Breeding Unicorns: Developing Trustworthy and Scalable Randomness Beacons

Samvid Dharanikota, Michael Jensen, Sebastian Kristensen, Mathias Michno, Yvonne-Anne Pignolet, Rene Hansen, Stefan Schmid
Randomness: An Important Tool of Our (Digital) Society

• In algorithm design:
  • Faster algorithms through randomization
  • Symmetry breaking in distributed algorithms
  • Secure cryptographic protocols based on random seeds (e.g., elliptic curves)

• In entertainment:
  • Lotteries, games (Roulette)
  • Drafts in sports

• In politics:
  • Military drafts, assignment of kids to schools

• Blockchain: more soon
Randomness: An Important Tool of Our (Digital) Society

- In *algorithm* design:
  - Faster algorithms through randomization
  - Symmetry breaking in distributed algorithms
  - Secure cryptographic protocols based on random seeds (e.g., elliptic curves)

- In entertainment:
  - *Lotteries, games* (Roulette)
  - Drafts in sports

- In politics:
  - Military *drafts*, assignment of kids to schools

- *Blockchain*: more soon

Can be solved by producing randomness locally myself (e.g., /dev/urandom, using quantum photonics, etc.)
Randomness: An Important Tool of Our (Digital) Society

- In *algorithm* design:
  - Faster algorithms through randomization
  - Symmetry breaking in distributed algorithms
  - Secure cryptographic protocols based on random seeds (e.g., elliptic curves)
- In entertainment:
  - *Lotteries, games* (Roulette)
  - Drafts in sports
- In politics:
  - Military *drafts*, assignment of kids to schools
- *Blockchain*: more soon

Requires randomness that is trustworthy for everybody! Output of „shared randomness“ is relevant to multiple stakeholders.
Randomness: An Important Tool of Our (Digital) Society

• In algorithm design:
  • Faster algorithms through randomization
  • Symmetry breaking in distributed algorithms
  • Secure cryptographic protocols based on random seeds (e.g., elliptic curves)

• In entertainment:
  • Lotteries, games (Roulette)
  • Drafts in sports

• In politics:
  • Military drafts, assignment of kids to schools

• Blockchain: more soon

How to realize such trustworthy shared randomness?
Vision: Randomness Beacon

- Vision: service emitting *unpredictable random values* …
- *Public*, seen by „everybody“
Vision: Randomness Beacon

- Vision: service emitting *unpredictable random values* …
- *Public*: seen by „everybody“
- Introduced by *Michael O. Rabin* in 1983
When Are Randomness Beacons Useful?

- When group of users need to agree on some random value but do not trust each other
- E.g.:
  - Lotteries
  - Secure elections
  - Prevent selfish mining
  - Blockchain/distributed systems‘ consensus mechanisms
  - Overcoming slow bootstrapping in zero-knowledge proofs
  - Rabin’s example: providing security to e-mail based protocols with minimal trust
When group of users need to agree on some random value but do not trust each other.

E.g.:

- Lotteries
- Secure elections
- Prevent selfish mining
- Blockchain/distributed systems' consensus mechanisms
- Overcoming slow bootstrapping in zero-knowledge proofs
- Rabin's example: providing security to e-mail based protocols with minimal trust

Trustworthy randomness: “A tool of democracy”
Available today?!

• E.g.: NIST‘s randomness beacon
• One new value per minute
Available today?!

- E.g.: NIST’s randomness beacon
- One new value per minute

Certificate

NIST Beacon 1.0 uses an X.509 certificate with the Federal Common Policy CA as the ultimate root authority for some of the records. The certificate is available here.

The Beacon Signing Key changed in May of 2017. Not all records will successfully validate.

Beacon Record

| Version: | Version 1.0 |
| Frequency: | 60 seconds |
| Time: | 12/26/2018 5:07 pm (1545840426) |
| Seed Value: | C2E5063FA9827C1E4F68A68E5BEC2A851B7519C295AEC78987F2A8549BCE |
| Previous Output: | 28F94CA35869465C444046672B0AF3F72039702ED532FF864858121318465577 |
| Signature: | B0994AE56A6D04CAFB881C12E84A11FC1FB6115C99732D88D9373C7C8C32 |

IEEE Blockchain, Atlanta, USA
Trustworthy?

Nonrandom Risk: The 1970 Draft Lottery

Abstract

The 1970 draft lottery for births is reviewed as an example of a government action at randomization whose incidence can be estimated by a wide variety of statistical approaches. Several methods of analyzing the data — which were of substantial importance to those concerned — are given explicitly. These methods, which were not of much interest to those concerned, are given explicitly. The corresponding data for 1971 and for 1972 are included.

1. Introduction

Nick Renn, a popular Pittsburgh television personality, and a state official were convicted in a county court here today of a million-dollar attempt to rig the Pennsylvania lottery.

The two were also convicted of committing perjury when they appeared before a Philadelphia grand jury that was investigating the attempt.
Trustworthy?

How to do better? Design choices and tradeoffs!
Design Choice 1: Input Sources

- **Private source**: e.g., NIST beacon
  - Potentially **high quality and high rate** randomness
  - But denies user access to inspect generation process
  - Requires trust: *not ok*
- **Publicly available data**: financial data, bitcoin block hashes, lottery data, weather (?)
  - Nice idea, *but* sufficiently random? Rate?
- **User input**: outsource to the users, i.e., locally generated randomness
  - Users responsible to provide input!
Design Choice 1: Input Sources

- **Private source**: e.g., NIST beacon
  - Potentially **high quality and high rate** randomness
  - But denies user access to inspect generation process
  - Requires trust: **not ok**
- **Publicly available data**: financial data, bitcoin block hashes, lottery data, weather (?)
  - Nice idea, **but** sufficiently random? Rate?
- **User input**: outsource to the users, i.e., locally generated randomness
  - Users responsible to provide input!

How random should it be? As random **as they need**!
Design Choice 2: Beacon Operator

- **Autocratic collector.** e.g. run by a third party
  - Computation is blackbox, *no proof of honesty*

- **Specialized Multi-Party Computation (MPC):** collectively produce randomness
  - … typically from their own inputs
  - Despite significant work in the field, this approach is *difficult to scale*
  - *Addition or removal of a user* requires a new setup phase
  - But might fit in a controlled private context

- **Transparent authority model:** single entity collects inputs and publishes them
  - Focus on *transparency*, users can observe and verify the beacon
Design Choice 2: Beacon Operator

- **Autocratic collector**: e.g. run by a third party
  - Computation is blackbox, *no proof of honesty*
- **Specialized Multi-Party Computation (MPC)**: collectively produce randomness
  - … typically from their own inputs
  - Despite significant work in the field, Byzantine behavior possible, but *difficult to hide*
  - Addition or removal of a user requires a new setup phase
  - But might fit in a controlled private context
- **Transparent authority model**: single entity collects inputs and publishes them
  - Focus on *transparency*: users can observe and verify the beacon
Our Contributions

• A randomness beacon requiring *minimal trust*
  • Based on the *transparent authority* model which relies on *user input*
  • Beacon operator has *no private information*

• Allows users to
  • *Join any time* and at low overhead
  • Make subtle decisions on *when to trust* the output

• Practical:
  • Prototype demonstrates *scalability* of our approach
  • Beacon can be deployed on distributed *ledger platforms*
Our Approach in a Nutshell

- No private information: all inputs are hashed and released to the public in batches before the computation.
- Uses commitments and verifiable delay function to ensure that the operator cannot try more than one commitment before running out of time.
- The beacon protocol also lets users verify:
  - inclusion of their input in the randomness generation (by committing to user inputs before starting the computation)
  - correct calculation of the random value from the provided inputs
- Several practical optimizations (e.g., for scalability)
A Deeper Dive

• To support “choosable” randomness: a user’s input rate is not limited (except for DoS)

• To increase security.
  • Fast and transparent publishing: input, output, and any data needed for verification needs to be published as soon as possible
  • “Deterministic”: any party can compute the randomness alongside the beacon operator
  • Open: anyone can contribute to the beacon to influence random generation

• For scalability: different channels for input and output
A Deeper Dive

- **To support „choosable“ randomness:** a user’s input rate is not limited (except for DoS)
- **To increase security.**
  - **Fast and transparent publishing:** input, output, and any data needed for verification needs to be published as soon as possible
  - **“Deterministic”:** any party can compute the randomness alongside the beacon operator
  - **Open:** anyone can contribute to the beacon to influence random generation
- **For scalability:** different channels for input and output

*Low market value of output: everyone can compute it!*
Delay Function

• Inherently *sequential functions* that require a given amount of time to run
  • No benefit from parallel execution
  • E.g., *sloth*, but also others

• Idea:
  • Cannot compute the delay function in the time between the users‘ input and the receiving the beacon’s commitment to the input:
    \[ t_{\text{COMMITMENT}} - t_{\text{INPUT}} < T_{\text{DELAYFUNCTION}} \]
  • Operator cannot try more than one commitment before running out of time
  • But while hard to compute, *easy to verify*! E.g., *sequential hashing.*
Delay Function

• Inherently \textit{sequential functions} that require a given amount of time to run
  
  • No benefit from parallel execution
  
  • E.g., \textit{sloth}, but also others

• Idea:
  
  • Cannot compute the delay function in the time between the users’ input and receiving the beacon’s commitment to the input:
    \[ t_{\text{COMMITMENT}} - t_{\text{INPUT}} < T_{\text{DELAYFUNCTION}} \]
  
  • Operator cannot try more than one commitment before running out of time
  
  • But while hard to compute, \textit{easy to verify}! E.g., \textit{sequential hashing}.  

User can choose threshold
Delay Function

• Inherently *sequential functions* that require a given amount of time to run
  • No benefit from parallel execution
  • E.g., *sloth*, but also others

• Idea:
  • Cannot compute the delay function in the time between the users’ input and receiving the beacon’s commitment to the input:
    \[ t_{\text{COMMITMENT}} - t_{\text{INPUT}} < T_{\text{DELAYFUNCTION}} \]
  • Operator cannot try more than one commitment before running out of time
  • But while hard to compute, *easy to verify*! E.g., *sequential hashing*.

Prevents *input manipulation* and *last-draw attacks*. 

User can choose threshold

IEEE Blockchain, Atlanta, USA
Combining Inputs

• During the input collection phase, users independently submit inputs to the beacon

• These inputs are aggregated into a Merkle Tree: the root is *input to the computation phase*

• This can also be used as commitment data that allows for *efficient verification of inclusion of a user’s input* in logarithmic time
Pipelining

• Beacon operation: *input collection phase* followed by *computation phase*

• But: computation (delay function) takes much time, where users *cannot submit input*

• Solution: parallel beacon operation
  
  • Several delay functions *run in parallel*, but *offset* in time and on different input
Proof-of-Concept Implementation

- Based on Python and ZeroMQ
- Two proxies:
  - *Forward proxy* between computation and publishers (forwards outputs, commitments, and proofs)
  - *Stream proxy* situated between input collectors and input processors
- Publicly available at: [https://github.com/randomchain/randbeacon](https://github.com/randomchain/randbeacon)
Evaluation

**Stream proxy** (bottleneck) can handle 1000s of msg/sec

Building **Merkle tree** is fairly fast

**Sloth** computation time for different parameters

**High asymmetry**: computation time vs verification time
Distributed Ledger Implementation, e.g.: Lottery Application

- **Lottery:** the random value determines a winner randomly from the participants
- **Tradeoffs** how much of the beacon is **on-chain** (as a smart contract) and how much is **off-chain**
- 3 players:
  - **Owner** - Runs the lottery (e.g. smart contract owner)
  - **User** - Takes part in the lottery by sending a small payment to the lottery smart contract
  - **Beacon** - Beacon operator that provides a random value / lucky winner
- Goal of **lottery owner:** shave off some of the users’ participation payments as a **reward**
- Users only participate if they trust random value provided by the beacon
# Lottery: Implementation Options

<table>
<thead>
<tr>
<th>Step / Version</th>
<th>Full-chain beacon</th>
<th>Delay: Function on-chain</th>
<th>Delay: Function off-chain, growing contract complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing</td>
<td>OFF</td>
<td>OFF</td>
<td>ON-chain, together with lottery fee</td>
</tr>
<tr>
<td>User Input</td>
<td>ON-chain, together with lottery fee</td>
<td>ON-chain, together with lottery fee</td>
<td>Like ITN</td>
</tr>
<tr>
<td>Beacon Commitment</td>
<td>Not necessary</td>
<td>OFF on-chain if users are not required to trust the lottery system</td>
<td>Like ITN</td>
</tr>
<tr>
<td>Beacon Computation</td>
<td>ON-chain</td>
<td>OFF on-chain, independent of lottery</td>
<td>Like ITN</td>
</tr>
<tr>
<td>Beacon Output</td>
<td>ON-chain</td>
<td>NEW beacon value on chain</td>
<td>Like ITN</td>
</tr>
<tr>
<td>Post-processing</td>
<td>OFF on-chain, independent of lottery</td>
<td>OFF on-chain</td>
<td>Like ITN</td>
</tr>
<tr>
<td>Reward</td>
<td>Users do not have to worry about verification</td>
<td>Users do not have to worry about verification</td>
<td>Like ITN</td>
</tr>
<tr>
<td>Fees</td>
<td>Users do not have to worry about verification</td>
<td>Users do not have to worry about verification</td>
<td>Like ITN</td>
</tr>
<tr>
<td>Cost</td>
<td>Requires many users to offload the on-chain computation cost, ensures all beacon operators can be verified</td>
<td>Users do not have to worry about verification</td>
<td>Like ITN</td>
</tr>
</tbody>
</table>
Maximum Offchain

Lottery Implementation Options

All beacon-related logic is **off-chain**, only the lottery logic is on-chain:

- The users **send inputs** to the beacon off-chain and **obtain commitment** off-chain
- Thereafter they **send the lottery payment** to the smart contract
- When the off-chain beacon value computation has finished, the lottery smart contract **fetches the value** and determines the **winner**
- Users can **verify** if the beacon matches the commitment and complain off-chain and decide not to trust this beacon in the future.
Adding beacon incentive and on-chain commitment (ITV): Compensated beacon operator with the smart contract

- The beacon publishes its public key and a nonce and **locks some funds** in the smart contract
- The beacon **loses** its locked funds **if verification fails**.
- If the verification succeeds, the owner, beacon and winner **receive rewards**.
- The verification of the correct execution of sloth on the Merkle root is performed on-chain.
Ethereum Smart Contract and Gas Cost

- Implemented an *Ethereum* smart contract
- Figure shows the *gas costs* for fetching the value from the beacon and drawing a winner for different number of users for the OFF model
- As expected, *linear increase* of the gas cost
Conclusions

• Design and implementation of a randomness beacon
  • Transparent and open: no trust needed
  • Beacon output can be publicly verified
• Performance study, also of the sloth protocol
• Lottery case study: realization tradeoffs

• Much future work. E.g., while nobody can bias the output, some parties may still know outcome a bit earlier than others.
Thanks. Questions?