Building Blocks for Blockchains and Distributed Systems

Philipp Schindler
pschindler@sba-research.org
Randomness Beacons

```c
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
             // guaranteed to be random.
}
```

https://xkcd.com/221
Why Randomness Beacons?

Randomness

Leader and Verifier Selection
Proof of Stake
E-Voting
Tor Hidden-Services
Publicly-Auditable Selections
Gambling and Lottery Services
Smart Contracts
Sharding
Byzantine Consensus
Blockchains
Properties

- Bias-Resistance
- Public-Verifiability
- Unpredictability
- Guaranteed Output Delivery
- Scalability
- Liveness
- Energy Efficiency

Guaranteed Output Delivery
Approaches

Publicly-Verifiable Secret Sharing (PVSS)
  • Ouroboros, Scrape, RandHerd, HydRand

Verifiable Random Functions (VRFs)
  • Algorand, Ouroboros Praos

(Verifiable) Delay Functions (VDFs)
  • Bünz et. al. [1], Ethereum Casper?

Threshold Signatures (e.g. BLS)
  • HoneyBadger BFT, Dfinity

Secret Sharing

**Distribution**
- Dealer
- Participants: $S_1$, $S_2$, $S_3$, $S_4$, $S_5$

**Reconstruction**
- Subset of Participants: $S_2$, $S_4$, $S_5$
- Final Secret: $S$
## (Publicly-Verifiable) Secret Sharing

<table>
<thead>
<tr>
<th>Shamir’s Secret Sharing</th>
<th>Schoenmakers’ PVSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>• $(t, n)$ threshold scheme</td>
<td>• $(t, n)$ threshold scheme</td>
</tr>
<tr>
<td>• dealer distributes secret value $s$ to $n$ participants</td>
<td>• correctness of shares can be verified prior to reconstruction</td>
</tr>
<tr>
<td>• any set of at least $t$ participants can reconstruct $s$</td>
<td>• uses non-interactive zero knowledge proofs</td>
</tr>
<tr>
<td>• dealer must be trusted</td>
<td>• malicious dealers are detected</td>
</tr>
</tbody>
</table>
Randomness Beacon via PVSS

Every node performs the following steps

1. share a random secret with all parties
2. run (BFT) consensus protocol to agree on the shared values
3. a) reveal previously shares secret
   b) recover missing shared secrets
4. output new random beacon as combination of shares values
HydRand's Approach in a Nutshell

- integrated low overhead BFT protocol
- pipelining: only one PVSS per round
Verifiable Random Functions (VRFs)

- each node commits to a VRF public key $pk$
- obtain new random number $R$ privately
  \[ R, \pi = VRF(sk, seed \ || \ round) \]
- reveal $(R, \pi)$ if $R < threshold$ as leadership-credentials
- correctness verified using $pk$
- implemented e.g. using unique signatures and hashes in practice
Verifiable Delay Function (VDFs)
Unique Threshold Signatures

1. sign message using individual secret key

2. aggregate signatures

3. check signature via group public key
Unique Threshold Signatures

• share master secret key among nodes
  o requires trusted dealer or
  o distributed key generation protocol (DKG)
• each node signs seed (e.g. round index) using its private key share
• shares are checked for correctness
• aggregation of shares as soon as enough correct shares are obtained
Unique Threshold Signatures cont.

• aggregated signature serves as new random number
• can be checked against master public key
• typically using pairing based cryptography
  ○ BLS signature scheme
## Comparison

<table>
<thead>
<tr>
<th>PVSS</th>
<th>VRFs</th>
<th>VDFs</th>
<th>Thres. Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ bias-resistance</td>
<td>+ low communication overhead</td>
<td>+ low communication overhead</td>
<td>+ low communication overhead</td>
</tr>
<tr>
<td>+ no DKG</td>
<td>+ no DKG</td>
<td>+ bias-resistance</td>
<td>+ bias-resistance</td>
</tr>
<tr>
<td></td>
<td>+ leader privacy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- communication overhead</td>
<td>- bias-resistance not ensured</td>
<td>- timing assumptions</td>
<td>- requires DKG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- throughput</td>
<td>- requires pairings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- computation compl.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- parameter setup</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Comparison & Our Protocol

HydRand: Efficient Continuous Distributed Randomness
Philipp Schindler*, Aljosha Judmayer*, Nicholas Stifter†, Edgar Weippl†

1 Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQIL), TU Wien
2 SBA Research
Email: (firstletterfirstname lastname)@sba-research.org

Abstract—A reliable source of randomness is not only an essential building block in various cryptographic, security, and distributed systems protocols, but also plays an integral part in the design of many new blockchain proposals. Consequently, the topic of publicly-verifiable, bias-resistant and unpredictable randomness has recently enjoyed increased attention. In particular random beacon protocols, aimed at continuous operation, can be a vital component for current Proof-of-Stake based distributed ledger proposals. We improve upon previous random beacon approaches with HydRand, a novel distributed protocol based on publicly-verifiable secret sharing (PVSS) to ensure unpredictability, bias-resistance, and public-verifiability of a continuous sequence of random beacon values. Furthermore, HydRand provides guaranteed output delivery of randomness at regular and predictable intervals in the presence of adversarial behavior and does not rely on a trusted dealer for the initial setup. Compared to existing PVSS based approaches that strive to achieve similar properties, our solution improves scalability by lowering the communication complexity from $O(n^2)$ to $O(n^2)$. Furthermore, we are the first to present a detailed comparison of recently described schemes and protocols that can be used for implementing random beacons.

• the secure generation of protocol parameters for cryptographic schemes [4, 34]
• privacy preserving messaging services [49, 47, 30]
• protocols for anonymous browsing, including Tor hidden services [46, 31, 28]
• electronic voting protocols [1]
• publicly-auditable selections [15]
• gambling and lottery services [15]

With the emergence of blockchain protocols additional areas that demand secure sources of public randomness, such as sharding approaches [23], were formed. In particular smart contracts often draw upon insecure sources of randomness or trusted third parties [2], [17] such as the NIST random beacon, Random.org or Oraclize.it.

The revealed backdoor in the Dual Elliptic Curve PRNG [9], the unreliability of proprietary beacons [15], and the possibility of a centralized provider buffering, manipulating, and benefiting from prior knowledge of the provided randomness

Distributed Key Generation

Applications

• randomness beacons
• (BFT) consensus protocols
• custodian and escrow schemes
• smart contracts
• threshold and time-lock encryption
• ...

20
1. sign message using **individual** secret key

2. aggregate signatures

3. check signature via **group public key**
individual secret / public key pairs
individual secret / public key pairs
smart contract on the Ethereum blockchain

client application run by all the parties
Client:

- generate BLS keypair
- submit public key

Smart Contract:

- checks eligibility of client to register
Client:

- run VSS protocol for all registered parties
- submit encrypted shares and verification vectors

Smart Contract:

- "basic" validity checks on the submitted data
- store hash of the submitted data
Client:

- verifies all of its shares received
- submits a dispute for all invalid shares

Smart Contract:

- checks if a claimed dispute is valid
- [withdraw security deposit on success]
verify that all shares are valid

check that a single share is indeed invalid

if a party claims that
Client:

• derive set of qualified nodes
• submit / recover final key shares
• compute master public key

Smart Contract:

• derive set of qualified nodes
• verify master public key
Scalability

The chart illustrates the gas costs for different scenarios as the number of participants increases. The categories include:

- Block gas limit
- Share distribution
- Dispute
- Key share recovery
- Master key submission

The x-axis represents the number of participants (n), while the y-axis shows gas costs. The chart shows a significant rise in gas costs as the number of participants increases, with notable spikes at 128, 192, and 256 participants.
ETHDKG: Distributed Key Generation with Ethereum Smart Contracts

Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl.

Abstract
Distributed key generation (DKG) is a fundamental building block for a variety of cryptographic schemes and protocols, such as threshold cryptography, multi-party coin tossing schemes, public randomness beacons and consensus protocols. More recently, the surge in interest for blockchain technologies, and in particular the quest for developing scalable protocol designs, has renewed and strengthened the need for efficient and practical DKG schemes. Surprisingly, given the broad range of applications and available body of research, fully functional and readily available DKG protocol implementations still remain limited. We hereby aim to close this gap by presenting an open source, fully functional, well documented, and economically viable DKG implementation that employs Ethereum’s smart contract platform as a communication layer. The efficiency and practicability of our protocol is demonstrated through the deployment and successful execution of a DKG contract in the Ropsten testnet. Given the current Ethereum block gas limit, it is possible to support up to 256 participants, while still ensuring that the key generation process can be verified at smart contract level. Further, we present a generalization of our underlying DKG protocol that is suitable for distributed generation of keys for discrete logarithm based cryptosystems.

and Blakley [4]. However, in contrast to secret sharing, DKG protocols do not rely on a (trusted) dealer which generates, knows and distributes the secret key, and hence avoid this single point of failure. Instead, the keypair is generated using a multi-party computation in a way that no single party learns the secret that is being shared.

Distributed key generation is a topic that has been studied and discussed for over two decades [6, 21, 22, 26, 27, 30, 33]. However, the extensive body of literature is currently not matched by a single clear, succinct, and practical protocol design template that reflects the state of the art and leverages on recent technical developments such as distributed ledgers. Moreover, real-world open source implementations of DKG protocols are still rare, and often not well documented.

We aim to close this gap by providing and evaluating a lightweight, scalable, and well-documented protocol design and open source implementation of a DKG protocol. Our design is based on the Joint-Feldman DKG protocol [21] and incorporates the enhancements proposed by Neji et al. [30] to address biasing attacks [21], without requiring two distinct secret sharing rounds. Additionally, we describe and implement a new mechanism that handles disputes during the protocol execution more efficiently. The resulting protocol design is described in its generality for any discrete