Aleator: Random Beacon via Scalable Threshold Signatures

Robert Chen
Mentored by Alin Tomescu

PRIMES Computer Science Conference
10/13/18
Why Scalability?

- **Scalable** threshold signature scheme
  - Increased security
  - Scalable random beacon
What is a Random Beacon?

A set of servers that periodically output a random number.
What is a Random Beacon?

A set of servers that periodically output a random number.

- Some servers could maliciously “bias” the output
What is a Random Beacon?

A set of servers that periodically output a random number.

- Some servers could maliciously “bias” the output
- Need **unbiasability**: servers cannot influence the output in their favor
Contributions

- Elegant, scalable random beacon design
- For 100,000 participants, a random output can be produced every 20 seconds with only 3.05 MB of bandwidth (~5 minutes if many dishonest)
- Limiting factor is bandwidth: For 33 outputs × 3.05MB/output ≈ 100 MB, we can produce a random output every 0.6 to 10 seconds

<table>
<thead>
<tr>
<th></th>
<th>Participants</th>
<th>Time</th>
<th>Total Time Across System</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randherd</td>
<td>512</td>
<td>6s</td>
<td>&gt;200s</td>
<td>&gt;100 MB</td>
</tr>
<tr>
<td>Aleator</td>
<td>33,000</td>
<td>4s</td>
<td>8s</td>
<td>1 MB</td>
</tr>
</tbody>
</table>
Naive Random Beacon: Combine all

**Approach**: Combine all *random inputs* to produce random output
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output

Assuming they can agree on everyone's random inputs
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output

**Diagram:**
- **Servers**
  - $R_1$
  - $R_2$
  - $\ldots$
  - $R_n$

**Output:**
- **Random Output**
- Cannot exclude any random inputs
Naive Random Beacon: Combine all

**Approach**: Combine all *random inputs* to produce random output

**Problem**: Last participant controls random output
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output

**Problem:** Last participant controls random output
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output

**Problem:** Last participant controls random output
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output

**Problem:** Last participant controls random output
Naive Random Beacon: Combine all

**Approach:** Combine all *random inputs* to produce random output

**Problem:** Last participant controls random output
Naive Random Beacon: Commit-then-reveal

**Approach:** *Commit*-then-reveal random inputs

\[
\begin{align*}
c_1 &= C(R_1) \\
c_2 &= C(R_2) \\
\vdots \\
c_n &= C(R_n)
\end{align*}
\]
Naive Random Beacon: Commit-then-reveal

**Approach:** *Commit*-then-reveal random inputs

 Servers

 ![Diagram of servers](image)

 $c_1$

 $c_2$

 $c_n$
Naive Random Beacon: Commit-then-reveal

**Approach:** Commit-then-reveal random inputs
Naive Random Beacon: Commit-then-reveal

**Approach:** *Commit*-then-reveal random inputs
Naive Random Beacon: Commit-then-reveal

**Approach:** Commit-then-reveal random inputs

$$c_1 = C(R_1), \ldots, c_n = C(R_n)$$
Naive Random Beacon: Commit-then-reveal

**Approach:** Commit-then-reveal random inputs

\[ c_1 = C(R_1), \ldots, c_n = C(R_n) \]

Verify all Commitments

<table>
<thead>
<tr>
<th>Servers</th>
<th>Random Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td></td>
</tr>
<tr>
<td>R_2</td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>R_n</td>
<td></td>
</tr>
</tbody>
</table>
Naive Random Beacon: Commit-then-reveal

**Approach:** Commit-then-reveal random inputs

**Problem:** Dishonest participants refuse to reveal

\[
c_1 = C(R_1), \ldots, c_n = C(R_n)
\]
Naive Random Beacon: Commit-then-reveal

**Approach:** Commit-then-reveal random inputs

**Problem:** Dishonest participants refuse to reveal
Naive Random Beacon: Commit-then-reveal

**Approach:** Commit-then-reveal random inputs

**Problem:** Dishonest participants refuse to reveal
Naive Random Beacon: Commit-then-reveal

**Approach:** Commit-then-reveal random inputs

**Problem:** Dishonest participants refuse to reveal

![Diagram](image)

Servers

R₁

R₂

No Random Output Produced
Solution: Use a threshold signature scheme

\[ \sigma_1, \sigma_2, \ldots, \sigma_n \Rightarrow \sigma = \text{random output} \]

(e.g., DFINITY blockchain)
Solution: Use a threshold signature scheme

\[ \sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \sigma = \text{random output} \]

(e.g., DFINITY blockchain)
Digital Signatures: Motivation

Alice

M = “Hello, this is Alice.”

Bob
Problem: Mallory can pretend to be Alice to Bob

Mallory: "Hello, this is Alice."
Problem: Mallory can tamper with Alice's messages

M = “Hello, this is Alice.”

M’ = “Hello, this is John.”
Solution: Digital Signatures

Alice has her own secret key

Bob has Alice’s public key

(Diffie-Hellman '76, RSA '78)
Solution: Digital Signatures (Diffie-Hellman '76, RSA '78)

Alice

M = “Hello, this is Alice.”

σ = Sign(M, SK_{Alice})

Alice has her own secret key

Bob

Bob has Alice’s public key
Solution: Digital Signatures

(Diffie-Hellman '76, RSA '78)

M = “Hello, this is Alice.”
\[\sigma = \text{Sign}(M, \text{SK}_\text{Alice})\]

Alice has her own secret key

Bob has Alice’s public key
Solution: Digital Signatures (Diffie-Hellman '76, RSA '78)

M = “Hello, this is Alice.”
σ = Sign(M, SK_{Alice})

Alice has her own secret key

Bob has Alice’s public key

Verify(σ, M, PK_{Alice}) = true

M = “Hello, this is Alice.”
σ = Sign(M, SK_{Alice})
Naive Threshold Signatures

\[ \sigma_1 = \text{Sign}(M, SK_1) \]

\[ \sigma_2 = \text{Sign}(M, SK_2) \]

\[ \vdots \]

\[ \sigma_k = \text{Sign}(M, SK_k) \]
Naive Threshold Signatures

\[ \sigma_1 = \text{Sign}(M, SK_1) \]

\[ \sigma_2 = \text{Sign}(M, SK_2) \]

\[ \vdots \]

\[ \sigma_k = \text{Sign}(M, SK_k) \]
Naive Threshold Signatures

\[
\sigma_1 = \text{Sign}(M, SK_1) \\
\sigma_2 = \text{Sign}(M, SK_2) \\
\vdots \\
\sigma_k = \text{Sign}(M, SK_k)
\]

Verify(\(\sigma_1\), M, PK_1) = true

Verify(\(\sigma_2\), M, PK_2) = true

Verify(\(\sigma_k\), M, PK_k) = true
Naive Threshold Signatures

\[ \sigma_1 = \text{Sign}(M, SK_1) \]

\[ \text{Verify}(\sigma_1, M, PK_1) = \text{true} \]

\[ \sigma_2 = \text{Sign}(M, SK_2) \]

\[ \text{Verify}(\sigma_2, M, PK_2) = \text{true} \]

\[ \vdots \]

\[ \sigma_k = \text{Sign}(M, SK_k) \]

\[ \text{Verify}(\sigma_k, M, PK_k) = \text{true} \]

Too large
- k signatures

Too much time
- k verifications
Threshold Signatures

(Desmedt, CRYPTO 1987)

\[ \sigma_1 = \text{Sign}(M, SK_1) \]
\[ \sigma_2 = \text{Sign}(M, SK_2) \]
\[ \ldots \]
\[ \sigma_k = \text{Sign}(M, SK_k) \]

\[ M \]
\[ \sigma_1 \]
\[ \sigma_2 \]
\[ \ldots \]
\[ \sigma_k \]

Aggregator

Verifies signature shares

M
\[ \sigma \]

Single threshold signature
Threshold Signatures

\[
\sigma_1 = \text{Sign}(M, SK_1)
\]

\[
\sigma_2 = \text{Sign}(M, SK_2)
\]

\[
\sigma_k = \text{Sign}(M, SK_k)
\]

Verify(\(\sigma_i\), M, PK) = true

(Desmedt, CRYPTO 1987)

- **One** threshold signature
- **One** verification

Verify(\(\sigma\), M, PK) = true

\[
\sigma = \text{Sign}(M, SK)
\]

Signature Shares

Verifies signature shares

Aggregator

Single threshold signature
Random Beacon via Threshold Signatures

$\sigma_1 = \text{Sign}(M, SK_1)$

$\sigma_2 = \text{Sign}(M, SK_2)$

$\sigma_k = \text{Sign}(M, SK_k)$

Participants sign $M = \text{current time}$.

Random Output = Single threshold signature
Random Beacon Throughput

- Random beacon throughput = signature scheme throughput (assuming good network)
- High traffic at leader
- Multiple leaders ⇒ more throughput ⇒ more traffic :(

\[ \sigma_1, \sigma_2, \ldots, \sigma_n = \text{random output} \]
Random Beacon: Benefits of Threshold Signatures

Original Problems
● Last participant controls random output
● Dishonest participants refuse to reveal

Addressed using Threshold Signature Scheme
● Guaranteed to produce a signature, as long as k of the total n servers are honest
● Each message has a unique threshold signature
But... We Want a Scalable Random Beacon!

- Servers can be compromised
- Crucial to have a very large set of servers
- Can we get a **scalable** threshold signature scheme?
Shamir’s Secret Sharing

- Recover secret given k shares
Shamir’s Secret Sharing

- Recover secret given k shares

1 Point - Point
Shamir’s Secret Sharing

- Recover secret given k shares

1 Point - Point

2 Points - Line
Shamir’s Secret Sharing

- Recover secret given k shares
Lagrange Interpolation for Secret Sharing

Current implementations are inefficient
- Given k points, takes $O(k^2)$ time to recover secret

We use some known mathematical tricks to speed this up to $O(k \log^2 k)$ time

Net result: We can aggregate a threshold signature from 100,000 participants in 20 seconds rather than 13 minutes.
Our Results: Scalable Threshold Signatures

**Implementation Details:**
Implemented in C++
Used libff and libntl
Our Results: Scalable Threshold Signatures

Implementation Details:
Implemented in C++
Used libff and libntl

Machine Details:
ASUS ZenBook
Core i7-8550U CPU @ 1.80Ghz
16 GB of RAM

Ubuntu 16.04.5 LTS running inside VirtualBox 5.2.18 r124319
O(k^2) Naive Aggregation Time
O(k log² k) Efficient Aggregation Time
$O(k^2)$ Naive Aggregation Time

- **Time (s)**
- **Participants**
$O(k \log^2 k)$ Efficient Aggregation Time
Threshold Signatures: Not just for Random Beacons

Applications to:
- Consensus algorithms (such as the one used by Bitcoin)
- Securing HTTPs (every time you access a webpage)
Future Work

**Implement random beacon protocol**
- Threshold signature implementation works

**Verifying signature shares is computationally expensive**
- We speed it up using batch verification
- Fast when almost all shares are valid, slow when many are not

**More parallelization by decreasing traffic**
- Optimistically guess subset of k honest servers
Acknowledgements

I would like to thank:

● My mentor, Alin Tomescu, for his support and guidance
● Srini Devadas, for coordinating CS-PRIMES
● My parents and family
● MIT-PRIMES program
Thank you!

Questions?