STROBE:
Stake-based Threshold Random Beacons

Donald Beaver Konstantinos Chalkias Mahimna Kelkar
Lefteris Kokoris Kogias Kevin Lewi Ladi de Naurois
Valeria Nikolaenko Arnab Roy Alberto Sonnino

Novi Research, Facebook

October 1, 2021

Abstract

We revisit decentralized random beacons with a focus on practical distributed applications. Decentralized random beacons (Beaver and So, Eurocrypt'93) provide the functionality for n parties to generate an unpredictable sequence of bits in a way that cannot be biased, which is useful for any decentralized protocol requiring trusted randomness.

Existing beacon constructions are highly inefficient in practical settings where protocol parties need to rejoin after crashes or disconnections, and more significantly where smart contracts may rely on arbitrary index points in high-volume streams. For this, we introduce a new notion of **history-generating decentralized random beacons** (HGDRBs).

Roughly, the *history-generation* property of HGDRBs allows for previous beacon outputs to be efficiently generated knowing only the current value and the public key. At application layers, history-generation supports registering a sparser set of on-chain values if desired, so that apps like lotteries can utilize on-chain values without incurring high-frequency costs, enjoying all the benefits of DRBs implemented off-chain or with decoupled, special-purpose chains. Unlike rollups, HG is tailored specifically to recovering and verifying pseudorandom bit sequences and thus enjoys unique optimizations investigated in this work.

We introduce STROBE: an efficient HGDRB construction which generalizes the original squaring-based RSA approach of Beaver and So. STROBE enjoys several useful properties that make it suited for practical applications that use beacons:

1. **history-generating**: it can regenerate and verify high-throughput beacon streams, supporting sparse (thus cost-effective) ledger entries;
2. **concisely self-verifying**: NIZK-free, with state and validation employing a single ring element;

3. **eco-friendly**: stake-based rather than work based;
4. **unbounded**: refresh-free, addressing limitations of Beaver and So;
5. **delay-free**: results are immediately available.

1 Introduction

A random beacon is a shared source of agreed-upon random bits. First introduced in 1981 by Rabin [49] in the context of digitally-signed documents, beacons use unpredictability to put adversarial strategies in doubt. Trusted beacons are increasingly recognized as a critical resource for Decentralized Finance (DeFi), blockchains, Byzantine Fault Tolerance (BFT), leader elections, and a range of Decentralized Applications (dApps) including lotteries.

At fundamental protocol layers such as consensus, beacons drastically reduce the time and effort needed to withstand protocol-defeating attacks. For example, Feldman and Micali [34] implemented a common coin to achieve 2-round probabilistic consensus, breaking the deterministic lower bound of $f + 1$ rounds [30].

At higher levels of abstraction, new and important properties and optimizations emerge. Crash failures, delays, and asynchrony in the interactions between processes or the execution of smart contracts in a ledger environment make it essential to have stronger and coordinated record-keeping, to make it easier to retrieve and verify past results.

To complicate matters, many high-level applications like lotteries or gaming require high-frequency streams. Frequencies and latencies at the scale of seconds are too slow. Registering streams bit-by-bit in real-time on a ledger is simply infeasible given transaction throughputs and latencies, let alone enormously cost-prohibitive fees.

Registering intermittent results is a feasible and cheaper workaround - as long as the intermittent results provide sufficient content and validation to enable optimistic fault detection and recourse for any on-chain relying parties. To date, such applications either run at very slow speeds (lotteries) or require centralized trust (gaming platforms).

While a naive approach might register a long sequence accompanied by aggregate signatures and rollups to prove step-by-step correctness, so that relying parties can check content and validity, we take a novel and alternative approach, employing **history generation** for streamlined communication size and speed. Our protocols are direct and simple, providing a concise element to *regenerate* the entire back sequence (effectively compressing it) and to *validate* it (obviating NIZKs, let alone aggregation).

Decentralized random beacons. Beaver and So presented the first decentralized beacon (DRB) in [7], achieved by way of a threshold homomorphic secret sharing of future bits. Their approach capitalized on the Blum-Blum-Shub pseudorandom generator [9], in which successive squares in an RSA ring are produced. By reversing the sequence and employing tricks for threshold Lagrange interpolation, [7] gave a *self-certifying sequence* obviating any need for ZKPs.

History-generating DRBs. A key contribution of this work is the notion and implementation of *history generation*, a technique that is essentially equivalent to compressing

long sequences and providing a maximally concise validity check. In the context of random streams produced by DRBs, we are able to take strong advantage of the sequential nature of the stream to achieve optimizations exceeding those available to general-purpose calculations-with-ZKPs.

Unbounded sequences. In the threshold homomorphic VSS approach of [7], there is a bound on the number of bits to be produced without engaging in some kind of refreshing state. The results are limited by stakeholders rather than VDF-style delays: there is no explicit notion of using work as a computationally-guaranteed limit on the cadence of a beacon. We address both limitations using a generalized construction without incurring proof-of-work.

Roadmap. In the sequel, we describe intuitions of our constructions and how applications can benefit from the novel, history-generating property. We cover related work in §2, with particular attention to feature tradeoffs in Table 1. Details of the construction appear in §3, with security model in §4 and proofs in §5. We describe some extension in §6 and application details in §7.

1.1 Construction Intuition

Our starting point is to view RSA decryption as a trapdoor one-way function in reverse, which can be efficiently verified. The beacon output of an epoch is essentially the RSA decryption of the previous epoch's output, and this carries on perpetually. The verification of an output is to just RSA encrypt it with the public key and see that if it equals to the previous epoch's output. In this sense, the beacon is *self-certifying*.

Viewed in a non-distributed setting, the setup of the beacon generates RSA modulus $N = pq$ along with a root-ing parameter s. Then the beacon proceeds perpetually as:

$$x \to x^{1/s} \to x^{1/s^2} \to \cdots \to x^{1/s^T} \to \cdots$$

To parties that just know about N as a public parameter, this provides some attractive properties: (1) The next value in the beacon is hard to predict given the earlier values. (2) It's easy to verify a beacon value against the last value. In fact, we can check the value against any historical value, except that it gets progressively harder with the gap. (3) An especially tantalizing property is that any historical beacon value, can in fact be *generated* by just knowing the current value.

Of course, the problem with this is that taking roots in an RSA ring is hard without knowing the prime factors p and q. So we need a trusted party holding the primes to be generating all the beacon values.

Recent advances in distributed RSA modulus generation allows us to generate a public modulus N, with no single party knowing the factors. Imagine that in addition we also give secret shares of s^{-1} $(mod\ \phi(N))$ to n distinct parties:

$$sk_1 + sk_2 + \cdots + sk_n = s^{-1}\ (mod\ \phi(N))$$

At the time epoch $T+1$, the n parties can then output $x_T^{sk_i}$ each, where x_T is the output of the last epoch. On multiplying all the public shares, we get $x_{T+1} = x_T^{s^{-1}}$ $(mod\ N)$. At the same time, observe that even $(n-1)$ of the secret keys are effectively independently random. Also observe every public share is also self-certifying wrt the last epoch, in the sense $x_T^{sk_i} = (x_{T+1}^{sk_i})^s$.

Adapting the n-of-n setting to a threshold t-of-n setting introduces additional challenges which do not affect known-order groups. Essentially, Shamir secret sharing involves fractional Lagrange interpolation coefficients which are efficient to compute to group elements if we know the order. However, this is not possible to do in the exponent of RSA group elements, as $\phi(N)$ is not public. We adapt and extend the techniques pioneered by [7] and also used by [52] to address this challenge. The core trick is to lift the Lagrange coefficients by a factor of $n!$, so that they are not fractional anymore.

2 Related Work

Random Beacons. Practical bias-resistant random beacons producing regular series of random outputs typically follow one of three approaches: the first one uses publicly verifiable secret sharing (PVSS) mechanisms, the second uses threshold signatures (or homomorphic VSS) and a distributed key generation (DKG), and the third relies on verifiable delay functions (VDFs).

Beacons of the first type use the following blueprint design, they need 4 rounds for n participants to generate a random value. In the first round all nodes simultaneously secret-share a freshly generated random value s among the rest of the nodes. They do so by publishing the following values: n shares of s each encrypted under receiving party's public key, a commitment to the secret, and a non-interactive zero-knowledge proof that those were generated correctly. In the second round, the parties run some sort of consensus algorithm to agree which nodes did the sharing correctly and which failed. This can also be replaced with posting the shares on chain where anyone can verify the proofs independently, although this typically relies on consensus provided by a chain (rather than enabled by the beacon). In the third round, the parties reveal their secrets and in the fourth round for parties that withheld their secrets, shares of those secrets are broadcast for recovery. The resulting beacon's value is derived from the revealed or recovered secrets. The rounds can be pipelined to get regular random outputs at each round. Starting from the original proposal of Ouroboros [41] a sequence of papers (Scrape [22], Albatross [23], HydRand [51], RandHerd [53]) has improved the communication, computation complexity per node and the verification complexity for beacon of this type. No centralized or trusted setup and standard cryptographic assumptions are the main advantages of those protocols. However, those protocols still remain communication-intensive, since they need to run the full protocol (including consensus) for every fresh random value and computation wise intensive, since every party needs to check that every secret-share has been correctly constructed. A public verifier needs $O(n)$ messages to verify each beacon's value, making it communication-wise expensive to verify a series of random values.

A full chain of our beacon, in contrast, can be verified using only the current beacon's output and the public parameters.

Beacons of the second type rely on a setup phase where a secret key is generated in a distributed manner, after which homomorphic VSS and/or threshold signatures can be employed. The use of homomorphic VSS was pioneered in the first DRB in 1993 [7], where successive values of a BBS generator are revealed. The shares of square roots are, homomorphically and similar in spirit to threshold RSA, square roots of shares. The shares and the reconstructed values each act as verifiers of previous values (*viz.* by way of squaring). This particular solution suffers from a need to refresh to new sequence values periodically; it is not unbounded as-is.

In numerous other, generalized approaches of this second style, nodes can use a unique threshold signature scheme, such as threshold BLS [14,15] (tBLS) in Dfinity [39] and drand [32] or as threshold RSA (tRSA) signatures [52] in Cachin et al. [20]) to sign an agreed-upon progression of values (either block-hashes, or round-numbers, or a combination of those). The main advantage of those protocols is efficiency (each party only sends a single message per beacon's value) and ease of public verifiability (current beacon's value can be verified using only public parameters). The disadvantages are a complicated setup phase (though straightforward when a trusted party is assumed), for example to generate the threshold key for tBLS over the Internet it still takes elaborate protocols with $\mathrm{O}(n^4)$ communication complexity costs [40,44] and $O(f)$ worse case run time. We improve on the approach of tRSA by building a random beacon straight from the RSA assumption requiring no additional proofs for correctness of signature's shares. Our scheme naturally gives a novel property of history generation in contrast to existing approaches.

There is also a third approach that relies on a proof-of-delay mechanism [10]: either a block-hash is passed through a verifiable delay function (VDF), or the values submitted by the participants are passed through a VDF function to generate a random value. Most prominent systems are RANDAO w. VDF [31], continuous VDF [33] and RandRunner [50]. But those approaches are highly computationally intensive for the prover and require precise estimates of concrete complexity, which are hard to predict in practice (competitions with high-reward incentives were set-up by Chia Networks (`chia.net`) and Ethereum (`ethereum.org`) in partnership with Protocol Labs (`protocol.ai`) to get concrete estimates of VDFs' complexity). In the presence of a quantum adversary a quantum-resistant VDF could be used to produce a quantum-resistant random beacon, e.g. Veedo [54].

Distributed generation of an RSA modulus and an inverse of a public exponent. The setup of our construction requires the generation of an RSA modulus N that is a product of two primes. The most common way of generating such a modulus in a centralized setting (a.k.a. with a trusted setup), is to randomly sample κ-bits integers running Miller-Rabin probabilistic primality tests on them [46,48] until two primes are obtained with overwhelming probability, multiplying them gives a 2κ-bits bi-prime N. Since currently there is no known way to sample a bi-prime (using only public randomness) for which nobody knows the factors, the only way to alleviate the trusted setup is to distribute the generation of the bi-prime modulus via a dedicated multi-party computation protocol. Boneh and Franklin [12, 13]

	History gen	Self certifying	Single round	Refresh-free	BA-free	ROM-free	Setup	Assumption
Albatross [23] HydRand [51] RandHerd [53]	**N**	**N**	**N**	**Y**	**Y**	**N**	none	PVSS
RandRunner [50] RANDAO++ [31] cVDF [33] Veedo [54]	**N**	**N**	**Y**	**Y**	**Y**	**N**	yes/no	VDF
Dfinity [39] drand [32]	**N**	**N**	**Y**	**Y**	**Y**	**N**	DKG	tBLS
C03 [18] BS93 [7]	**N**	**Y**	**Y**	**N**	**Y**	**Y**	mod.gen.	RSA
STROBE (this work)	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	mod.gen.+ inverse	RSA

Table 1: Comparison among several beacon protocols. **Self-certifying** beacons give inexpensive verification of a beacon value against the previous one with no zero-knowledge proofs. **Refresh-free** beacons allow for generation of indefinite sequence of random values per single setup. **BA-free** beacons do not need to use Byzantine-Agreement protocols. **ROM-free** beacons do not rely on a random oracle assumption. Beacons based on the RSA assumption require a trusted setup or a distributed RSA modulus generation (**mod.gen**), our protocol additionally requires the generation of an inverse of a public exponent. Some VDF-based beacons do require a setup: those based on RSA.

initiated the study of distributed RSA modulus generation devising a protocol in a passive security model with honest majority. The follow-up protocol of Algesheimer, Camenisch and Shoup [3] devised a protocol for generation of N that is a product of two *safe* primes, passively secure with honest majority. The follow-up work has hardened the original Boneh-Franklin's protocol to be secure in the presence of actively malicious parties and honest majority [35]. The works mentioned above also generate an inverse (RSA decryption key) in a distributed manner. An improvement to this part of the protocol was also made by Catalano et al. [24]. A promising approach of getting rid of the setup phase altogether, that was proposed in RandRunner [50] and in the work of Damgård and Koprowski [28], can potentially get applied to this work.

3 The STROBE Protocol

We now define the syntax of a History Generating Decentralized Random Beacon (HGDRB) and describe our STROBE construction.

Definition 1 (HGDRB). *A History Generating Decentralized Random Beacon (HGDRB) is a set of algorithms* $(Setup, Gen, Eval, VerifyShare, Combine, Verify, Back)$*:*

Setup: $(\lambda, n, t) \to (pk, sk_1, \cdots, sk_n)$**.** *The Setup algorithm takes the security parameter* λ *and threshold parameters* n *and* t*. The scheme allows* t*-of-*n *reconstruction, with secret shares given to* n *parties. The output is a public key* pk *and secret shares* $sk_1, \cdots, sk_n$*.*

Gen: $pk \to x_0$**.** *The (one-time) Gen algorithm samples an initial random value* x_0*.*

Eval: $(sk_i, x_T) \to x_{T+1,i}$**.** *Each party takes the last epoch's* (T) *output* x_T*, computes and outputs a share of the next epoch's output* $x_{T+1,i}$*.*

VerifyShare: $(pk, x_{T,i}, x_{T+1,i}) \to \{0, 1\}$**.** *The VerifyShare algorithm checks the epoch* $T+1$ *shares against the corresponding epoch* T *shares.*

Combine: $(pk, x_{T,P_1}, x_{T,P_2}, \ldots, x_{T,P_t}) \to x_{T+1}$**.** *Given* t *shares from epoch* T *that pass* VerifyShare *the Combine algorithm outputs the next epoch's beacon value* x_{T+1}*.*

Verify: $(pk, x_T, x_{T+1}) \to \{0, 1\}$**.** *The Verify algorithm checks the epoch* $T+1$ *beacon output against the epoch* T *beacon output.*

Back: $(pk, x_T, k) \to x_{T-k}$**:** *This outputs the beacon value at epoch* $T-k$ *given the epoch* T *beacon value* x_T *and* $k < T$*.*

Informally, correctness asserts that honestly computed beacon values will pass verify checks with respect to previous beacon outputs. The same should hold for share outputs as well. The *Back* function allows to compute any historical beacon value efficiently (going back a polynomial number of epochs).

The STROBE protocol. We now provide our HGDRB construction, called *STROBE*. It is based on threshold inversion in RSA groups and its security follows from the RSA assumption.

Setup(λ, n, t)**:** The Setup algorithm takes the security parameter λ and samples an RSA modulus $N = pq$ with $\phi(N) = 4p'q'$.

Sample primes p, q such that $p - 1 = 2p', q - 1 = 2q'$ with p', q' also being primes. Pick a prime s, s.t. $\min(p', q') > s > n$. Let $N = pq$ and observe that $s \nmid \phi(N)$. Sample $a_1, \cdots, a_{t-1} \leftarrow [1, N]$ and let $f(X) = v + a_1 X + \cdots + a_{t-1} X^{t-1}$, where $v = (n!s)^{-1} \pmod{p'q'}$.

Send secret shares $sk_i = f(i) \pmod{N}$ to parties $i \in [1, n]$. Publish $pk = (N, s)$.

Gen(pk)**:** The Gen algorithm samples a seed value $\mathsf{seed} \leftarrow [1, N]$. This is a public random value that can be computed by MPC or by taking a block hash. It then outputs $x_0 = \mathsf{seed}^{4(n!)^2} \pmod{N}$.

The $4(n!)^2$ factor is an artifact of the security proof and will be explained in Section 5.

Eval(sk_i, x_T)**:** Each party takes the last epoch's (T) output x_T, computes and outputs a share of the next epoch's output $x_{T+1,i}$:

$$x_{T+1,i} = x_T^{sk_i} = x_T^{f(i)} \pmod{N}.$$

VerifyShare$(pk, x_{T,i}, x_{T+1,i})$**:** The epoch $T+1$ shares can be self-verified against the corresponding epoch T shares, by checking that

$$x_{T+1,i}^s = x_{T,i} \pmod{N}.$$

Combine$(pk, x_{T,P_1}, x_{T,P_2}, \ldots, x_{T,P_t})$**:** Given t epoch T shares, first check that each of them pass VerifyShare. Let γ denote the set of these indices $\{P_1, \cdots, P_t\}$. The Combine algorithm then computes the combined epoch T beacon value x_T by computing the interpolation:

$$x_T = \prod_{i \in \gamma} x_{T,i}^{n! L_i(0)} \pmod{N},$$

where the Lagrange basis polynomials L_i's are defined as:

$$L_i(X) = \prod_{j \in \gamma, j \neq i} \frac{X - j}{i - j}.$$

The polynomials satisfy the following property: for $\forall i, j \in \gamma : L_i(i) = 1$, and $L_i(j) = 0$ for $i \neq j$. The $(n!)$ factor is essential to clear the denominators of the interpolation coefficients, which makes sure there is no fractional exponent to compute.

Verify(pk, x_T, x_{T+1})**:** The epoch $T+1$ beacon output can be self-verified against the corresponding epoch T beacon output, by checking that

$$x_{T+1}^s = x_T \pmod{N}.$$

Back(pk, x_T, k)**:** This outputs the beacon value at epoch $T - k$ as:

$$x_{T-k} = x_T^{s^k} \pmod{N}.$$

We assume that each share carries the identity information (metadata) about which party generated it. Apart from the $Verify$ (and $VerifyShare$) algorithms checking against the last epoch outputs, the *Back* algorithm provides an alternate way to check against any previous epoch output, all the way to epoch 0 (and 1). In fact if the shares from epoch 1 are also made part of the trusted pk, then this provides a way of checking the beacon value without accessing the past beacon values at all. The trade-off is that checking against beacon values in the past grows computationally expensive with the number of epochs elapsed. As an extension of the core protocol, we can also leverage techniques from popular VDF constructions, as described in Section 6.

Correctness. We have $x_0 = \mathsf{seed}^{4(n!)^2}$. Assume inductively, $x_{T-1} = \mathsf{seed}^{4(n!)^2 s^{-T+1}}$. Correctness of the beacon outputs follows as below:

$$x_T = \prod_{i \in \gamma} x_{T,i}^{n! L_i(0)} = x_{T-1}^{n! \sum_{i \in \gamma} f(i) L_i(0)} = \mathsf{seed}^{4(n!)^2 s^{-T+1} n! \sum_{i \in \gamma} f(i) L_i(0)}$$

$$= \mathsf{seed}^{4(n!)^2 s^{-T+1} n! f(0)} = \mathsf{seed}^{(n!)^2 s^{-T} 4(n!) vs}$$

As $v = (n!s)^{-1} \pmod{p'q'}$, we have $4(n!)vs = 4 \pmod{\phi(N)}$. Therefore, $\mathsf{seed}^{4(n!)vs} = \mathsf{seed}^4 \pmod{N}$.

Substituting, we get:

$$x_T = \mathsf{seed}^{4(n!)^2 s^{-T}}.$$

This carries the induction successfully forward, and also leads to successful verification: $x_T^s = x_{T-1}$. Similar steps apply to the individual shares as well.

4 Security Model

There are various flavors of security that we could require of a random beacon. To narrow down the syntax, we will focus on *stake-based, self-certifying, threshold* beacons. The baseline security we want is that an adversary should not be able to predict future beacon values based on seeing past values and corrupting less than a threshold number of participants.

Unpredictability vs. Pseudorandomness. We could require the next beacon value to be pseudorandom, instead of just unpredictable. We observe that we could essentially compile an unpredictable beacon into a pseudorandom one, either by applying a random oracle (similarly to what is described as "tick-tock" in [33]), or if we want to avoid the RO assumption, by extracting hardcore bit(s), as in [9]. There exist applications where unpredictability suffices, but in most of the cases, such as a decentralized lottery or leader election, unbiased randomness is essential. .

We also note that, for a self-certifying and/or history generating beacon, there needs to be a part of the beacon output that cannot be pseudorandom, as otherwise it cannot be used for the certification assertion and/or historical value computation.

Active vs. Passive. A passive adversary just observes the transcript of an honest run of the protocol, including public shares and beacon outputs and then tries to predict the next beacon value. An active adversary, on the other hand, can actively modify the public share values and beacon outputs. In this paper we consider active adversaries. In our setting, the self-certification essentially ensures that there is only a unique value for each expected public share or beacon value that can pass onto the next phase. This makes any adversarial modifications immediately noticeable and subject to rejection.

Selective vs. Adaptive. Another dimension to specify the adversary is on whether we restrict it to corrupt parties that it declares upfront (selective), or allow the parties to be corrupted dynamically as it observes and interacts with the protocol (adaptive). Our core protocol only satisfies selective security in this respect. We leave it as an open problem to construct an adaptively secure protocol which retains the efficiency standard of our selective one. A generic (complexity-leveraging) approach can be used to get an adaptively secure scheme from a selectively secure one: the reduction simply needs to guess the set of corrupted parties and then run the selective reduction aborting if the selection of corrupted parties does not match the one requested by the adversary. This, unfortunately, leads to a loss in security that is exponential in the number of parties, n, thus limiting the number of parties to be at most logarithmic in the security parameter λ. There are other promising approaches in the works for threshold RSA cryptosystems. Canetti et al. [21] proposed a methodology for transforming a selectively-secure threshold scheme into an adaptively-secure one, where the protocol needs to be modified to carefully erase secrets and to use simple zero-knowledge proofs, the adversary is rewinded in the proof which also incurs a security loss although not as large as with the complexity-leveraging approach. A follow-up work of Almansa et al. [4] simplified this result for RSA, but at the cost of the secret's re-sharing after every round, with an emergent benefit of making the scheme proactively secure. In a proactively secure scheme the adversary can corrupt at most t players in a time period determined by the protocol. Since the set of corrupted parties changes, each party can become corrupt at some point (i.e. leak its secrets), but if the party recovers from a compromise then a subsequent secret's re-sharing will enable the party to be honest again.

Given the above discussion, we now formally define the security model that we consider for our core protocol: unpredictable, selective and adaptive.

Definition 2 (Selective-secure Unpredictability)**.** *We say that an HGDRB is* Selective-secure Unpredictable *if the following adversary has negligible advantage:*

1. *The Challenger runs* $Setup(\lambda, n, t)$ *and outputs* pk *to the Adversary.*
2. *The Adversary selects a time epoch* $S < poly(\lambda)$ *and a set of parties* $\gamma = \{P_1, P_2, \cdots, P_{t-1}\}$ *to corrupt.*
3. *The Challenger sends the transcript of the protocol till time epoch* S *to the Adversary, as well as the secret shares for parties in* γ*. This includes the outputs of Gen and those of Eval for all* $i \in [1, n]$ *and* $T \leq S$*.*
4. *The Adversary outputs a quantity* x'*.*
5. *The Adversary wins if* $Verify(pk, x_S, x')$ *passes.*

Although the above definition does not explicitly let the adversary modify public share and beacon values, this is still equivalent to an active adversary, as the self-verification of beacon and share values ensure that only the unique correct values would not be rejected.

5 Proof of Security

In this section, we show that the STROBE protocol satisfies Selective-secure Unpredictability under the RSA assumption.

Definition 3 (RSA Assumption)**.** *The Challenger samples RSA number $N = pq$ and picks a quantity s co-prime to $\phi(N)$. Then it randomly samples $z \leftarrow [1, N]$ and sends (N, s, z) to the Adversary. The Adversary outputs y. The RSA assumption states that the probability of $y^s = z \pmod N$ is negligible.*

Theorem 5.1 (Security)**.** *The STROBE protocol is Selective-secure Unpredictable under the RSA Assumption.*

Proof. Let (N, s, z) be an RSA challenge for a prime $s > n$.

Setup: The Challenger outputs $pk = (N, s)$. Suppose the Adversary picks an epoch S and corrupts parties in the set $\gamma = \{P_1, \cdots, P_{t-1}\}$. Sample $b_{P_1}, \cdots, b_{P_{t-1}} \leftarrow [1, N]$. Send Adversary shares $sk_i = b_i$, for all $i \in \gamma$.

Eval: Consider an implicit polynomial $f(X)$, such that $f(0) = v$, where $v = (n!s)^{-1} \pmod{p'q'}$, and $f(i) = b_i$, for all $i \in \gamma$:

$$f(X) = vL_0(X) + \sum_{i \in \gamma} b_i L_i(X).$$

Here $L_i(X)$ are Lagrange basis polynomials of degree $t - 1$ each:

$$L_i(X) = \prod_{j \in \{\gamma \cup \{0\}\}, j \neq i} \frac{X - j}{i - j}.$$

The polynomials satisfy the following property: for $\forall i, j \in (\gamma \cup \{0\}) : L_i(i) = 1$, and $L_i(j) = 0$ for $i \neq j$. Note that it is possible to evaluate $z^{n!L_i(j)}$ for any $i, j \in [0, n]$, since the $(n!)$ factor eliminates the denominators of interpolation polynomials, making it possible to evaluate the exponentiation.

Set $x_0 = z^{4(n!)^2 s^S}$ and $x_{T,j} = z^{4f(j)(n!)^2 s^{S-T+1}}$. Now, for $j \in \gamma$, we can compute $x_{T,j}$ explicitly based on $f(j) = b_j$. We now show that for $j \notin \gamma$, we can compute $x_{T,j}$ without explicitly computing $f(j)$. Observe that $4v = 4(n!s)^{-1} \pmod{\phi(N)}$, therefore $z^{v4n!s} = z^4$ hence $x_{T,j}$ can be explicitly constructed as follows:

$$x_{T,j} = z^{4f(j)(n!)^2 s^{S-T+1}} = z^{(f(j)4n!s)\cdot(n!)s^{S-T}} = z^{(4L_0(j) + \sum_{i \in \gamma} b_i \cdot (4n!s) \cdot L_i(j)) \cdot (n!) s^{S-T}}$$

Note that the last $(n!)$ factor is essential to make sure we can clear the denominator of $L_0(j)$ and thus avoid any divisions in the exponent. Send x_0 and the $x_{T,j}$'s for all $T \in [1, S]$ and $j \in [1, n]$ to the adversary.

RSA response: Let's suppose the adversary comes up with x' as the next epoch candidate. Given the self-certification checks upto time epoch T, we inductively have $x_k = x_0^{s^{-k}} =$

$z^{4(n!)^2 s^{S-k}}$, for $k \in [0, S]$. If the adversary response x' passes verification, then we should have $x'^s = x_S = z^{4(n!)^2}$, Let $w = (4(n!)^2)^{-1} \pmod{s}$ and let $w(4(n!)^2) = 1 + ks$ for some computable integer k. Then $x'^{sw} = z^{4(n!)^2 w} = z^{1+ks}$. Therefore, $z = (x'^w z^{-k})^s$. Hence $x'^w z^{-k}$ will be a winning response to the RSA challenge.

Distributions: Finally, we observe that as s^S is invertible module $\phi(N)$ and sampling uniformly from $[1, N]$ and $[1, \phi(N)]$ are statistically indistinguishable, the distribution of x_0 in the STROBE construction and this proof are statistically indistinguishable. Matching these distributions is the technical reason behind the extra $4(n!)^2$ factor in the *Gen* algorithm. □

6 Extensions

6.1 Dynamic beacon committees.

STROBE can also easily handle dynamically changing the participants executing the beacon protocol. For example, this could involve rotating to a newly chosen committee (even of a different size) of parties after some fixed number of epochs. The new committee will then continue to generate future outputs of the beacon. In fact, the new committee can also be chosen based on the output of the distributed beacon. Dynamic participation can be accommodated by using the old committee to reshare the secret key to the new committee through existing literature on dynamic proactive secret sharing [8, 45]. Note however, that such protocols need to make the assumption that honest parties from previous committees delete their shares so that an adversary cannot recover the secret key even by corrupting more than a threshold number of parties from a previous committee. The above scenario is also referred to as a "long range" attack (c.f., [26]).

6.2 Succinct proofs of beacon validation.

In earlier sections we observed that it is possible to check the beacon value x_T at epoch T against the seed value x_0 by checking $x_0 = x_T^{s^T}$. However, this takes sequential time T. One way to speed up verification is to exploit the RSA repeated powering structure of this check and use existing techniques like [47, 55] to add a proof in addition to the beacon value.

In particular, we can just apply Wesolowski's [55] proof technique in reverse and produce the proof $\pi = x_T^{\lfloor s^T/\ell \rfloor}$, where ℓ is the result (a prime number) of a random oracle H applied to (x_0, x_T, T). The verifier computes ℓ and r as the remainder on dividing s^T by ℓ and then checks $x_0 = \pi^\ell x_T^r$. However, a drawback of this method is that producing the proof takes about T time as well, which may not be ideal to do every epoch.

A better approach is to use a continuous VDF [33]. In a continuous VDF, it is efficient to publish and use intermediate proofs at every time epoch. The Ephraim et al [33] continuous VDF makes use of the recursive structure of Pietzak's VDF [47]. The high level idea is that they checkpoint a logarithmic number of past values and keep recursively merging an appropriate number of them in order to prevent growing the proof size. We can also similarly

checkpoint and merge based on a similar recursion, while reversing the order: $x_0 = x_u^{s^{T/2}}$ and $x_u = x_T^{s^{T/2}}$, where $u = T/2$ is the midpoint.

We can use a continuous VDF at defined intervals as well, instead of every epoch. A verifier can sequentially compute the expected value until the last such interval and then just check the VDF proof.

7 Applications

In this section, we discuss applications of random beacons. In particular, we argue how the novel history generation feature of STROBE can enable attractive technical advancements in many scenarios.

7.1 Blockchain-based gambling and lotteries

Lotteries and gambling smart contract solutions are gaining in popularity, and a portion of them advertise transparency, unbiased randomness generation (uncontrollable from participants) and consumer privacy. It is well known that the original success of Bitcoin was partly due to gambling activities, especially via the Satoshi Dice which operated since 2012 and dominated the bitcoin transactions in its first years of operation [36]. There is a growing number of blockchain gambling contracts, and as of September 2021, according to statistics provided in [29], there exist at least 151 lottery, 129 casino, 70 poker and generally 600+ gambling smart contracts in Ethereum alone. Moreover, lotteries have also been proposed as an alternative reward scheme for miners by randomly recirculating lost coins and collecting gold dust [37]. Furthermore, leader election in BFT consensus schemes, can also be implemented via a lottery [11, 38].

The most common approach in proof of work ledgers is to use the block-hash as a seed to pick winning numbers, due to its unpredictability characteristics. However, this value can be biased [16] and VDFs on top of the original seed have been proposed [17], which however introduce significant delays. Various other attacks have been reported which are related to modulus bias or lack of a proof of origin mechanism for the seed [25].

We should note that for high-throughput games, like continuous poker shuffling, VDF delays might not be tolerable for UX reasons. On the other hand, updating the blockchain state too often can be expensive, thus ideally a dynamic balance between the PRNG output frequency and gas cost should be configurable. One of the main advantages of STROBE is that the latest beacon can be automatically used to "verifiably" derive all of the previous random numbers down to the original genesis beacon. This property can be utilized by smart contract developers to minimize cost by skipping beacon epochs when required, but lotteries can still continuously run for every epoch. It also works as a DoS defence, especially when the beacon is provided by an external oracle service to the blockchain; if there is significant delay on updates, the latest beacon suffices to execute all of the pending lottery games. This is equivalent to supporting {smart contract}-based "light clients" of a beacon generation committee.

Finally, lotteries could be executed in optimistic mode [1], by blindly accepting any submitted beacon without correctness checking. Then, in a reasonable time-frame and before processing winner's payout, the beacon's validity can be challenged by providing a fraud proof, which is nothing more than submitting a flag which triggers history validation.

7.2 Deeper Blockchain Integration

The design described in Section 3 relies on authorities on-the-side for issuing credentials. In this section, we present designs that incorporate STROBE authorities within the infrastructure of a number of semi-permissioned Blockchains. This enables the generation of randomness as a side effect of the normal system operations, taking no additional dependency on extra authorities. It remains an open problem how to embed STROBE into permissionless systems, based on proof of work or stake. These systems have a highly dynamic set of nodes maintaining the state of their blockchains, which cannot readily be mapped into STROBE's authorities.

Integration of STROBE into permissioned (BFT-based) blockchain platforms is straightforward. In Hyperledger Fabric Fabric [19] for instance, contracts run on private sets of computation nodes and use the Fabric protocols for cross-contract calls. In this setting, STROBE authorities can coincide with the Fabric smart contract authorities. Upon a contract setup, they perform a setup and key distribution, and then start generating randomness when authorized by the contract. For generating randomness, the only secrets maintained are the private STROBE authorities keys; all other operations of the contract can be logged and publicly verified. The threshold trust assumption—namely that of integrity and availability— is guaranteed under the corruption of a subset of authorities is preserved, and prevents forgeries by a single corrupted node.

We can also naturally embed STROBE into sharded scalable blockchains, as exemplified by Omniledger [43] (which supports digital tokens), and Chainspace [2] (which supports general smart contracts) or as part of a heterogeneous shard [5]. In these systems, transactions are distributed and executed on 'shards' of authorities, whose membership and public keys are known. STROBE authorities can naturally coincide with the nodes within a shard—a special transaction type in Omniledger, or a special object in Chainspace, can signal to them that generating randomness. The authorities, then issue the partial randomness share necessary to reconstruct the random value, and attach it to the transaction they are processing anyway. Users (or anyone else) can aggregate, and generate the random value by parsing the transactions on-chain.

Incentives for Randomness Generation. One open question is on managing the incentives of releasing shares for the randomness beacon. On a first sight the fault-tolerance of STROBE makes the problem a public goods game, since only a threshold needs to participate. One way to break this has been proposed in prior work [6, 42] where the shares are published on chain and rewards are given only to "useful" shares (*i.e.* the first t that make it on-chain). This breaks the public goods game and makes it a race between the authorities who are eager to release their share and get rewarded.

7.3 Beacon Streams

There exist applications requiring constant high-throughput of beacons, especially in the online gaming sector. Most games require a combination of "skills" and "luck", and as video gaming has become a market where professional players participate in tournaments with real prizes, a fair beacon stream would enable new types of transparent gaming features. Due to potential internet speed issues, server overloading and other factors, reliability might be at stake [27] and thus, some game providers prefer UDP connections to offer greater flexibility by executing packets out of order or discarding non important ones [56]. STROBE's history generation feature fits really well in streaming designs, and allows client software to generate game states by computing every missing beacon. Note that in these applications, VDF-based beacons are not good candidates due to the intrinsic delay, while the proof of stake nature of STROBE offers fairness guarantees by not relying in the event organizer's or software publisher's honesty. Along the same lines, STROBE has an advantage in low or expensive bandwidth locations (i.e., remote IoT devices) by allowing reading a beacon infrequently and generating past randomness internally.

References

[1] Adler, J., Quintyne-Collins, M.: Building scalable decentralized payment systems. arXiv preprint arXiv:1904.06441 (2019)

[2] Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., Danezis, G.: Chainspace: A Sharded Smart Contracts Platform. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2018)

[3] Algesheimer, J., Camenisch, J., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In: Annual International Cryptology Conference. pp. 417–432. Springer (2002)

[4] Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold rsa with adaptive and proactive security. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 593–611. Springer (2006)

[5] Alp, E.C., Kokoris-Kogias, E., Fragkouli, G., Ford, B.: Rethinking general-purpose decentralized computing. In: Proceedings of the Workshop on Hot Topics in Operating Systems. p. 105–112. HotOS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3317550.3321448, https://doi.org/10.1145/3317550.3321448

[6] Avarikioti, Z., Kogias, E., Wattenhofer, R., Zindros, D.: Brick: Asynchronous incentive-compatible payment channels. In: International Conference on Financial Cryptography and Data Security (2021)

[7] Beaver, D., So, N.: Global, unpredictable bit generation without broadcast. In: IACR Eurocrypt 1993. Lecture Notes in Computer Science, vol. 765, pp. 424–434. Springer (1993). https://doi.org/10.1007/3-540-48285-7_36

[8] Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a blockchain keep a secret? In: TCC. pp. 260–290 (2020)

[9] Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. **15**, 364–383 (1986)

[10] Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Annual international cryptology conference. pp. 757–788. Springer (2018)

[11] Boneh, D., Eskandarian, S., Hanzlik, L., Greco, N.: Single secret leader election. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 12–24 (2020)

[12] Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: CRYPTO. pp. 425–439 (1997)

[13] Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. Journal of the ACM **48**(4), 702–722 (2001)

[14] Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: International conference on the theory and applications of cryptographic techniques. pp. 416–432. Springer (2003)

[15] Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)

[16] Bonneau, J., Clark, J., Goldfeder, S.: On bitcoin as a public randomness source. IACR Cryptol. ePrint Arch. **2015**, 1015 (2015)

[17] Bünz, B., Goldfeder, S., Bonneau, J.: Proofs-of-delay and randomness beacons in ethereum. IEEE Security and Privacy on the blockchain (IEEE S&B) (2017)

[18] Cachin, C.: An asynchronous protocol for distributed computation of rsa inverses and its applications. In: Proceedings of the twenty-second annual symposium on Principles of distributed computing. pp. 153–162 (2003)

[19] Cachin, C.: Architecture of the hyperledger blockchain fabric. In: Distributed Cryptocurrencies and Consensus Ledgers (2016)

[20] Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. Journal of Cryptology **18**(3), 219–246 (2005)

[21] Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: Annual International Cryptology Conference. pp. 98–116. Springer (1999)

[22] Cascudo, I., David, B.: Scrape: Scalable randomness attested by public entities. In: International Conference on Applied Cryptography and Network Security. pp. 537–556. Springer (2017)

[23] Cascudo, I., David, B.: Albatross: publicly attestable batched randomness based on secret sharing. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 311–341. Springer (2020)

[24] Catalano, D., Gennaro, R., Halevi, S.: Computing inverses over a shared secret modulus. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 190–206. Springer (2000)

[25] Chalkias, K., Kichidis, A.: Why firelotto's blockchain-based random engine is not fair? R3 Corda Seminars (2018)

[26] Chatzigiannis, P., Chalkias, K.: Proof of assets in the diem blockchain. In: International Conference on Applied Cryptography and Network Security. pp. 27–41. Springer (2021)

[27] Consalvo, M.: Cheating: Gaining advantage in videogames. Mit Press (2009)

[28] Damgård, I., Koprowski, M.: Practical threshold rsa signatures without a trusted dealer. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 152–165. Springer (2001)

[29] Dapp.com: List of gambling ethereum smart contracts (2021), https://www.dapp.com/search_product?keyword=gambling

[30] Dolev, D., Strong, H.: Polynomial algorithms for multiple processor agreement. SIAM J Computing **12**(4), 656–666 (1982). https://doi.org/https://doi.org/10.1137/0212045

[31] Drake, J.: Minimal vdf randomness beacon, https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566

[32] Drand a distributed randomness beacon daemon, https://drand.love/

[33] Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 125–154. Springer (2020)

[34] Feldman, P., Micali, S.: Byzantine agreement in constant expected time. In: IEEE Symposium on Foundations of Computer Science. pp. 267–276 (1997)

[35] Frankel, Y., MacKenzie, P.D., Yung, M.: Robust efficient distributed rsa-key generation. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 663–672 (1998)

[36] Gainsbury, S.M., Blaszczynski, A.: How blockchain and cryptocurrency technology could revolutionize online gambling. Gaming Law Review **21**(7), 482–492 (2017)

[37] Gjermundrød, H., Chalkias, K., Dionysiou, I.: Going beyond the coinbase transaction fee: Alternative reward schemes for miners in blockchain systems. In: Proceedings of the 20th Pan-Hellenic Conference on Informatics. pp. 1–4 (2016)

[38] Gouget, A., Patarin, J., Toulemonde, A.: Unpredictability properties in algorand consensus protocol. In: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–3. IEEE (2021)

[39] Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system. arXiv preprint arXiv:1805.04548 (2018)

[40] Kate, A., Goldberg, I.: Distributed key generation for the internet. In: 2009 29th IEEE International Conference on Distributed Computing Systems. pp. 119–128. IEEE (2009)

[41] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual International Cryptology Conference. pp. 357–388. Springer (2017)

[42] Kokoris-Kogias, E., Alp, E.C., Gasser, L., Jovanovic, P., Syta, E., Ford, B.: Calypso: Private data management for decentralized ledgers. Proceedings of the VLDB Endowment **14**(4), 586–599 (2020)

[43] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger via sharding. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 583–598. IEEE (2018)

[44] Kokoris Kogias, E., Malkhi, D., Spiegelman, A.: Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1751–1767 (2020)

[45] Maram, S.K.D., Zhang, F., Wang, L., Low, A., Zhang, Y., Juels, A., Song, D.: Churp: Dynamic-committee proactive secret sharing. In: CCS. pp. 2369–2386 (2019)

[46] Miller, G.L.: Riemann's hypothesis and tests for primality. Journal of Computer and System Sciences **13**(3), 300–317 (1976)

[47] Pietrzak, K.: Simple verifiable delay functions. In: 10th innovations in theoretical computer science conference (itcs 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)

[48] Rabin, M.O.: Probabilistic algorithm for testing primality. Journal of Number Theory **12**(1), 128–138 (1980)

[49] Rabin, M.O.: Transaction protection by beacons. Journal of Computer and System Sciences **27**(2), 256–267 (1983)

[50] Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., Weippl, E.: Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness. In: NDSS 2022 (2021)

[51] Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: Hydrand: Efficient continuous distributed randomness. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 73–89. IEEE (2020)

[52] Shoup, V.: Practical threshold signatures. In: EUROCRYPT 2000. pp. 207–220 (2000)

[53] Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 444–460. Ieee (2017)

[54] Veedo is a stark-based verifiable delay function (vdf) service, https://github.com/starkware-libs/veedo

[55] Wesolowski, B.: Efficient verifiable delay functions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 379–407. Springer (2019)

[56] Xue, Z., Wu, D., He, J., Hei, X., Liu, Y.: Playing high-end video games in the cloud: A measurement study. IEEE Transactions on Circuits and Systems for Video Technology **25**(12), 2013–2025 (2014)