot r_x \]

\[ \sigma_a = t_b^{(1)} \cdot H(m) + t_b^{(2)} \cdot r_x \]
Hardening for Malicious Security

\[ pk = sk_a \cdot sk_b \cdot G \]
\[ R = k_a \cdot k_b \cdot G \]

\[ \sigma_a = t_a^{(1)} \cdot H(m) + t_a^{(2)} \cdot r_x \]
\[ \sigma_a = t_b^{(1)} \cdot H(m) + t_b^{(2)} \cdot r_x \]
Hardening for Malicious Security

1. Maliciously secure multiplication

\[ \sigma_a = t_a^{(1)} \cdot H(m) + t_a^{(2)} \cdot r_x \]

\[ \sigma_a = t_b^{(1)} \cdot H(m) + t_b^{(2)} \cdot r_x \]
Hardening for Malicious Security

1. Maliciously secure multiplication
2. Enforce input consistency

\[ pk = sk_a \cdot sk_b \cdot G \]
\[ R = k_a \cdot k_b \cdot G \]

\[ \sigma_a = t_a^{(1)} \cdot H(m) + t_a^{(2)} \cdot r_x \]

\[ t_b^{(1)} \cdot H(m) + t_b^{(2)} \cdot r_x \]
# Hardening for Malicious Security

## Malicious Multiplication

1. Checks per OT
   - $\frac{1}{2}$ probability getting caught per OT
2. High entropy encoding scheme
   - Bob encodes his input into multiplication

## Input Consistency

- Verify output is an ECDSA signature
- A new consistency check
Assumptions Needed

Malicious Multiplication

1. Checks per OT
2. High entropy encoding scheme
   statistical in ROM

Input Consistency

Verify output is an ECDSA signature
ECDSA is a signature scheme

A new consistency check
Computational Diffie-Hellman

CDH implied by generic group model, which is what ECDSA is proven in
Security Against Malicious Bob

\[ t_{a}^{(1)} + t_{b}^{(1)} = \frac{1}{k} \]
\[ t_{a}^{(2)} + t_{b}^{(2)} = \frac{sk}{k} \]

Goal:
Enforce Consistency
Security Against Malicious Bob

\[ t_a^{(1)} + t_b^{(1)} = \frac{1}{k} \]

\[ t_a^{(2)} + t_b^{(2)} = \frac{sk}{k} \]
Security Against Malicious Bob

\[ t_a^{(1)} + t_b^{(1)} = \frac{1}{k} \]

\[ (t_a^{(2)} + t_b^{(2)}) \cdot G = \left( \frac{sk}{k} \right) \cdot G \]
Security Against Malicious Bob

\[ t_a^{(1)} + t_b^{(1)} = \frac{1}{k} \]

\[
\left( t_a^{(2)} + t_b^{(2)} \right) \cdot G = \left( \frac{sk}{k} \right) \cdot G \\
= \frac{1}{k} \cdot pk
\]
Security Against Malicious Bob

\[ t_a^{(1)} + t_b^{(1)} = \frac{1}{k} \]

\[ (t_a^{(2)} + t_b^{(2)}) \cdot G = \frac{1}{k} \cdot pk \]
Security Against Malicious Bob

\[ t_a^{(1)} + t_b^{(1)} = \frac{1}{k} \]

\( (t_a^{(2)} + t_b^{(2)}) \cdot G = \frac{1}{k} \cdot pk \)
Security Against Malicious Bob

\[ (t_a^{(1)} + t_b^{(1)}) \cdot pk = \frac{1}{k} \cdot pk \]

\[ (t_a^{(2)} + t_b^{(2)}) \cdot G = \frac{1}{k} \cdot pk \]
Security Against Malicious Bob

\[(t_a^{(1)} + t_b^{(1)}) \cdot pk = \frac{1}{k} \cdot pk\]

\[(t_a^{(2)} + t_b^{(2)}) \cdot G = \frac{1}{k} \cdot pk\]

\[\downarrow\]

\[(t_a^{(1)} + t_b^{(1)}) \cdot pk = (t_a^{(2)} + t_b^{(2)}) \cdot G\]
Security Against Malicious Bob

\[
(t_a^{(1)} + t_b^{(1)}) \cdot pk = (t_a^{(2)} + t_b^{(2)}) \cdot G
\]

\[
t_a^{(1)} \cdot pk - t_a^{(2)} \cdot G = t_b^{(2)} \cdot G - t_b^{(1)} \cdot pk
\]
Security Against Malicious Bob

\[ t_a^{(1)} + t_b^{(1)} = \frac{1}{k_a} + \frac{\delta}{k_a} \]

\[ t_a^{(1)} \cdot pk - t_a^{(2)} \cdot G = t_b^{(2)} \cdot G - t_b^{(1)} \cdot pk + \frac{\delta}{k_a} \cdot pk \]

Computing this is as hard as CDH!
Security Against Malicious Bob

\[ \Gamma: \quad t_{a_1} \cdot pk - t_{a_2} \cdot G = t_{b_2} \cdot G - t_{b_1} \cdot pk \]
Consistency Check Optimization

\[ \Gamma : \ t_a^{(1)} \cdot pk - t_a^{(2)} \cdot G = t_b^{(2)} \cdot G - t_b^{(1)} \cdot pk \]

\[ Enc_{\Gamma}(\sigma_a) \]
Instance Key Exchange

\[ D_b = k_b \cdot G \]

\[ R' = k'_a \cdot D_b \]

Multiplication

Bob’s OT Messages

Consistency Check

Alice’s OT Messages

\[ \Gamma \text{ Check} \]

\[ \sigma_a \]

Final signature output

Protocol

\[ sk_a \]

\[ k_a \]

\[ sk_b \]

\[ k_b \]
Protocol

Instance Key Exchange

$D_b = k_b \cdot G$

Multiplication

$R' = k'_a \cdot D_b$

Consistency Check

$\Gamma$ Check

Final signature output

Alice’s OT Messages

$k_a$

Bob’s OT Messages

$\sigma_a$

$sk_b$

$sk_a$
Multiplication

Bob's OT Messages

\[ D_b = k_b \cdot G \]

Alice's OT Messages

\[ R' = k'_a \cdot D_b \]

Instance Key Exchange

\[ \Gamma \text{ Check} \]

Consistency Check

\[ \sigma_a \]

Final signature output

Note: This 2 round comes at the cost of a slight relaxation to definition where Alice is allowed negl bias in instance key.
On the Benefit of Two Messages
Why not generic MPC

• Highly efficient multiplication in 2 rounds
  • Don’t amortize over large number of gates
• Exploit verifiability
  • Take advantage of public values with respect to signature scheme to verify inputs
  • Don’t need expensive techniques to ensure input consistency
Implementation

• Open source implementation in Rust
  • SHA-256, same as ECDSA
  • 10,000 samples for setup, 100,000 samples for signing
  • Setup is 5 rounds and all n parties participate
## Signing Communication Costs

<table>
<thead>
<tr>
<th></th>
<th>$\kappa = 256$</th>
<th>$\kappa = 384$</th>
<th>$\kappa = 521$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gennaro et al.</td>
<td>$\sim 1808$ KiB</td>
<td>$\sim 4054$ KiB</td>
<td>$\sim 7454$ KiB</td>
</tr>
<tr>
<td>Boneh et al.</td>
<td>$\sim 1680$ KiB</td>
<td>$\sim 3768$ KiB</td>
<td>$\sim 6924$ KiB</td>
</tr>
<tr>
<td>This Work (2-of-$n$)</td>
<td>232.8 KiB</td>
<td>481.3 KiB</td>
<td>844.7 KiB</td>
</tr>
<tr>
<td>This Work (2-of-2)</td>
<td>169.8 KiB</td>
<td>350.7 KiB</td>
<td>615.3 KiB</td>
</tr>
<tr>
<td>Lindell</td>
<td>769 B</td>
<td>897 B</td>
<td>1043 B</td>
</tr>
</tbody>
</table>
2-of-$n$ Setup over LAN

![Graph showing execution time vs number of parties]

- Execution Time (ms)
- Number of Parties
Benchmarks over WAN: 2-of-2 and 2-of-n

Round-trip latency between Virginia and Paris: 78.2 ms
Benchmarks over WAN: 2-of-4 Setup

Round-trip latency between US data centers: 11.2 ms to 79.9 ms
Benchmarks over WAN: 2-of-10 Setup

Round-trip latency between Ireland and Mumbai: 282 ms
Times in ms over WAN

<table>
<thead>
<tr>
<th></th>
<th>Setup</th>
<th>Signing</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-of-2</td>
<td>354.36</td>
<td>81.34</td>
</tr>
<tr>
<td>2-of-4 (US)</td>
<td>376.86</td>
<td>81.83</td>
</tr>
<tr>
<td>2-of-10 (World)</td>
<td>1228.46</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• ECDSA threshold with no more assumptions than ECDSA
• Improved efficiency
• Open-source implementation in Rust
  • https://gitlab.com/neucrypt/mpecds
• Can be extended to k-out-of-n
Thank You!
Appendix: 2-of-n Signing

\[ \text{sk} = \text{sk}_a + \text{sk}_b \]

\[ \frac{1}{k_a} \cdot \frac{1}{k_b} \cdot H(m) + \frac{1}{k_a} \cdot \frac{1}{k_b} \cdot (\text{sk}_a + \text{sk}_b) \cdot r_x \]

\[ \frac{1}{k_a} \cdot \frac{1}{k_b} \cdot H(m) + \frac{\text{sk}_a}{k_a} \cdot \frac{1}{k_b} \cdot r_x + \frac{1}{k_a} \cdot \frac{\text{sk}_b}{k_b} \cdot r_x \]