Proof-of-Useful-Randomness: Mitigating the Energy Waste in Blockchain Proof-of-Work

Efe Ulas Akay Seyitoglu[1], Attila Altay Yavuz[1] and Thang Hoang[2]
[1]*Department of Computer Science and Engineering, University of South Florida, Tampa FL, USA*
[2]*Department of Computer Science, Virginia Tech, Blacksburg VA, USA*
{efe3, attilaayavuz}@usf.edu, thanghoang@vt.edu

Keywords: Blockchain, Proof-of-Work

Abstract: Proof-of-Work (PoW) is one of the fundamental and widely-used consensus algorithms in blockchains. In PoW, nodes compete to receive the mining reward by trying to be the first to solve a puzzle. Despite its fairness and wide-availability, traditional PoW incurs extreme computational and energy waste over the blockchain. This waste is considered to be one of the biggest problems in PoW-based blockchains and cryptocurrencies. In this work, we propose a new useful PoW called Proof-of-Useful-Randomness (PoUR) that mitigates the energy waste by incorporating pre-computed (disclosable) randomness into the PoW. The key idea is to inject special randomness into puzzles via algebraic commitments that can be stored and later disclosed. Unlike the traditional wasteful PoWs, our approach enables pre-computed commitments to be utilized by a vast array of public-key cryptography methods that require offline-online processing (e.g., digital signature, key exchange, zero-knowledge protocol). Moreover, our PoW preserves the desirable properties of the traditional PoW and therefore does not require a substantial alteration in the underlying protocol. We showed the security of our PoW, and then fully implemented it to validate its significant energy-saving capabilities.

1 INTRODUCTION

The proof-of-work (PoW) algorithm provides a means of consensus in blockchains and therefore plays a central role in various blockchain applications. The traditional PoW puzzle has been shown to be fair and is widely adopted in practice. Specifically, a traditional PoW (at minimum) satisfies the following properties: (i) *Hardness:* Each attempt to solve the PoW puzzle should succeed in a similar probability. This gives incentives to invest in computational resources because the puzzle rate (i.e., the hashing effort) grows linearly with the possibility of finding a solution. (ii) *Efficiency:* A puzzle must have a solution, and should be generated/verified in polynomial time. *(iii) Completeness:* Only a valid puzzle solution should be accepted by the verifier (Ball et al., 2018).

The traditional PoW accounts for about 90% of the total market in digital cryptocurrencies (Gervais et al., 2016), as well as dominating the blockchain-based applications. However, despite its merits and wide-use in practice, the traditional PoW also has been shown to be extremely energy costly. For instance, Bitcoin was measured to have consumed 39 TWh of electricity in 2018 which at the time was more than the electricty consumption of Qatar and Bulgaria (Bahri and Girdzijauskas, 2018). A big chunk of these electricity consumption are due to the computational wastefulness of traditional PoW algorithms. This severe energy waste is considered as one of the biggest disadvantage of traditional PoW (Gervais et al., 2016). Various alternative consensus algorithms have been developed to reduce the energy waste in blockchains. Below, we first outline some of the prominent approaches related to our work, and then elaborate our contributions in this paper.

2 RELATED WORK

The existing PoW algorithms can be classified into three groups: (i) Traditional Proof-of-Work, (ii) Proof-of-Non-Wasteful-Work, (iii) Proof-of-Useful-Work.

• *Traditional Proof-of-Work:* The most common example for such consensus algorithms is the hash-based Bitcoin PoW algorithm. These type of algorithms are based on the fact that the hash function output is pseudorandom. The puzzle is constructed in a way that, every miner in the network tries to couple an arbitrary nonce with the block content whose hash will be satisfying the puzzle (Meng et al., 2018). The more computational power a miner has, the more nonce values they can try which can potentially end up satisfying the requirements of the puzzle. We take the Bitcoin PoW algorithm as a starting point and thus, will elaborate on this idea further in the proposed PoW algorithm section.

Note: There are other variants of PoW algorithm such as Number Theoretic PoW which is used in popular cryptocurrencies such as PrimeCoin (King, 2013). Another important variants are linear algebra based (e.g. (Ball et al., 2017)) or graph theoretic ones (e.g. (Tromp, 2015)). However, the focus of this paper is going to be a hash-based approach for two reasons: (i) our construction is on top of the traditional hash-based Bitcoin PoW, (ii) the hash-based PoW dominates the cryptocurrency market (Loe and Quaglia, 2018).

• *Proof-of-Non-Wasteful-Work:* A proof-of-non-wasteful-work, is an algorithm that does not consume computational resources for establishing consensus. One good example is the the proof-of-stake (PoS) algorithm. In this algorithm, the nodes deposit certain amount of financial resources as stakes. The node that will add a new block to the blockchain is chosen with a probability that is linear to the stake they deposit. PoS is a clever attempt to solve the electricity/computation waste problem. An instantiation of the PoS puzzle idea can be found in (Bentov et al., 2016). PoS has also been partially used in other novel PoW schemes (e.g. (Bentov et al., 2014)). Furthermore, Krol et. al. proposed Proof-of-Prestige (PoP) which is built on top of PoS. In PoP, the users earn prestige via performing useful work which in turn increases their chances of obtaining the mining reward (Król et al., 2019).

However, PoS is not perfect due to its innate foundational differences (i.e. incentivizing the miner via stakes instead of compuatation power) compared to PoW. This is a problem because many popular blockchains (e.g. Bitcoin) use the wasteful PoW as the consensus algorithm (Nakamoto, 2009). Therefore, a big algorithmic jump from PoW to a completely different consensus algorithm (PoS) would require costly development efforts. Furthermore, coin holders in the network would need to be convinced to participate in a whole different consensus algorithm which could be problematic (Kiayias et al., 2017). Another fundamental difference between traditional PoW algorithms and the PoS algorithm is the source of pseudorandomness. In a desirable PoW algorithm, the pseudorandomness should be external. In other words, finding the solution to the PoW puzzle should have no relation with the cryptocurrency itself but should be a process that is dependent on external factors such as hash output. In the PoS algorithm, the pseudorandomness is internal (Brown-Cohen et al., 2019).

The PoS variants also suffer from the "nothing at stake" problem. In PoS, validators can maximize their earning by joining various forks (and generating clashing blocks) without taking on extra risk for losing the stake (Li et al., 2017). Some important approaches to mitigate this problem can be found in (Kiayias et al., 2017). Another concern with PoS is the security impact of stale blocks as described in (Gervais et al., 2016). GHOST protocol works to solve this problem (Gervais et al., 2016).

Another good example is the Intel's proof-of-elapsed-time (PoET) which is based on time instead of execution power. The first node to finish waiting gets the mining reward (Chen et al., 2017). This algorithm is a good candidate to solve the PoW electricity consumption problem, but it does not incentivize nodes and has a considerable deployment cost on PoW dominated blockchains.

• *Proof-of-Useful-Work (PoUW):* This type of algorithms produce a useful component during consensus. One good example is from Ball et. al. who proposed a PoUW that helps tackle computational problems (3SUM, Orthagonal Vectors etc.) while solving the PoW puzzle (Ball et al., 2017). Another good approach is by (Loe and Quaglia, 2018) who generated an instance of the Travelling-Salesman problem which is known to be NP-Hard during the PoW consensus. There are other notable approaches such as Hastings et. al., who proposed a PoUW algorithm that computes the discrete logarithm of a cyclic group during consensus (Hastings et al., 2019) or by Lihu et. al. who incorporates ML training into puzzle solving (Lihu et al., 2020) to save energy.

To sum up, the traditional (wasteful) PoW is still the most widely adopted consensus algorithm despite its wastefulness. Therefore, we need a new useful PoW algorithm that stays in the realm of traditional PoW consensus but also mitigates its energy waste problem (Nakamoto, 2009).

3 Our Contribution

In this work, to the best of our knowledge, *we proposed the first PoUW that creates useful-randomnesses, which not only substantially reduces the energy waste, but also preserves the main PoW architecture, thereby avoid the hurdles of heavy* *transition costs of alternative consensus algorithms. We name our scheme as proof-of-useful-randomness (*PoUR*).*

Main Idea: In the traditional PoW, one-time randomnesses (nonces) are used as a solution to the puzzle. However, this is expensive and wasteful, as generating a bitcoin block takes on average 2^{32} attempts (O'Dwyer and Malone, 2014), and mostly importantly, those nonces are just discarded (wasted). Our key observation is that the wasted randomness generation in PoW can be turned into an opportunity, if one can produce (special) useful randomness that can be harnessed by the cryptographic algorithms. Our PoUR algorithm randomly generates (one-way) algebraic commitments that not only serve as the randomness source for PoW but also can be integrated into a vast range of public key cryptographic primitives. We call these useful randonmness as Precomputed Algebraic Commitments (PAC), which can be integrated into various cryptographic schemes in offline-online settings (e.g., digital signatures, key exchange). We further elaborate the desirable properties of our scheme as follows:

- Energy-Saving and Efficient Consensus: PoUR saves considerable energy by creating useful-randomnesses that can be used by various cryptographic operations as needed. For example, after 2^{30} puzzle-solving attempts, PoUR saves around $2.18 \cdot 10^5$ J worth of energy in elliptic-curve settings. This energy-saving process does not come with the cost of any additional efficiency overhead to the protocol since the target value of the puzzle could be adjusted to accommodate the desired hardness (i.e. the necessary number of attempts to solve the puzzle on average).
- Security and Open-Source Implementation: We present a security discussion for PoUR and provide the implementation of our proposed scheme on commodity hardware. We will open-source our implementation.
- Potential Use-Cases: PoUR is applicable to a wide-array of cryptographic schemes that admit an offline-online phase, in which PACs can be generated via PoUR and then later consumed by cryptographic scheme. Some example cryptographic schemes that receive benefit from PoUR includes ,but not limited to digital signatures, key-exchange, public-key encryption, zero-knowledge proofs, random beacons and their quantum-safe variants. We have shown two specific PoUR instantiations with elliptic-curve and lattice based digital signatures.
- Ease-of-Integration: PoUR algorithm requires minimal alteration to the existing and traditional PoW algorithms in popular platforms such as Bitcoin. We save commitments during consensus without impacting the other parts of the system.

4 Preliminaries

We now present the prelimineries of our scheme.

Definition 1. *An Algebraic One-way function* AOWF *consist of two algorithms* Gen,F *(Catalano et al., 2013):*

1. $K \leftarrow \mathsf{Gen}(1^\kappa)$*: Given the security parameter* κ, $\mathsf{Gen}(1^\kappa)$ *outputs a key* K*. The key* K *determines the order of the groups* A_K *and* B_K *for* F*.*
2. $F_K : A_K \to B_K$*: For input* $a \in A_K$*, there is an output* $b \in B_K$ *such that* $b = F_K(a)$ *is computed in polynomial-time. However, given b, it should be infeasible for a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *to come up with a function* F' *such that* $Pr[F'(a') = b] \geq \varepsilon$.
3. $\forall\, \kappa \in \mathbb{N}$, $K \leftarrow \mathsf{Gen}(1^\kappa)$ *such that K constitutes an order for* A_K *and* B_K *in* $F_K : A_K \to B_K$.
4. *We can perform efficient group operations:* $F(A_K) \bigoplus F(B_K) = F(A_K \bigoplus B_K)$ *where* $\bigoplus$ *is an operator over the ring R.*

AOWF is popular in many cryptographic domains such as ECDLP, DLP and Post-Quantum Cryptography. We leverage them in PoUR for energy saving purposes.

One of the most popular AOWF is the hash function H. A hash function is a one-way function takes in an arbitary length input and produces a fixed output. We leveraged SHA-256 in our constructions as an AOWF. Bitcoin also leverages the SHA-256 hash function durining consensus (Nakamoto, 2009).

Definition 2. *Given an elliptic curve* $E(\mathbb{F}_q)$ *over a finite field* $\mathbb{F}_q$*, ECDLP asks to find a value* $k \in \mathbb{Z}_q$ *such that* $P_2 = k \cdot P_1$ *where* P_1, P_2 *are points on* $E(\mathbb{F}_q)$ *(Behnia et al., 2019).*

Definition 3. ***Short Integer Solution Problem (SIS):*** *Given a collection of n-dimensional modulo q (prime number) vectors* $\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_m \in \mathbb{Z}_q^n$*, the SIS problem asks to find a collection of non-trivial* $s_1, \ldots, s_m \in \{-1, 0, 1\}$ *such that* $s_1 \cdot \tilde{\mathbf{v}}_1 + \ldots + s_m \cdot \tilde{\mathbf{v}}_m = \tilde{\mathbf{v}}_0 \in \mathbb{Z}_q^n$ *where* $\tilde{\mathbf{v}}_0$ *is an n-dimensional vector that contains all zeros (Peikert, 2016).*

Definition 4. ***Learning with Errors Problem (LWE):*** *Given dimension n, modulo q, an error distribution e such that* $\sqrt{n} \leq e \leq q$*, and* $b \in \mathbb{Z}_q$*, the LWE problem asks to find a secret* $s \in \mathbb{Z}_q^n$*, such that* $b = \langle \mathbf{s}, \mathbf{a} \rangle + e \in \mathbb{Z}_q$ *where* $\mathbf{a} \in \mathbb{Z}_q^n$ *and* $\langle \mathbf{s}, \mathbf{a} \rangle$ *denotes the inner-product* *between* **s** *and* **a**. *The decision version of the same problem asks to distinguish a collection of pairs from truly random pairs (Peikert, 2016).*

Definition 5. ***Cryptographic Pseudorandom Number Generators (CPRNG):*** *A* CPRNG*, G is a deterministic algorithm that takes a uniform input seed and produces a pseudorandom output. Given a short uniform seed s, G is a* CPRNG *satisfies the following properties:*
• $\forall s$, $L(G(s)) > L(s)$ *where L is gives the string length.*
• $|Pr[\mathcal{T}(G(s)) = 1]$ - $Pr[\mathcal{T}[(r) = 1]]| \leq \varepsilon$ *where* ε *is a negligible value,* $\mathcal{T}$ *is an efficient statistical randomness test (distinguisher), and* $r \in \{0,1\}^{L(G(s))}$ *(Katz and Lindell, 2014).*

Definition 6. ***Pre-computed Algebraic Commitments (PAC):*** *PACs are expensive cryptographic tokens created in the offline phase and directly used in the online phase for computational saving. Creation of a PAC consists of three steps:*

1. $r_i \overset{\$}{\leftarrow} \mathcal{D}$ *for all i = 1 to K where* r_i *is selected from is selected from a specified distribution* $\mathcal{D}$ *which is random-uniform.*
2. $R_i \leftarrow \mathsf{AOWF}(r_i)$ *for all i = 1 to K (All* R_i *are disclosable randomnesses)*
3. $\mathcal{U} \leftarrow \mathcal{U} \cup \{(r_i, R_i)\}$ *for all i = 1 to K where* $\mathcal{U}$ *is a set that stores the commitments.*

Definition 7. *A PoW scheme has three algorithms (Ball et al., 2018):*

- $c \leftarrow \mathsf{Gen}(1^\kappa)$*: Generates the challange c given the security parameter* κ*.*
- $\pi \leftarrow \mathsf{Solve(c)}$: *Satisfies the requirements of the challenge c and generates the proof* π*.*
- $\{0,1\} \leftarrow \mathsf{Ver}(c, \pi)$*: Verifies that the proof* π *satisfies the requirements of the challenge c.*

Miners try to solve the puzzle by finding a nonce v such that $H(B_C \parallel v) < T$ (target value) where $\parallel$ denotes concatenation and B_C denotes the block content. The first miner with such a nonce keeps the reward.

5 Proposed Proof-of-Work Algorithm

The classical PoW compares the target value with H($B_C \parallel v$) (v is the nonce). If the comparison fails, PoW increases the nonce by one and compares the hash output against the target value. The idea in traditional Bitcoin PoW is that the miners try out different nonces for which when appended with the block content, will result in a hash output that is smaller than the target. In PoUR we replace the duty of nonces with disclosable PACs and store them for energy-saving. Storing PACs is essential for our scheme because the usage of PACs are common in blockchain-based public-key infrastructure where offline-online cryptography is leveraged. Furtermore, in our PAC creation we leverage B_R. This prevents miners to pre-create and use these PACs prior to mining.

PoUR Energy Saving: PACs are computationally expensive to create and are neccessary in cryptographic operations. Our PoUR construction pre-computes PACs during consensus and stores them. The stored PACs results in energy-efficiency because instead of creating them in future cryptographic methods, we fetch the stored ones.

Generic PoUR Algorithm: We now present the generic construction of our PoUR algorithm. We use this construction as a starting point to show the usage of PoUR in other cryptographic settings such as offline-online cryptographic settings as well as instantiations in elliptic-curve and post-quantum settings. We note that $\overrightarrow{m} : (m_1, \ldots m_n)$ means a vector of messages. $\{0,1\}^*$ denotes a string with arbitrary length and B_R denotes the block randomness.

Algorithm 1 Generic Constructions of PoUR

```
    $c \leftarrow \mathsf{PoUR.Gen}(1^\kappa)$ :
1:  Target value $T$ is determined
2:  Let $(B_C, B_R) \in \{0,1\}^*$
3:  return $c = (B_C, T, B_R)$

    $\pi \leftarrow \mathsf{PoUR.Solve}(c = (B_C, T, B_R))$:
1:  $i = 0$
2:  do
3:      $s_i \overset{\$}{\leftarrow} \mathcal{D}$
4:      $r_i \leftarrow G(s_i \parallel B_R)$
5:      $R_i \leftarrow \mathsf{AOWF}(r_i)$
6:      $\mathcal{U} \leftarrow \mathcal{U} \cup \{(r_i, R_i)\}$
7:      $i = i + 1$
8:  while $H(B_C \parallel R_i) \geq T$
9:  return $\pi = (s_{i-1}, R_{i-1})$
    Note: We disclose only a single PAC out of the
    $2^{32}$ (on average) PACs we save. Therefore, our
    $2^{32}$ (on average) -1 PACs are processable and we
    save the energy that would have been needed to
    create the commitments.

    $\{0,1\} \leftarrow \mathsf{PoUR.Ver}(c = (B_C, T, B_R), \pi = (s, R))$:
1:  $r' \leftarrow \mathsf{AOWF}(G(s \parallel B_R))$
2:  if $\mathsf{AOWF}(r') = R$ & $G(B_C \parallel R) < T$ then
3:      return 1
4:  return 0
```

In our PoUR.Solve method, we store the PACs. These pre-computed PACs could later be utilized in many cryptographic methods instead of creating them from stratch (which results in energy efficiency). In the subsequent sections, we show how we leverage PoUR in cryptographic applications.

Usage in Offline-Online Cryptography: In blockchains with public-key infrastructure, online stage execution time is crucial. Thus, we store the pre-computed and compuationally expensive algebraic commitments in $\mathcal{U}$ during the offline-phase for faster execution in the online phase. CryptOp.process is a function for cryptographic operations like encryption, signing etc. whereas CryptOp.verify refers to its verification.

Algorithm 2 Offline-Online Settings with PoUR

```
$\mathcal{U} \leftarrow$ PoUR.Offline($1^\kappa$)
1: $c \leftarrow$ PoUR.Gen($1^\kappa$)
2: $\pi \leftarrow$ PoUR.Solve($c$) //PACs are in $\mathcal{U}$
3: if PoUR.Verify($c$, $\pi$) = 0 then
4:     Go Back to Step 1

$(\mathcal{F}, \mathcal{R}) \leftarrow$ PoUR.Online($\overrightarrow{m}$, $\mathcal{U}$)
1: for $i = 1$ to $n$ do
2:     $(r_i, R_i) \leftarrow$ Pick and remove from $\mathcal{U}$
3:     $\mathcal{R} \leftarrow \mathcal{R} \cup \{(R_i)\}$
4:     $\mathcal{F} \leftarrow \mathcal{F} \cup$ CryptOp.process($r_i$, $R_i$, $m_i$)
   return $(\mathcal{F}, \mathcal{R})$

$\{0,1\} \leftarrow$ PoUR.Verify($\overrightarrow{m}, \mathcal{F}, \mathcal{R}$)
1: Let $\mathcal{F} : (f_1, .., f_n)$
2: Let $\mathcal{R} : (R_1, .., R_n)$
3: $\overrightarrow{m} = (m_1, .., m_n)$
4: for $i = 1$ to $n$ do
5:     $f_i \leftarrow$ Pick from $\mathcal{F}$
6:     $R_i \leftarrow$ Pick from $\mathcal{R}$
7:     if CryptOp.verify($R_i$, $f_i$, $m_i$) = 0 then
8:         return 0
9: return 1
```

5.1 Generic Constructions of PoUR

We now describe our elliptic-curve and lattice instantiations with PoUR. The core idea is to generate commitments during consensus and use them in cryptographic operations later. If it were not for the PoUR, these commitments would need to be created from scratch which would be computationally costly.

Note: The type of commitments that are saved in PoUR.Solve is dependent on the underlying cryptographic instantiation.

PoUR Construction in Elliptic Curves: We now show how PoUR uses PACs and saves energy in ECDLP settings. We chose SchnorrQ signatures (Schnorr, 1991) but any digital signature with special features (e.g., forward-security, aggregation (e.g. (Seyitoglu et al., 2020)) would benefit from PoUR. We denote (k, K) as a public, private key pair. The key generation, signature generation and verification functions are denoted by Kg, Sig, Ver respectively. R_x is the x-coordinate of R. P is the elliptic curve generator.

Algorithm 3 PoUR Construction in Elliptic-Curves and its usage in SchnorrQ Signature

```
$c \leftarrow$ PoUR.Gen($1^\kappa$) :
1: Target value $T$ is determined
2: Let $(B_C, B_R) \in \{0,1\}^*$
3: return $c = (B_C, T, B_R)$

$\pi \leftarrow$ PoUR.Solve($c = (B_C, T, B_R)$):
1: $i = 0$
2: do
3:     $s_i \xleftarrow{\$} \mathbb{Z}_q^*$
4:     $r_i \leftarrow H(s_i \,||\, B_R)$
5:     $R_i \leftarrow r_i \cdot P$ //$P$: Elliptic Curve Generator
6:     $\mathcal{U} \leftarrow \mathcal{U} \cup \{(r_i, R_i)\}$
7:     $i = i + 1$
8: while $H(B_C \,||\, R_i) \geq T$
9: return $\pi = (s_{i-1}, R_{i-1})$

$(0,1) \leftarrow$ PoUR.Ver($c = (B_C, T, B_R), \pi = (s, R)$):
1: $t \leftarrow H(s \,||\, B_R)$
2: if $P \cdot t = R$ & $H(B_C \,||\, R)) < T$ then
3:     return 1
4: return 0

$(k, K) \leftarrow$ SchnorrQ.Kg($1^\kappa$):
1: $(k, K) \leftarrow$ Pick and remove from $\mathcal{U}$
2: return $(k, K)$

$\sigma \leftarrow$ SchnorrQ.Sig($k, M$):
1: $(r, R) \leftarrow$ Pick and remove from $\mathcal{U}$
2: $e \leftarrow H(M \,||\, R_x)$ //$R_x$: $x$ coordinate of the point $R$
3: $s \leftarrow r - e \cdot k$
4: return $\sigma = (s, e)$

$\{0,1\} \leftarrow$ SchnorrQ.Ver($K, M, \sigma = (s, e)$):
1: $R' \leftarrow K \cdot e + P \cdot s$
2: if $e = H(M \,||\, R'_x)$ then
3:     return 1
4: else return 0
```

PoUR Construction in Lattices: PoUR can be useful in Lattices. φ is the coefficient threshold. $\mathbf{R}_q^l$ is a vector with l elements. $\mathbf{R}_q^{m\times l}$is a matrix with dimensions $m \times l$. $\mathsf{S}_\varphi^l \in \mathbf{R}_q^l$ and $\mathbf{G}$ is the generator in $\mathbf{R}_q^{m\times l}$. We use the sampling in lattices Samp() to incorporate the B_R in our secret. β is the maximum coefficient. High and Low refer to high and low bits due to signature compression in Dilithium (Ducas et al., 2018). $\|z\|$ is the norm operation on z. ∞ is the norm infinity.

Algorithm 4 PoUR Construction in Lattices and its usage in Dilithium Signature (Ducas et al., 2018)

```
$\underline{c \leftarrow \mathsf{PoUR.Gen}(1^\kappa):}$
1: Target value $T$ is determined
2: Let $(B_C, B_R) \in \{0,1\}^*$
3: return $c = (B_C, T, B_R)$

$\underline{\pi \leftarrow \mathsf{PoUR.Solve}(c = (B_C, T, B_R)):}$
1: $i = 0$
2: do
3:     $\mathbf{s}_i \xleftarrow{\$} \mathsf{S}_\varphi^l$, $\mathbf{r}_i \leftarrow \mathsf{Samp}(s_i, B_R)$, $\mathbf{R}_i \leftarrow \mathbf{G} \cdot \mathbf{r}_i$
4:     $\mathbf{e}_i \xleftarrow{\$} \mathsf{S}_\varphi^l$, $\mathcal{U} \leftarrow \mathcal{U} \cup (\mathbf{r}_i, \mathbf{e}_i)$, $i = i + 1$
5: while $H(B_C \,\|\, \mathbf{R}_{i-1}) \geq T$
6: return $\pi = (\mathbf{s}_{i-1}, \mathbf{R}_{i-1})$

$\underline{(0,1) \leftarrow \mathsf{PoUR.Ver}(c = (B_C, T, B_R), \pi = (\mathbf{s}, \mathbf{R})):}$
1: $\mathbf{x} \leftarrow \mathsf{Samp}(\mathbf{s}, B_R)$
2: return $\mathbf{G} \cdot \mathbf{x} = \mathbf{R}$ & $H(B_C \,\|\, \mathbf{R})) < T$

$\underline{(k, K) \leftarrow \mathsf{Dilithium.Kg}()}$
1: $(\mathbf{r}, \mathbf{e}) \leftarrow$ Pick and remove from $\mathcal{U}$
2: $\mathbf{t} = \mathbf{G} \cdot \mathbf{r} + \mathbf{e}$
3: return $(k = (\mathbf{r}, \mathbf{e}, \mathbf{G}, t)$, $K = (\mathbf{G}, \mathbf{t}))$

$\underline{\sigma \leftarrow \mathsf{Dilithium.Sig}(\mathbf{k}, M)}$
1: do
2:     $(\mathbf{r}, \mathbf{e}) \leftarrow$ pick from $\mathcal{U}$
3:     $w \leftarrow \mathbf{G} \cdot \mathbf{r}$, $c \leftarrow H(\mathsf{High}(w) \,\|\, M)$
4:     $\mathbf{z} \leftarrow \mathbf{y} + c \cdot \mathbf{r}$
5:     $\mathsf{b1} \leftarrow \|\mathbf{z}\|_\infty \geq \varphi_1 - \beta$
6:     $\mathsf{b2} \leftarrow \|\mathsf{Low}(\mathbf{w} - c \cdot \mathbf{e}), M)\|_\infty \geq \varphi_2 - \beta$
7: while b1 or b2
8: return $\sigma \leftarrow (\mathbf{z}, c)$

$\underline{(0,1) \leftarrow \mathsf{Dilithium.Ver}(\mathbf{K}, M, \sigma)}$
1: $a \leftarrow H(\mathsf{High}(\mathbf{G} \cdot \mathbf{z} - c \cdot \mathbf{t}) \,\|\, M)$
2: $b_1 \leftarrow \|\mathbf{z}\|_\infty < \varphi_1 - \beta$, $b_2 \leftarrow H(M \,\|\, a) = c$
3: if $b_1$ & $b_2$ then
4:     return 1
5: else return 0
```

5.2 Security Analysis of PoUR

We base our security claim on the fact that the disclosable commitments we save in PoUR operates identical as the nonce in the original hash-based PoW. The difference between the traditional hash-based PoW and our PoUR is that we replace the nonce in traditional hash-based PoW algorithm with the disclosable PAC component in our construction.

The secret component of PACs in PoUR is selected from a uniformly distributed set of numbers. Furthermore, the disclosable commitment portion is created as a AOWF of the secret. Thus, the discloable PAC component operates identical as the traditional nonce since the probability of selecting the same disclosable secret component is neglible by κ. Furthermore, we assume a collusion-free AOWF. The collusion-freeness property is preserved in Schnorr (Schnorr, 1991) variants or discrete log based problems. Even if the AOWF is not collusion-free, PoUR still preserve its security but can potentially cause a slight (negligible) performance drawback.

Remark: The security of SchnorrQ (Schnorr, 1991), Dilithium (Ducas et al., 2018), and, CryptoOp are presevered when instantiated with PoUR since the corresponding commitments (which are located in $\mathcal{U}$) with PoUR are identical to the original corresponding commitments.

Therefore, since our disclosable commitment component operates identical as the original nonce in the traditional consensus, PoUR is as secure as the traditional Bitcoin PoW.

5.3 Alternative Constructions of PoUR

We now discuss alternative constructions of PoUR.

Key-Exchange: PACs created during PoUR can be used as session keys during key-exchanges.

Signcryption: PACs created during PoUR can be used in signcryption when the public-keys are known prior to mining.

Zero-Knowledge Proofs: PoUR provides support for creating PACs for zero-knowledge proof constructions such as (Biasse et al.,) the same way it provides support with digital signatures.

Cryptographic Beacons: We could store the random beacon outputs during consensus with PoUR (Bünz et al., 2017).

Remark: Due to the generation of PACs, a single iteration in PoUR can take more time than a single iteration in a traditional PoW. However, this does not result in a loss in performance because it is possible to adjust the target value of the puzzle and use PoUR instead of PoW so that we do not give up on the hardness of the puzzle and also leverage a useful PoW.

6 Performance Analysis

We now present our evaluation metrics and experimental setup.

6.1 Evaluation Metrics and Experimental Setup

Hardware: We fully-implemented the PoUR on commodity hardware with an Intel i7-8750H 2.20 GHz processor on a 32GB of RAM.

Implementation: Our implementation is based on the Bitcoin infrastructure (Nakamoto, 2009). We replace the classical hash-based Bitcoin[1] PoW with PoUR. This allows for the creation of commitments during consensus to be used at a later time and thus, prevents energy and execution-time waste.

```
www.github.com/efeseyitoglu/PoUR
```

Open-Source Tools: We executed the open-source code of Dilithium[2] (Ducas et al., 2018) for generating post-quantum commitments. For elliptic-curve commitment savings, we measured SchnorrQ (Schnorr, 1991) implementation of the FourQlib library [3]. We chose Dilitium (Ducas et al., 2018) and SchnorrQ (Schnorr, 1991) due to their popularity and leveraged the FourQlibrary because it is a fast elliptic curve library. We obtained the energy savings and power consumption by executing Intel's powertop tool[4].

Performance: We calculate the performance efficiency in terms of execution and energy savings we acquire via PACs. Considering that a traditional commitment pair (e.g. (Schnorr, 1991)) would require around 32KB of storage, this is a small price to pay considering our efficiency results.

Methodology: Since it would be difficult to create large number of commitments (e.g. 2^{45}), we measured the execution time and power requirements for generating $2^5, 2^{10}, 2^{15}$ commitments and extrapolated with Matlab for larger number of commitments 2^{20}, 2^{25}, 2^{30}, 2^{35}.

[1]https://github.com/bitcoin/bitcoin
[2]https://github.com/pq-crystals/dilithium
[3]https://github.com/Microsoft/FourQlib
[4]https://github.com/fenrus75/powertop

6.2 Performance Evaluation

We focused on the savings that PoUR enables on execution time and energy consumption. J and W mean Joule and Watt. Our experiments showed a linear relation between the puzzle solving attempts and the energy/execution time savings.

Table 1: Experimental Results for Savings in Elliptic-Curve and Post-Quantum Settings

Puzzle Solving Attempts	**Execution Time Savings (ms)**	**Energy Savings (J)**	**Power Consumption Rate (W = J/Sec)**
Elliptic-Curve Commitments			
2^5	0.306	0.0065	
2^{10}	10.6	0.2248	21.2
2^{15}	314.74	6.6724	
Post-Quantum Commitments			
2^5	2.47	0.06	
2^{10}	84.60	2.12	25.2
2^{15}	2596.295	65.85	

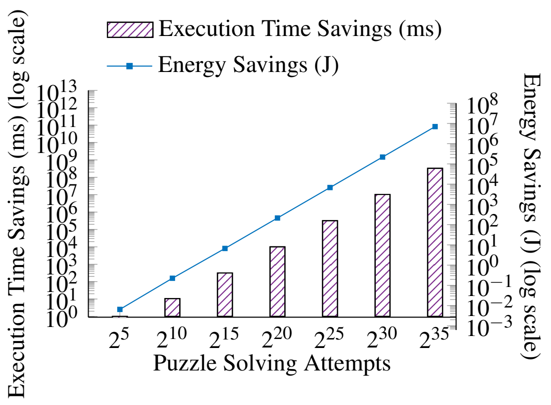

Figure 1: Execution Time (ms) and Energy (J) Savings for Elliptic-Curve Commitments

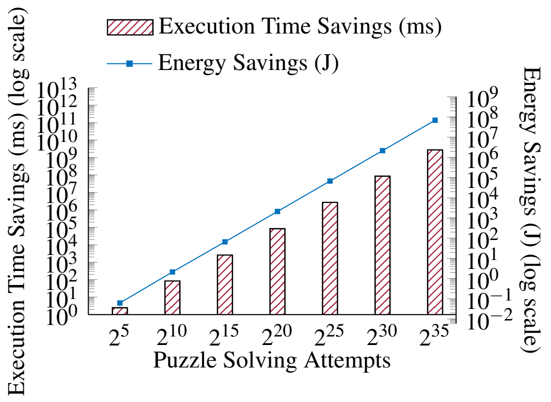

Figure 2: Execution Time (ms) and Energy (J) Savings for Post-Quantum Commitments

7 CONCLUSION

We propose PoUR: a PoW algorithm that requires minial alteration to the existing hash-based consensus. Our experiments showed that PoUR saves around $2 \cdot 10^6$ J in post-quantum and $2 \cdot 10^5$ J in elliptic-curve settings. We fully-implemented PoUR to demonstrate its' energy-saving capabilities. We also algorithmically showed how the PACs we save with PoUR could be used in a wide array of cryptographic schemes.

Acknowledgements: We thank the anonymous reviewers for their feedback. This work is supported by the NSF CAREER Award CNS-1917627.

REFERENCES

Bahri, L. and Girdzijauskas, S. (2018). When trust saves energy: a reference framework for proof of trust (pot) blockchains. In *Companion Proceedings of the The Web Conference 2018*, pages 1165–1169.

Ball, M., Rosen, A., Sabin, M., and Vasudevan, P. N. (2017). Proofs of useful work. Cryptology ePrint Archive, Report 2017/203. `https://eprint.iacr.org/2017/203`.

Ball, M., Rosen, A., Sabin, M., and Vasudevan, P. N. (2018). Proofs of work from worst-case assumptions. In *Annual International Cryptology Conference*, pages 789–819. Springer.

Behnia, R., Ozmen, M. O., and Yavuz, A. A. (2019). Aris: Authentication for real-time iot systems. *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6.

Bentov, I., Lee, C., Mizrahi, A., and Rosenfeld, M. (2014). Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y. *SIGMETRICS Perform. Eval. Rev.*, 42(3):34–37.

Bentov, I., Pass, R., and Shi, E. (2016). Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919.

Biasse, J.-F., Chellappan, S., Kariev, S., Khan, N., Menezes, L., Seyitoglu, E., Somboonwit, C., and Yavuz, A. Trace-σ.

Brown-Cohen, J., Narayanan, A., Psomas, A., and Matthew Weinberg, S. (2019). Formal barriers to longest-chain proof-of-stake protocols?

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

Catalano, D., Fiore, D., Gennaro, R., and Vamvourellis, K. (2013). Algebraic (trapdoor) one-way functions and their applications. In *Theory of Cryptography Conference*, pages 680–699. Springer.

Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., and Shi, W. (2017). On security analysis of proof-of-elapsed-time (poet).

Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., and Stehlé, D. (2018). Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268.

Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. (2016). On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16.

Hastings, M., Heninger, N., and Wustrow, E. (2019). Short paper: The proof is in the pudding. In *International Conference on Financial Cryptography and Data Security*, pages 396–404. Springer.

Katz, J. and Lindell, Y. (2014). *Introduction to modern cryptography*. CRC press.

Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, pages 357–388. Springer.

King, S. (2013). Primecoin: Cryptocurrency with prime number proof-of-work. *July 7th*, 1(6).

Król, M., Sonnino, A., Al-Bassam, M., Tasiopoulos, A., and Psaras, I. (2019). Proof-of-prestige: A useful work reward system for unverifiable tasks. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 293–301. IEEE.

Li, W., Andreina, S., Bohli, J.-M., and Karame, G. (2017). Securing proof-of-stake blockchain protocols. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 297–315. Springer.

Lihu, A., Du, J., Barjaktarevic, I., Gerzanics, P., and Harvilla, M. (2020). A proof of useful work for artificial intelligence on the blockchain.

Loe, A. F. and Quaglia, E. A. (2018). Conquering generals: an np-hard proof of useful work. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pages 54–59.

Meng, W., Wang, J., Wang, X., Liu, J., Yu, Z., Li, J., Zhao, Y., and Chow, S. S. M. (2018). Position paper on blockchain technology: Smart contract and applications. In Au, M. H., Yiu, S. M., Li, J., Luo, X., Wang, C., Castiglione, A., and Kluczniak, K., editors, *Network and System Security*, pages 474–483, Cham. Springer International Publishing.

Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system.

O'Dwyer, K. J. and Malone, D. (2014). Bitcoin mining and its energy footprint.

Peikert, C. (2016). A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424.

Schnorr, C. (1991). Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174.

Seyitoglu, E. U. A., Yavuz, A. A., and Ozmen, M. O. (2020). Compact and resilient cryptographic tools for digital forensics. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9.

Tromp, J. (2015). Cuckoo cycle: a memory bound graph-theoretic proof-of-work. In *International Conference on Financial Cryptography and Data Security*, pages 49–62. Springer.