zkBeacon: Proven Randomness Beacon Based on Zero-Knowledge Verifiable Computation

Thomas Lavaur, Jérôme Lacan

zkBeacon: Proven Randomness Beacon Based on Zero-Knowledge Verifiable Computation

Extended version of paper accepted at SECRYPT 2022

Lavaur Thomas[1,2], Lacan Jérôme[1]
[1]*ISAE-Supaero, Université de Toulouse, France*
[2]*Paul Sabatier, Université de Toulouse, France*
{*thomas.lavaur, jerome.lacan*}*@isae-supaero.fr*

Keywords: Randomness Beacon - Random Number Generation - zk-SNARK - zk-STARK - Verifiable Computation.

Abstract: The generation of random numbers by a trusted third-party is essential to many cryptographic protocols. Recently, the NIST proposed the standardization of randomness beacons, which are hash-based chains of pulses. Each pulse contains a random number and is generated at regular time intervals. However, if the owner of the beacon generator is untrusted, several attacks allow the manipulation of the provided random numbers. In this paper, we firstly suggest protecting the first hash functions of the NIST scheme by adding a verifiable argument of knowledge. More precisely, we propose furnishing a zk-SNARK or a zk-STARK with the hash to make the system more transparent and resistant to randomness manipulation. Secondly, we propose a verifiable computation-based interactive protocol to allow a client, with the help of the beacon, to generate proven randomness. Finally, we show that connecting this system to a blockchain could have several benefits. We provide a security analysis with a model allowing a malicious beacon generator. We prove that our first application improves the resilience of the system against randomness manipulation attacks and that the interactive protocol rules out timing attacks for the client and ensures the non-predictability of the random numbers. Finally, we evaluated the computation cost with zk-SNARKs.

1 Introduction

The concept of the randomness beacon was introduced in (Rabin, 1983). It can be viewed as a service that generates a new beacon at regular time intervals. This beacon is called a pulse and mainly contains a fresh random number. While Rabin used it to build specific cryptographic applications such as contract signing, several other applications have been proposed such as voting protocols (Moran and Naor, 2010) or traceable signatures (Kiayias et al.,). In order to make compatible potential concurrent randomness beacons, the US National Institute of Standards and Technology (NIST) suggests a standard draft (Kelsey et al., 2019) enabling the emission of structured random numbers starting from raw randomness. It has also proposed its own beacon server (Nis, 2020), as well as some projects in Chile (UChile, 2020) and Brazil (Bra, 2020). Basically, the main issues that must be solved by these systems are the quality of the randomness of the outputs, their freshness and their immutability. These issues are detailed in (Kelsey et al., 2019) and some countermeasures are proposed. However, the system is not flawless and attacks exist, especially conceivable from the operator of the beacon.

The first objective of our paper is to make the system more transparent and resistant to various attacks while maintaining its interesting structure and properties i.e., a chain of pulses using external raw entropy. Secondly, we propose an interactive protocol enabling the collaboration of the randomness beacon and another party in order to generate a new proven random number from their raw data. Both proposals make use of verifiable computation (VC), which allows proving that a given computation has been correctly performed with public or private inputs. Used in the context of randomness beacons, they prove that specific steps of the system were effectively done in the process of the generation of the given outputs.

This paper is organized as follows. The next section presents the state of the art concerning various aspects related to randomness beacons. In Section 3, we describe the NIST scheme and its weaknesses. Then, we detail our three main propositions: the addition of a zero-knowledge VC scheme to prove the proper generation of the pulses, an interactive protocol to ensure the freshness of the generated data, and the advantages of connecting the system to a blockchain. In the fourth section, we present a security analysis for our proposition. To this end, we analyze the potential attacks and examine how our systems respond to them. In Section 5, we discuss the choice of certain mechanisms (zk-SNARK or zk-STARK, hash function used, security level) and provide a performance evaluation of an implementation of proof generation for our systems. The last section concludes.

2 STATE OF THE ART

This section introduces the main concepts used in this paper. First, we present the problem of reliable random number generation and explain the principle of the randomness beacon. The last part describes the zero-knowledge VC schemes that will be used in this paper to improve the trustworthiness of these randomness beacons.

2.1 Random Number Generation

Basically, a sequence of numbers is considered random if and only if it verifies multiple properties such as uniform distribution and unpredictability. These properties are needed in different fields in order to obtain good performance, for example in simulation, decision-making or in mathematical analysis with the Monte-Carlo method. However, these properties are necessary and essential to security in computer science and cryptography. Random numbers are used in many critical protocols such as cryptographic key generation, authentication challenges, blinding and even side-channel countermeasures and others. They are crucial for everything related to internet security. A lack of randomness can lead to vulnerabilities in all of the protocols relying on the weak random number.

The generation of random sequences can be achieved via different techniques using a non-deterministic method based on randomness classically produced by natural phenomena such as quantum noise (Jacak et al., 2021) or using a pseudo-random number generator (PRNG) and a seed (e.g. Linear-Feedback Shift Register). Random numbers generated by a non-deterministic generator are often consequently released by a trusted third party due to the expensive equipment required.

Introduced by Micali, Rabin and Vadhan in 1999 (Micali et al., 1999), verifiable random functions (VRF) aim to generate proven randomness. Based on asymmetric cryptography, the inputs of this pseudo-random function are a public value (the seed) and a secret generation key. The output is composed of a proof and the result. With a given public key and the two parts of the output, anyone can verify the proper execution of this function and thus of the random number generation.

2.2 Public Randomness Beacons

When a system needs randomness generated by a third-party, it is possible to use the services of a public generator called a randomness beacon. For example, this is the case for blockchains and some cryptographic protocols. This concept of randomness beacons was introduced in (Rabin, 1983) which describes the generation of random numbers by a satellite or a network node at regular time intervals. These regular beacons are used to build security protocols such as secure contract signing or confidential disclosure.

Recently, the NIST has developed a standard describing an architecture that produces a chain of random beacons. This standard explains how to convert raw data from a source of entropy into a signed and timestamped random number with the help of hash functions. This proposition has already been deployed and several organizations provide random numbers based on it, such as the NIST itself (Nis, 2020) or public organizations from Chile (UChile, 2020) or Brazil (Bra, 2020). In addition to the description of the system, (Nis, 2020) provides a security analysis by identifying potential attacks and proposing certain countermeasures.) Efficient randomness beacons have been extensively studied in the literature. For example, (Dharanikot et al.,) designed a randomness beacon combining distributed randomness, by allowing users to join at any time, and cryptographic delay functions. Other examples are (Bonneau et al., 2015) or (Bunz et al., 2017) which extract randomness from blockchains.

2.3 Verifiable Computation and Zero-Knowledge Proofs

Introduced in (Goldwasser et al., 1989), a verifiable computation is a cryptographic protocol allowing a prover to provide a proof with the result of an NP computation which attests to the validity of the computation. Given the input, the output and the related

proof, the verifier can validate the output of a computation as being the result of a public program without needing to run it again. This proof can be interactive or non-interactive.

There are a multitude of verifiable computation protocols, but three main categories have emerged and represent the vast majority of the systems used: Succinct Non-interactive ARgument of Knowledge (SNARK), Scalable and Transparent ARgument of Knowledge (STARK) and universal SNARK. We will not go into the history of the conception of these proofs but we will detail the functioning, the advantages and the disadvantages of these two families.

With the addition of zero-knowledge, some of the inputs can be kept secret by hiding them and the intermediate steps of the computation (this secret part is call the witness). This allows the prover to justify the result of a computation without revealing specific inputs. If the verifier cannot retrieve the private input from the output (when the program cannot be inverted), they will not learn anything from the proof except the result of that calculation and the fact that the public program was computed correctly. With this addition, the system is called zk-SNARK or zk-STARK.

For most SNARKs and STARKs the program needs to be transformed into a polynomial representation. This step is often called arithmetization.

2.3.1 SNARKs and Universal SNARKs

SNARKs were introduced in practice in 2013 (Parno et al., 2013), the terms used to name this family of protocols are defined as follows:

Succint: Communications are tiny compared to the length of the computation. The verifier complexity is in $O(\log n)$, with n being the size of the circuit (the program).

Non-interactive: No or little interaction (no rounds of interactions), often a setup phase and a small message from the prover to verifier.

Argument: The verifier is only protected against a computationally limited prover. This is called the computational soundness, as opposed to perfect soundness, which describes a proof (we will often use and read proof instead of argument).

of Knowledge: It is not possible for a prover to construct a proof/an argument without knowing the witness. Formally, for any prover who can produce a valid argument, there is an extractor able to extract a witness from the statement.

SNARKs often rely on pairing, q-Knowledge of Exponent (q-KEA) and q-power Diffie-Hellman which are generalizations of Diffie-Hellman decisional and computational problems.

The security of a zk-SNARK protocol relies on a trusted setup. In order to limit vulnerabilities, we propose using the Powers of Tau ceremony (Bowe et al., 2017) to decentralize the trusted setup. The Powers of Tau ceremony allows for the decentralization of the first step of the trusted setup which is common to all specific zk-SNARK circuits. At the end of this protocol, the only way to compromise the system and forge a fake proof is if everyone who participated in the protocol is corrupted or malicious. By attracting many diverse and reputable participants, it becomes unrealistic that all of them would be compromised.

In order to prove an NP program, it must be reduced to a Rank 1 Constraint System (R1CS). A R1CS is composed of polynomials extracted from multiplication and addition gates; their number, denoted by n, impacts prover complexity. This is the arithmetization step.

The trusted setup can either be circuit specific for SNARKs or up to a certain amount of constraints for universal SNARKs.

The two most used SNARK systems are Groth16 (Groth, 2016) and PLONK (Gabizon et al., 2019). In the Groth16 system, the proof size is exactly two elements of $\mathbb{G}_1$ and one of $\mathbb{G}_2$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are the bilinear groups used for pairing. This implies that the proof size does not depend on the program's complexity and is shorter than 200 bytes on classical elliptic curves. That is why Groth16 is often used. PLONK is a universal SNARK protocol with great performance.

2.3.2 STARKs

STARKs are more recent, introduced in 2018 (Ben-Sasson et al., 2018). The terms common to both SNARKs and STARKs have the same definitions. The others are defined as:

Scalable: The verifier verifies in $O(\log^{O(1)} n)$ and the proof is computed in $O(n \log^{O(1)} n)$, with n being the size of the circuit (the program).

Transparent: There is no trusted setup.

STARKs rely only on the collision resistance of a hash function in order to build merkle trees, and the properties of error correction schemes. In most of the schemes, STARKs are based on Reed-Solomon codes but some research has explored using other codes (Bordage and Nardi, 2020). When a quantum collision-resistant hash function can be used, STARKs are quantum resistant.

The arithmetization phase of STARKs is the reduction of the computation to an Algebraic Intermediate Representation (AIR). An AIR is a set of polynomials extracted from transition steps; their number, denoted by n, impacts prover complexity.

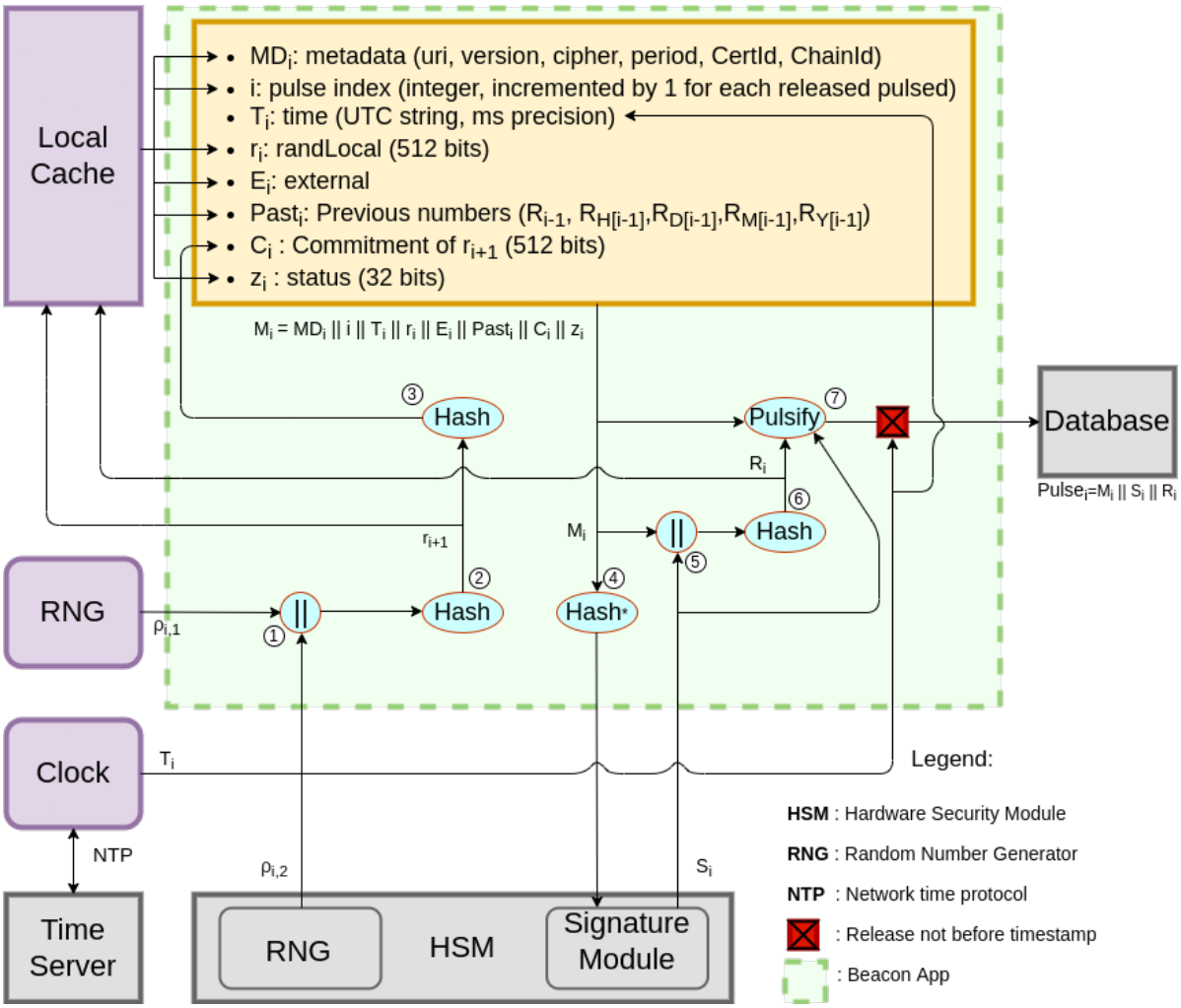

Figure 1: The NIST random number generation model.

3 A TRUSTWORTHY BEACON

3.1 NIST Randomness Beacon

The NIST generation scheme aims to furnish random numbers at regular time intervals (Kelsey et al., 2019). These numbers are linked through a commitment scheme and form a chain. Each number is sent along with information about it and its construction, all of this is structured and formatted into a pulse. According to the NIST proposition, a pulse contains data such as URI, chainId, time, signature, the freshly-generated random number, a pulse index, etc. The pulse contains all the information needed for the generation of the random number. Everything is constructed and calculated by the beacon app, represented in the green dotted rectangle in Fig. 1, with the help of a signature module and a hardware security module (HSM) which furnishes raw randomness.

In order to construct a new random number at index i, 1024 bits of full entropy are needed, equitably furnished by 2 random number generators, one from the beacon app that constructs the number and one from the HSM ①. These two input values are named $\rho_{i,1}$ and $\rho_{i,2}$. A third generator can be used; the NIST suggests a quantum generator. These two (or three) values are concatenated and hashed. The use of SHA512 to hash $\rho_{i,1}||\rho_{i,2}$ has been suggested. The entropy seed called randLocal for the next random number is then obtained: $r_{i+1} = Hash(\rho_{i,1}||\rho_{i,2})$ ②. The link between two consecutive pulses is ensured by this value. It is stored in the local cache and committed through the pulse by computing $C_i = Hash(r_{i+1})$ ③.

At this point, all of the information needed to construct the message M_i, which structures everything, has been collected. M_i is the concatenation of everything: the metadata MD_i, the pulse index i, the

time T_i, the randLocal value r_i calculated during the generation of the previous number, external information E_i, the past random numbers to link pulses $Past_i = R_{i-1}, R_{H[i-1]}, R_{D[i-1]}, R_{M[i-1]}, R_{Y[i-1]}$ (where $H[i-1], D[i-1], M[i-1]$ and $Y[i-1]$ stand for the first random number of the previous hour, day, month and year respectively), the commitment C_i and a status z_i.

This message M_i is hashed to be signed by the HSM ④. The HSM answers with the signature S_i related to M_i ⑤. The final random number R_i is equal to $Hash(M_i||S_i)$ ⑥ and the pulse released by the beacon app at time T_i is $P_i = M_i||R_i||S_i$ ⑦. This whole process is summarized in Fig. 1.

All these steps and recommendations are explained in the NIST reference (Kelsey et al., 2019). For our proposition, we will add cryptographic tools to this scheme to increase transparency and secure vulnerabilities, allowing users to be more confident in it. Indeed, this scheme is vulnerable and users should not be convinced that what they receive is fresh and has not been altered.

Put simply, our first idea was to apply VRFs to the entropy seeds $\rho_{i,*}$. However, the verification step of the VRF needs the inputs and thus, would force the system to reveal the values of the seeds, which significantly change the principle of the beacon generator. Moreover, different types of operations need to be proved. These constraints led us to use more general proofs, like VC arguments.

3.2 Zero-Knowledge VC to Ensure the Proper Generation of the Pulse

3.2.1 Generation of Pulses

We propose proving the first hash functions, which transform the raw randomness of different random number generators into a seed used to generate the final random number and a commitment, with the help of VCs. In fact, the NIST model is already a bit transparent in this regard but there is no guarantee about the seed's r_i provenance. For a user, when they receive the pulse, they can already verify all the information except for the r_i value by redoing the whole computation. However, several attacks can arise from the user's inability to check the origin of this value. The hash function can be replaced in the protocol and nobody would notice. With the addition of a VC argument, we aim to patch this vulnerability and make the generation reliable by enabling the verification of r_i while keeping the inputs secret. By doing this, we ensure that C_i is the output of a public program: two hash functions.

For our scheme, we propose proving the two first hash functions: the public program is the hash functions themselves ①. We propose furnishing the time T_i of the pulse as a public input and $\rho_{i,1}, \rho_{i,2}$ as the private input.

Finally, we follow one of the NIST suggestions, not present in the previous section, which consists of adding a XOR between the output of the first hash and the freshly-generated random number R_i to obtain the new randLocal value: $r_{i+1} = Hash(\rho_{i,1}||\rho_{i,2}) \oplus R_i$ ②. All these changes are presented in Fig. 2. For the security aspect, we decided to use a deterministic signature instead of an HSM due to the fact that its security is based entirely on the trustworthiness of its creator. Information inside the message M_i changes slightly from the original proposition so as to adapt to our scheme. We removed the external value which is not useful in our case and added the proof of the commitment C_i that resulted from a VC scheme.

3.2.2 Verification

When a user receives a new pulse, they can verify the proper generation of R_i. To do that, they check that index, time, previous numbers and the status are consistent with the current chain ③. After that, they verify that the commitment in the previous pulse is equal to the hash function applied to the new r_i received XORed with R_{i-1} ④. The pulse is authenticated using the public key of the beacon and the signature S_i: the user verifies that the signature is from the beacon and related to $Hash(M_i)$ by hashing the message M_i ⑤. With the help of M_i a user can reconstruct the public input consisting of T_i, the output C_i and the proof p_i. Since they have the public program, the public input, the output and the proof (and a public verifier key, if needed), the user can check the validity of the generation of C_i ⑥. Finally, if $R_i = Hash(M_i||S_i)$ the new random number was correctly generated by the beacon app which holds signature keys (and the prover key, if needed).

All these verifications only require redoing hash functions, verifying a signature and an argument from a VC protocol, which implies quick verification on the user-end.

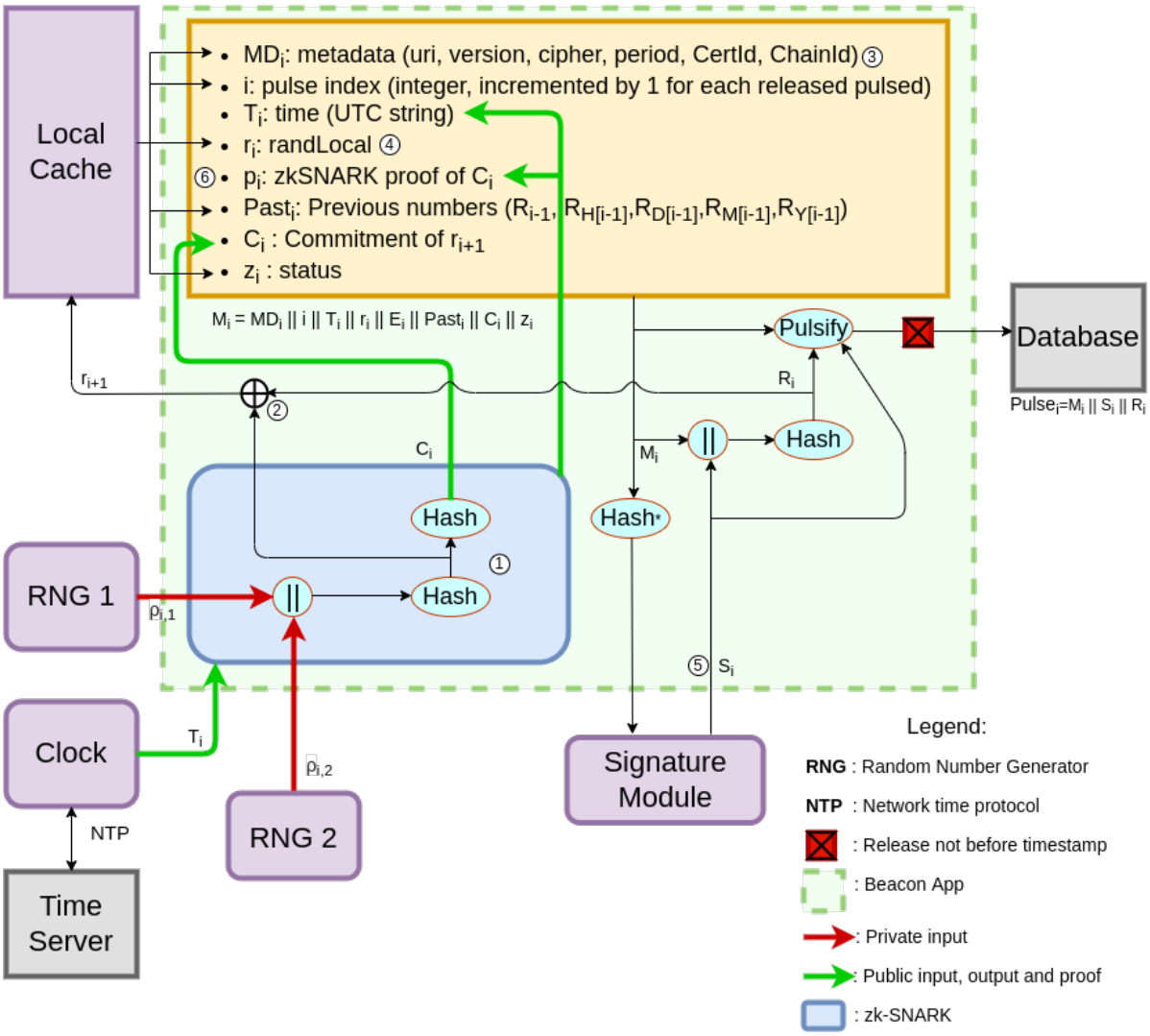

Figure 2: The zkBeacon random number generation model.

3.3 Interactive Protocol Based on Zero-Knowledge VC to Ensure the Freshness of the Pulses

Here, we exhibit our interactive protocol, allowing a user to combine their entropy with that of the randomness beacon. This protocol is completely independent of the beacon chain based on VC. Our second proposition can even be applied to a classical beacon, such as the one from the NIST.

When a client receives a new pulse from the randomness beacon, even if all the computation was done correctly, they cannot be sure that the entropy used to generate the random number inside $\rho_{i,1}$ and $\rho_{i,2}$ is sufficient and unmanipulated. Moreover, the pulse is supposed to be fresh but nothing guarantees that the pulse was not created a long time ago.

In order to counter these possibilities, we propose an interactive protocol where a user asks the beacon to create an *orphan* pulse with three inputs: the same $\rho_{i,1}$ and $\rho_{i,2}$ used in the last pulse and a new value $\rho_{i,3}$ that they provide. This enables multiple possibilities to counter time attacks and bias on random numbers from a malicious owner or adversary. This will be discussed in Section 4.

In addition to classical fields (metadata, status, index), an *orphan* pulse generated by the beacon contains the randLocal r computed as $Hash(\rho_{i,1}||\rho_{i,2}||\rho_{i,3})$ and an argument of VC which attests to the correct computation of r and that the same $\rho_{i,1}$ and $\rho_{i,2}$ were used in the last pulse of the main beacon chain. Here, the VC algorithm takes

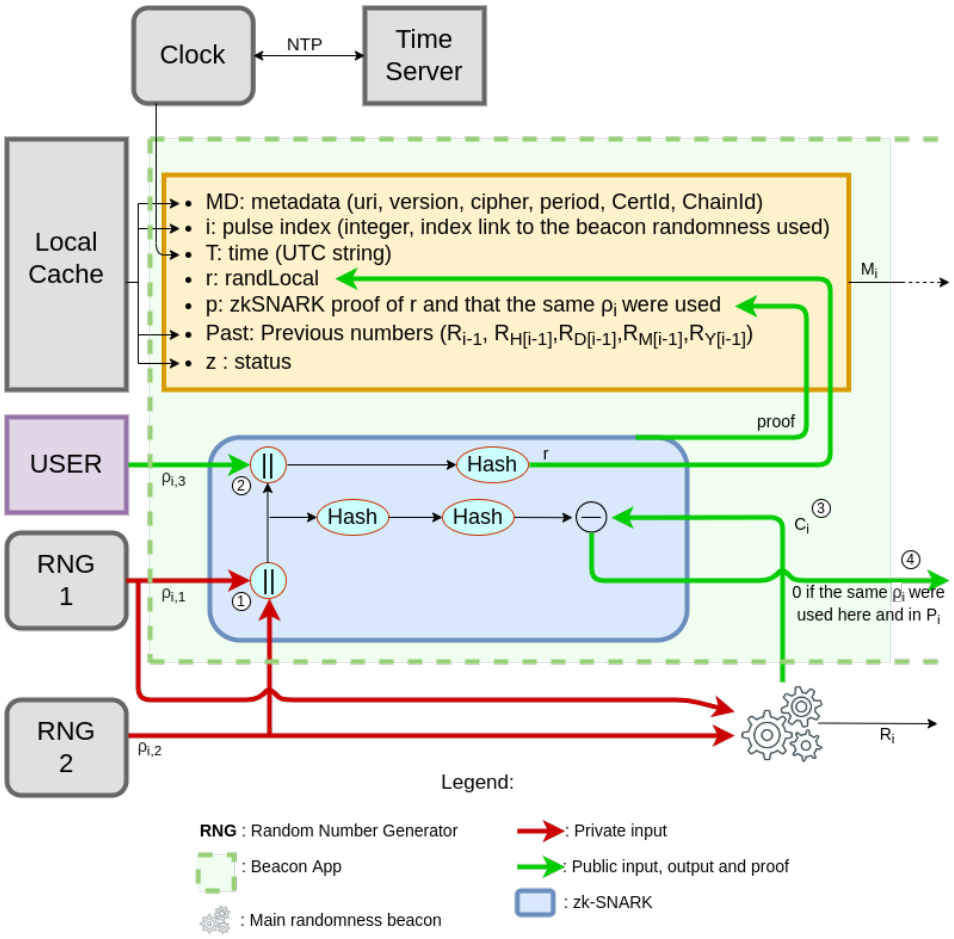

Figure 3: Generation of an orphan pulse with the interactive protocol.

$\rho_{i,1}$ and $\rho_{i,2}$ as private inputs so as not to disclose the (main chain) randLocal value which will be used in the (main chain) pulse ①, $\rho_{i,3}$ as public input to include the user's values ② and C_i as public input ③ in order to certify that $\rho_{i,1}$ and $\rho_{i,2}$ correspond to the commitment sent in the last pulse ④. All these steps are summarized in Fig. 3. If the beacon responds quickly (an order of seconds), the user is sure to have fresh randomness if their input is random.

Depending on the context, the user can generate a random number as described in (Kelsey et al., 2019) (i.e., as the hash of the message and the signature) or, if they want to avoid any potential beacon manipulation, they can use the provided randLocal value directly (as a VRF). The interactive protocol has the advantage over a VRF to provide external entropy.

In the case of a frequently consulted beacon, to ensure smooth scaling, it is possible to include several random seeds from different users in the same *orphan* pulse. Depending on the application and the context, the beacon must choose between one *orphan* pulse per user or regrouping several users for an *orphan* pulse at a regular time interval.

3.4 Connecting the Randomness Beacon to a Blockchain

The links between randomness and blockchains has been studied in several recent works (Bonneau et al., 2015), (Bunz et al., 2017), (Dharanikot et al.,). In this Section, we show that connecting a blockchain to our randomness beacon has several benefits for both systems.

3.4.1 Beacon Chain Immutability.

Even though the beacon chain provides a way to link the pulse and thus, to avoid data modifications *a posteriori*, we suggest decentralizing a part of the database at the backend, storing the pulses or the random numbers on a blockchain. This strategy should improve the resilience of pulses against modifica-

tion and remove vulnerabilities linked to a malicious database thanks to blockchain immutability. This part of the proposition is independent of the protocols proposed above.

3.4.2 Timestamped Random Number.

A user can use the interactive protocol to certify the freshness of the randomness by submitting an entropy input $\rho_{i,3}$ deduced from a recent block of a blockchain to the beacon. For example, assuming Ethereum takes approximately 14 seconds to forge a new block, if $\rho_{i,3}$ contains the nonce of the last block mined in the blockchain, every third party using the random number published after the protocol will be convinced that the number was not generated before the publication of the block on the blockchain. Indeed, if a malicious beacon is able to know the next nonce in advance, it can reverse a hash function, which is supposed to be impossible. This suggestion is dependent upon the existence of an interactive protocol or the use of a user input. It cannot be directly applied to the official NIST beacon, or it requires a slight modification in order to take a user value into account.

3.4.3 Randomness for Blockchains.

Blockchains often need a source of randomness for many internal protocols. With our propositions, detailed above, we can use the interactive protocol to automate the reception and use of random numbers. An oracle can send a query to the beacon at regular time intervals, asking it to furnish a random number with entropy deduced using the last block as input. The freshly generated randLocal value can be used as the new random number inside the blockchain. Indeed, we cannot store the whole pulse on a blockchain due to the size of M_i. However, we can use the randLocal value as randomness on the blockchain by storing it and its VC argument. By doing this, we ensure that the randomness is not manipulated by the beacon, and everything can be verified on the blockchain.

It should be noted that the oracle Chainlink provides a randomness service to smart contracts (Breidenbach et al., 2021). The principle consists of applying a Verifiable Random Function to a seed provided by the smart contract. Like this solution, our proposal is cryptographically verifiable, but it has the advantage of using external entropy.

4 SECURITY ANALYSIS

In this section, we will follow the NIST draft (Kelsey et al., 2019) and compare their analysis of possible attacks on their system with our propositions. We will reiterate any aspects necessary to the study of the security analysis.

4.1 Desired Properties

The main properties desired for a beacon are established in three essential categories in (Kelsey et al., 2019):

- relations that ensure the correct chaining process between each pulse.
- availability of pulses with reduced human intervention.
- random quality to ensure unpredictability and unbiased, fresh randomness in each pulse; these properties mainly come from the randLocal value.

The draft (Kelsey et al., 2019) asserts that the properties above derive from the pulse design itself when everything works as expected: without a breach, with a full entropy input, with synchronous and adequately fast communication and with a database that is available and reliable.

To achieve all of these required qualities, we based our model on cryptographic assumptions. Like in (Kelsey et al., 2019), we will assume that the hash functions used are one-way and collision resistant. We will also consider that the output of the hash function used is indistinguishable from true randomness when the input is unpredictable. Secondly, we will assume that the signature scheme is unforgeable and that its public key is certified correctly. For our model, we will use a verifiable computation scheme, more precisely, zk-SNARKs or zk-STARKs. These arguments provide three additional cryptographic properties:

- completeness: if the prover (the number generator in our case) is honest with a correct witness, they must convince a verifier.
- knowledge soundness: an honest verifier will not accept an invalid proof.
- zero-knowledge: the proof does not leak any information aside from the correct computation.

4.2 Adversary Model

In order to evaluate our system we need to describe what kind of adversary can attempt an attack, what their goal is and how they can reach it. We will consider two different adversaries with different capabilities. An adversary, or an attacker, can be semi-honest or malicious. A semi-honest, or passive, adversary

can leak information from the internal state but cannot alter the protocol or directly interact with it. On the other hand, a malicious attacker is able to modify the behavior of the structure depending on what they have access to. The latter case may well concern the malicious owner of the system. Our proposition is more general than that of (Kelsey et al., 2019) regarding the security assumptions by considering that the entire beacon can be fully malicious and by removing the HSM which was supposed safe.

Adversaries' goals are almost always the same but the benefits they could reap are plentiful. Firstly, someone may want to discover the next random number or information about the next number in advance. For example, if someone knows the parity of the next random number ahead of the time indicated in its timestamp, they are twice as likely to win if the number is used in a betting game. Secondly, an attacker may also want to directly induce a change in the behavior of the protocol in order to alter the next random number and once again be able to take advantage of it. With the same goal, an adversary can modify the database and change pasts numbers. Finally, the last case we studied is the ability to shut down the system, fork a chain or anything that can block users from accessing the random beacon for a certain period of time.

4.3 Attacks and Countermeasures

We will now present all of the attacks evoked in (Kelsey et al., 2019), how they can occur, which parts of the system have to be vulnerable and what is suggested in order to avoid them. For each of them, we will study the possibility of its occurrence and how our protocol is different from the NIST version. For each attack, we will specify which proposal improves the security of our overall model. Again, each proposal is independent and can be applied or not. We consider the complete system all three of our proposals.

4.3.1 Bias on RNG Outputs Due to a Malicious Beacon App.

In both the NIST scheme and ours, the ρ_i which are outputted from the RNGs, are never revealed to the public and an attacker could choose values to replace them arbitrarily. They can even directly replace $r_i = Hash(\rho_{i,1}||\rho_{i,2})$ due to the one-way nature of the hash function. Adding a XOR between r_i and R_{i-1} to produce the randLocal seed has been suggested. With this adjustment, an attacker could not completely foresee the final randLocal value which would come from $Hash(\rho_{i,1}||\rho_{i,2}) \oplus R_{i-1}$ and the commitment value $C_i = Hash(Hash(\rho_{i,1}||\rho_{i,2}))$. However, if the adversary is able to control the beacon app for several consecutive pulses, the fresh randomness property is not conserved.

By adding our first proposition, this attack is not possible. Even if the ρ_i are not revealed, the proof provided by the VC protocol gives the ability to verify that the proposed r_i is the result of a concatenation followed by a given hash function. In our case, an attacker who wants to alter the randomness of a pulse will have to anticipate all of the information within the pulse, i.e., two consecutive hashes, and a valid proof. The proof itself is non-deterministic by the zero-knowledge nature of VC and it changes for each execution due to the different random numbers used at each step in the protocol. Moreover, our scheme includes the modification suggested in (Kelsey et al., 2019). This attack is completely avoided with our system and renders the system more transparent thanks to the guarantee of the correct r_i computation.

4.3.2 Prediction and Exfiltration Due to a Malicious Beacon App.

As in the previous Section, a malicious adversary can alter the r_i or ρ_i values. By replacing these values with a symmetric encryption scheme that has an appropriate output length and outputs indistinguishable from random, an attacker gains the ability to communicate with someone else who shares the secret key without anyone knowing. This communication link can be used to exfiltrate internal information from the beacon or from any program accessed by the attacker. Another possible usage is the ability to completely predict the next pulses. When the encryption scheme is deterministic and the attacker sets the input of the encryption equal to the counter of the pulse, the external adversary can predict all of the next r_i values. The draft (Kelsey et al., 2019) does not provide any possible mitigation of this attack due to the indistinguishable nature of the encryption scheme, this attack can not be detected. The addition of a XOR between r_i and R_{i-1} is not sufficient to prevent it.

With the use of the proven beacon (first proposition), such a change is, here too, not a possibility. The proof will not be valid if a modification has been made on the public program and the random number will be rejected by the users. This is a significant improvement compared to the previous proposition, even if the generation of ρ_i can still be manipulated.

4.3.3 Bias on the Freshly-generated Random Number Due to a Malicious Beacon App.

If an adversary can make multiple requests to an honest HSM, they can ask for several signatures with different values of r_i until they obtain a result with a certain property enabling them to produce a final R_i with a desired attribute. They can also produce several pulses in advance to give themselves more time and be able to alter more bits of R_i.

A possible mitigation proposed in (Kelsey et al., 2019) is to use an HSM which imposes a delay between each signature or which limits the number of signature requests per established session. Another possible mitigation is to partition the beacon app in order to completely hide the result of the signature from the component that queries it.

This mitigation does not solve the problem and relies on an HSM, which we avoided in our system. Moreover, the adversary may be the operator of the system and if they have access to the beacon app, they would have access to all partitions including the HSM. Furthermore, the clients who receive the pulses cannot verify if the system is partitioned or not. This mitigation relies on the trustworthiness of the owner and only avoids external attacks, not those from all possible malicious adversaries.

Thanks to the interactive protocol, the beacon server has a shorter time range to attempt this attack because it cannot know the user's input in advance. We suggest introducing an X-second lag before the beacon can answer, which drastically reduces the time window for an adversary to generate multiple random numbers.

4.3.4 Advanced Knowledge and Prediction.

It is clear that the construction of the pulse requiring the knowledge of r_i creates a vulnerability if the beacon is semi-honest. In fact, the beacon stores the randLocal value for the next pulse during the creation of a new one. During this period of time, the beacon gains the knowledge of the next randLocal, one pulse before users and thus has the information to produce R_i. For the same reasons, the beacon also knows R_i before users due to the time it needs to wait before transmitting. If the beacon app is malicious, this leads to a prediction of r_i and R_i. Another attack leading to advanced knowledge of R_i relies on the internal clock. If this clock, or time-server used by the internal clock to synchronize itself, is malicious, it can give the beacon app a fake time allowing for the premature creation of a new pulse. If the beacon app remains honest, it would release the pulse to the database. A malicious database could then disclose the random numbers at the right time, gaining the knowledge of every number generated in advance of other users.

Three possible ways to reduce or remove the time interval during which an adversary has access to r_i before others are proposed in (Kelsey et al., 2019) to mitigate this attack. A first proposition is to change the definition of randLocal r_i. The draft reintroduces the XOR proposal between r_{i+1} and R_i which imposes a wait time on an attacker before they can access the knowledge of R_i and completely determine the value of randLocal. The two remaining mitigations rely on a change of what the beacon has access to and on a safer HSM. To avoid malicious clocks or time-servers, (Kelsey et al., 2019) submitted the idea to certify the internal clock or timestamp service in order to induce trust in the system. This does not remove the vulnerability, it redirects confidence to the timestamp service.

We followed the first proposition and inserted a XOR into the computation of randLocal, delaying the knowledge of r_{i+1} for the beacon. The interactive protocol is not at all vulnerable to these attacks due to the interactivity with the client. Even if the beacon's clock were malicious, the interactivity drastically reduces the time during which the new random number is known by the beacon, and a database is not involved. Moreover, the client can check if the time furnished by the beacon is in sync with their own local clock. The beacon cannot compute the random number before it has access to the client's input, delaying the knowledge of the input for the beacon.

Secondly, with our third proposition, by using the entropy from the last block of a blockchain as a part of the public input, everyone will agree that the random number is fresh and was not known until it was released.

4.3.5 History Modification Due to a Malicious Database and Signature Key Access.

When an adversary has access to the database and the signature module key stored in the HSM, they have all the tools needed to completely alter the information stored in the database. The only thing they cannot change is the link between two pulses due to the commitment value C_i. Consequently, they can compute their own chain of random numbers and replace the entire chain or the end of the existing one. The NIST draft promotes the multiplication of external repositories. This attack is not completely avoided by this idea.

Our system highly supports the decentralization of the database and suggests using a blockchain to store, at least, the proofs or the random numbers, and exploiting the immutability and distributed properties of

blockchains.

5 PERFORMANCE EVALUATION

This section describes what we used to evaluate our system. We propose using of a new generation of tools that enable drastically reducing the memory usage of the prover as well as the argument generation time. After that, we discuss the performances obtained, which demonstrate that deploying our system is realistic on a standard computer.

5.1 Hash Functions Used

Today, programs are optimized to be evaluated as quickly as possible. This often implies the use of operations close to the CPU and therefore Boolean operations. However, these operations are expensive to prove. It is therefore important to find programs that are no longer just optimized for evaluation but for evaluation and proving. Using arithmetic operations instead of boolean operations can increase the evaluation time but greatly reduce the proof time, thus improving overall complexity.

Even if VC protocols can have a succinct proof size and verifier constant time computation, like zk-SNARKs, the complexity of the prover is at least linear in the number of constraints. With this in mind, even if classical hash functions are optimized for CPU usage and very fast, they have a lot of constraints due to the transformation of their logic construction into arithmetic. In section 5.2, we show that SHA256 with a 1024-bit input has around $90,000$ constraints in the R1CS model (for SNARKs). It is important to note that for SNARKs, the size of the prover key is linear in regard to the number of constraints.

However, several propositions of an alternative SNARK-friendly (also STARK-friendly) hash function, which is arithmetically oriented, have appeared in the literature over the past few years. They aim to be more efficient regarding the number of constraints. Following the lead of a recent survey (Ben-Sasson et al., 2020) based on multiple security analyses, such as (Beyne et al., 2020) and (Canteaut et al., 2020), we will consider the usage of the *Rescue* hash function (Aly et al., 2020). The *Rescue* function is based on a sponge construction and can output different sizes of data depending on the chosen parameters. In order to compare the *Rescue* and *SHA* functions, we implemented their R1CS circuits using Circom (cir, 2020) and generated their proofs and witnesses with SnarkJS (sna, 2020). However, SnarkJS only relies on the $BN128$ and $BLS12-381$ curves which are 128-bit security curves. We decided to only evaluate 128-bit security hash functions to remain consistent. We will evaluate our proposition based on the SHA256 and *Rescue* function. Indeed, contrary to the NIST suggesting the use of SHA512, it would not be coherent to prove a hash function on 256 bits of security with 128 bits of security. This is why we evaluated SHA256 which brings 128 bits of security. The *Rescue* hash function is interesting for its relatively small number of constraints, allowing for faster computation and lighter memory usage. Also, the use of the *Rescue* function drastically reduces prover and verifier key sizes for SNARKs, as well as memory consumption.

Even though this function is arithmetically oriented and much faster to prove, it is slower to evaluate than SHA256 which is oriented for CPU use. It is important to differentiate the proving time and the evaluation time. At our scale, with only a thousand bits in input, the *Rescue* hash functions take 7 times less time to be evaluated and proved than SHA256 with our implementation.

5.2 Implementation and Evaluation

For our implementation, we propose the use of the Groth16 zk-SNARK scheme (Groth, 2016) which is very efficient in terms of communication (proof size) and verifier computation complexity, both in $O(1)$. In terms of computation and memory usage, this complexity is $O(Nlog(N))$. For the trusted setup phase, we reuse the ceremony done in 2018 on the Zcash blockchain (zca, 2018) in order to involve many famous participants without redoing all of the organization.

We chose the Groth16 protocol because the proof is tiny and this is important for our global system. Groth16 proof size is less than 200 bytes while STARK of two *Rescue* with weaker parameters than ours is around 11 KB according to the evaluation based on winterfell (win, 2021). The blockchain's block size is a key parameter. Most of existing blockchains only support a small amount of information per transaction, or the fees paid to include the transaction increase with size. The second is because the blockchain verification cost is feasible for Groth16 and costly for STARKs (around 300k gas on Ethereum for Groth16 and estimated at more than 2.5M for STARKs). Finally, the prover and verifier keys are small enough due to the minuscule amount of constraints needed for our protocols.

To evaluate our system, we used Circom to implement the *Rescue* and SHA256 circuits and evaluated their proof generation using snarkJS. The SHA256

Table 1: Evaluation of the different circuits on 100 executions.

	constraints	witness	prover key	average proving time
SHA256 (1024 bits)	89,698	2.9 MB	59.9 MB	2.710 s
Rescue (1016 bits)	1,196	38.3 KB	727 KB	382 ms
zkBeacon (Section 3.2.1)	1,667	53.4 KB	961.5 KB	408 ms
interactive protocol (Section 3.3)	2,390	76.6 KB	1.5 MB	447 ms

function has already been implemented by Circom in the Circomlib available on github. We implemented the *Rescue* function according to the *Rescue-prime* specification (Szepieniec et al., 2020). We used a 32 GB RAM computer with an Intel Core i7-10850H CPU at 2.7 GHz on a 64-bit Ubuntu 20.04.3 LTS.

We implemented *Rescue* using parameters related to elliptic curve choice. Having chosen the *BN*128 curve, we generated all other parameters based on a capacity of 3 elements, a state width of 4 and a security level of 128 bits with a 40% margin. For the prime field, we constructed this *Rescue* function to be proved with *BN*128 pairing. Therefore, when we implement circuits onto it, its operations are done in the curve's scalar field. Then, all operations are executed modulo the group order of the curve. To lower the amount of constraints, we have chosen to take this group order as the prime number used in *Rescue*. To remain coherent with our system, we evaluated this function with 4 elements of $\mathbb{F}_p$ as input which is approximately 1016 bits.

We evaluated the interactive protocol with the same *Rescue* function. As private inputs, we took the same length of 4 elements of $\mathbb{F}_p$ and for the client's public input, we took 2 elements of $\mathbb{F}_p$, for a total of 1524 bits. In the end, the circuit is almost entirely 3 *Rescue* functions: one with 6 elements of input, one with 4 and one with 1. Our results were obtained by averaging over 100 executions and are summarized in Table 1.

Thanks to this evaluation, we can state that our proposition can be easily deployed on a standard computer. Indeed, according to these results evaluated on a standard personal computer, a zk-SNARK of a pulse and a proof for the interactive protocol can be both generated in less than 0.5 seconds. Moreover, SnarkJS is not optimized and most of the measured time is dedicated to loading the verifier key and initializations.

6 CONCLUSION

Our proposition aims to improve the randomness beacon concept standardized by the NIST allowing the production of random numbers at regular time intervals. Our main objective is to keep the main interesting concepts of randomness beacons, i.e., the use of external entropy and the chaining of the pulses, but also to strengthen user confidence in the outputs. For that, we proposed the addition of a verifiable computation argument on the first hash functions used in each pulse generation, making our system more transparent.

The second part of our proposal is an interactive protocol, limiting the possibility of timing attacks and the lack of fresh randomness. We also suggest connecting the database to a blockchain in order to improve the immutability of the database, provide timestamped random numbers and use the randomness beacon as a proven random number oracle for the blockchain. We proved that our scheme supports a security model with a malicious beacon generator. Finally, we benchmarked an implementation of our proposal based on the *Rescue* hash function and Groth16 zk-SNARK and proved that the constraints in terms of memory and CPU, are acceptable, even for a personal computer.

ACKNOWLEDGEMENTS

The authors would like to thank Jonathan Detchart, Thibault Gateau and Caroline Chanel for their help on several aspects of this work.

REFERENCES

(2018). Zcash powers of taus ceremony attestation. Accessed: 24/03/2022.

(2020). CIRCOM: Circuit Compiler for Zero-Knowledge Proofs. Accessed: 24/03/2022.

(2020). Inmetro Randomness Beacon. Accessed: 24/03/2022.

(2020). Nist Randomness Beacon. Accessed: 24/03/2022.

(2020). SNARKJS: JavaScript Implementation of zk-SNARKs. Accessed: 24/03/2022.

(2021). Winterfell : A STARK prover and verifier for arbitrary computations. Accessed: 28/04/2022.

Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., and Szepieniec, A. (2020). Design of symmetric-key primitives

for advanced cryptographic protocols. *IACR Transactions on Symmetric Cryptology*, pages 1–45.

Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*.

Ben-Sasson, E., Goldberg, L., and Levit, D. (2020). STARK Friendly Hash-Survey and Recommendation. *IACR Cryptol. ePrint Arch.*, 2020:948.

Beyne, T., Canteaut, A., Leander, G., Naya-Plasencia, M., Perrin, L., and Wiemer, F. (2020). On the security of the Rescue hash function. *IACR Cryptol. ePrint Arch.*, 2020:820.

Bonneau, J., Clark, J., and Goldfeder, S. (2015). On Bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015.

Bordage, S. and Nardi, J. (2020). Interactive Oracle Proofs of Proximity to Algebraic Geometry Codes. *arXiv preprint arXiv:2011.04295*.

Bowe, S., Gabizon, A., and Miers, I. (2017). Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. Cryptology ePrint Archive, Report 2017/1050. https://ia.cr/2017/1050.

Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., et al. (2021). Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks.

Bunz, B., Goldfeder, S., and Bonneau, J. (2017). Proofs-of-delay and randomness beacons in Ethereum. In *IEEE Security & Privacy on the Blockchain (IEEE S&B)*.

Canteaut, A., Beyne, T., Dinur, I., Eichlseder, M., Leander, G., Leurent, G., Plasencia, M. N., Perrin, L., Sasaki, Y., Todo, Y., et al. (2020). Report on the Security of STARK-friendly Hash Functions (Version 2.0).

Dharanikot, S., Jensen, M., Kristensen, S., Michno, M., Pignolet, Y., Hansen, R., and Schmid, S. Breeding Unicorns: Developing Trustworthy and Scalable Randomness Beacons.

Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*.

Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208.

Groth, J. (2016). On the size of pairing-based noninteractive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer.

Jacak, M., Jóźwiak, P., Niemczuk, J., and Jacak, J. (2021). Quantum generators of random numbers. *Scientific Reports*, 11(16108).

Kelsey, J., Brandao, L., Peralta, R., and Booth, H. (2019). Reference for Randomness Beacons, Format and Protocol Version 2, Draft NISTIR 8213. Technical report. Accessed: 24/03/2022.

Kiayias, A., Tsiounis, Y., and Yung, M. Traceable Signatures. In *Advances in Cryptology - EUROCRYPT 2004*.

Micali, S., Rabin, M., and Vadhan, S. (1999). Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE.

Moran, T. and Naor, M. (2010). Split-Ballot Voting: Everlasting Privacy with Distributed Trust. *ACM Trans. Inf. Syst. Secur.*, 13(2).

Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE.

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

Szepieniec, A., Ashur, T., and Dhooghe, S. (2020). Rescue-Prime: a standard specification (SoK). *Cryptology ePrint Archive*.

UChile, R. (2020). Public, Transparent and Verifiable Randomness. Accessed: 24/03/2022.