RandShare: Small-Scale Unbiasable Randomness Protocol

Mathilde Raynal

Semester Project
Decentralized and Distributed Systems lab

**Responsible**
Prof. Bryan Ford
EPFL / DEDIS

**Supervisor**
Philipp Jovanovic
EPFL / DEDIS
Outline

• Motivation
  • Public Randomness
  • Towards unbiasable randomness

• RandShare

• RandSharePVSS
  • Implementation

• Results
  • Security properties
  • Experimental results

• Limitations

• Future Work
Public Randomness

Applications:
- **Random selection**: lotteries, sweepstakes, jury selection, voting and election audits
- **Games**: shuffled decks, team assignments
- **Protocols**: parameters, IVs, nonces, sharding
- **Crypto**: challenges for NZKP, authentication protocols, cut-and-choose methods, “nothing up my sleeves” numbers

Public Randomness Approaches Without Trusted Parties:
- **Bitcoin** (Bonneau, 2015)
- **Slow cryptographic hash functions** (Lenstra, 2015)
- **Financial data** (Clark, 2010)
What makes a «good» randomness?
Towards unbiased randomness

<table>
<thead>
<tr>
<th></th>
<th>Availability</th>
<th>Unpredictability</th>
<th>Unbiasability</th>
<th>Verifiability</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strawman I</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Strawman II</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Strawman III</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

**Strawman I**

**Idea:** Combine random inputs of all participants.

**Problem:** Last node controls the output.

**Strawman II**

**Idea:** Commit-then-reveal random inputs.

**Problem:** Dishonest nodes can choose not to reveal.

**Strawman III**

**Idea:** Secret-share random inputs.

**Problem:** Dishonest nodes can send bad shares.
RandShare

<table>
<thead>
<tr>
<th>Availability</th>
<th>Unpredictability</th>
<th>Unbiasability</th>
<th>Verifiability</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandShare</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

**Idea:** Strawman III + Verifiable Secret Sharing (Feldman, 1987)

**Problems:**
- Not scalable: $O(n^3)$ communication/computation complexity
- Not publicly verifiable
RandSharePVSS

**Idea:** RandShare + PVSS

- Publicly Verifiable Secret Sharing (PVSS)
  - Each node computes the collective string along with a transcript of the protocol run that includes all the shares used in the construction of the random output and proofs of their validity.
For the rest of the presentation, \( n \) will denote number of nodes, 
\[ f = \frac{n}{3} \] the number of faulty nodes and \( t = f + 1 \) the threshold.

Nodes only accept messages with a correct identifier, and a tracker ensures that we handle only one message per node per step.
RandSharePVSS

• Share Distribution
  • Secret splitting
  • Encryption then distribution with a proof
  • Check received shares against their proof, discard it if not verified
  • Done when \( f + t \) of them are received from every other node

• Voting Process
  • \( t \) secrets are enough for unpredictability
    • Choose a subset of servers
  • Vote for a node depends on how many correct shares we received from it
  • If a node receives too many negative votes, then it is discarded
RandSharePVSS

• Share Decryption
  • Decryption then distribution to nodes kept after voting process
  • When receiving a decrypted share from another node, check it against its proof
  • Done when at least \( t \) decrypted shares are collected and verified from every node

• Secret Recovery
  • Recover secrets through Lagrange interpolation
  • Combine them to create the collective string
  • Output it along with the transcript consisting of shares used and their proofs
# Security properties

<table>
<thead>
<tr>
<th></th>
<th>Availability</th>
<th>Unpredictability</th>
<th>Unbiasability</th>
<th>Verifiability</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strawman I</td>
<td>✗️</td>
<td>✗️</td>
<td>✗️</td>
<td>✗️</td>
<td>✗️</td>
</tr>
<tr>
<td>Strawman II</td>
<td>✗️</td>
<td>✓</td>
<td>✗️</td>
<td>✗️</td>
<td>✗️</td>
</tr>
<tr>
<td>Strawman III</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗️</td>
<td>✗️</td>
</tr>
<tr>
<td>RandShare</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗️</td>
<td>✗️</td>
</tr>
<tr>
<td>RandSharePVSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗️</td>
</tr>
</tbody>
</table>
Experimental Results

Implementation in Go, based on DEDIS code (Crypto library; Network library; Cothority framework). Deterlab Setup: 10 machines, each equipped with an Intel(R) Xeon(R) E3-1260L quad-core processor running at 2.4 GHz, 16GB of RAM, and imposed 200 ms round-trip latencies on all communication links.

Total wall-clock time of a protocol run

Wall clock time in second

Number of nodes

8 16 32 64 128

RandShare Randomness Generation
RandSharePVSS Randomness Generation
RandSharePVSS Transcript Verification
Demo

github.com/dedis/student_17_randomness
Limitations

• Lack of scalability
  • All-to-all communication pattern
  • PVSS is computationally expensive

• Attacks
  • Impersonation
  • Network Splitting

\[
\begin{align*}
n &= 5 \\
f &= 1 \\
t &= 2
\end{align*}
\]
Future Work

• Scale
  • SCRAPE

• Signing
  • \((t, n)\)-Threshold Schnorr Signature.

• Network Splitting Attack
  • Collective string combines \(2 \cdot f + 1\) secrets instead of \(f + 1\)