Designing Proof of Transaction Puzzles for Cryptocurrency

Taotao Li[1,3], Parhat Abla[1,2]and Mingsheng Wang[1], Qianwen Wei[1,2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
parhat,wangmingsheng,weiqianwen@iie.ac.cn
[2] University of Chinese Academy Science, Beijing 100049, China
[3] Human University of Science and Technology, Xiangtan 411201, China
taotaolittl@outlook.com

Abstract. One of the Bitcoin's innovations is the Proof of Work puzzle (aka scratch-off puzzle) as a consensus protocol for anonymous networks without pre-established PKI. Bitcoins based on the Proof of Work puzzle have been harshly blamed today for problems such as energy wasted and not easily scalable. In this paper, we construct a novel Proof of Transaction(PoT) puzzle, and prove that PoT puzzle satisfies the basic construction conditions of scratch-off puzzle. We also show construction of PoTcoin as application. To reduce the network load we use sequential aggregate signature. PoTcoin has many advantage but not limited as strengthening the network topology, promoting currency circulation, anti-outsourcing computing and environment-friendly.

Keywords: proof of transaction, sequential aggregate signature, blockchain

1 Introduction

Since 2008, Bitcoin[Nak08] becomes the most popular cryptocurrency in the world. The most attractive part is a decentralized, distributed consensus mechanism (aka. Nakamoto consensus), that enables all participants in a peer-to-peer(P2P) network to consensus on a distributed public ledger (blockchain). In other words, the consensus is that thousands of independent nodes follow the simple rules that follows a spontaneously,asynchronous interactions.All Bitcoin attributes, including currency, transactions, payments, and security models that do not rely on central agencies and trust, are all derivatives of this mechanism.

Nakamoto consensus mechanism is as follows: In a Bitcoin network, a node (aka. miner) initiates a transaction and broadcasts it to the network. The other nodes receive the transaction and validate the transaction according to the verification algorithm. The transaction that satisfies the verification is temporarily saved to the node's transaction pool and is again broadcast to other nodes. Miners compete with others to solve a puzzle for obtaining the opportunity of adding confirmed transactions to Bitcoin's public ledger of past transactions. Then, the solution will be broadcasted to other miners in the P2P network. If

the solution been verified, the miners will add the new block to the corresponding blockchain and continue to mine the new block. At the same time, the lucky miner(who firstly find the solution to the puzzle)will get corresponding reward and transaction fee. The mining process is a concurrently process, it may be the case that conflicting versions of blockchain, called branches or forks. Due to the Bitcoin consensus mechanism [Nak08], miners solved the forking problem by mining on the longest chain. Instead, the blocks on the shorter chain become orphaned blocks, and all transactions in the orphaned blocks are returned to the transaction pool for recertification.

In the consensus mechanism, the most important part is the Scratch off Puzzle(SoP). The SoP used by Bitcoin is based on moderate hard computational puzzle[Bac02][DN92], known as Proof of work(PoW) puzzle. Essence of PoW puzzle is to solve the inequality, the miners keeps doing hash calculation to find a randomness which satisfies the special inequality. Now, there are several challenges for PoW puzzle. Firstly, unequal mining probability. Some oligarch use their own mining resources or mining strategy, making their possession of scarcity of resources (ie, physical resources) account for most of the entire network of resources. Eventually leading to the probability of common miners successful mining block is getting smaller and smaller. Indeed, over the past few years, the computation power of large mine pool have exceeded the entire network by one-third over several times [Vit13]. For example, GHash.io[GHa14] computation power has more than half the amount of power in the entire network. Now, large mine pools (F2Pool, AntPool, BTCC and BW) are all located in China. Their total computing power reaches 60% of the whole network. Second, security is severely challenged. Bitcoin's security relies on the assumption that honest miner control most of the power in the network. In other words, if the attacker controls more than 50% of the network's total power, the network is not secure (for example, there exist double spending attack). Then, the above assumption has been seriously questioned. Eyal and Sirer[ES13] state that when a selfish mine pool owned more than 25% of the total network, and under the influence of selfish mining strategies[LK16][SSZ16]and economies of scale[Moo59], it attracted more Miners are constantly adding to the pool, making the computation power of the mine pool has been more than half the power of the entire network or even more. Eventually, the Bitcoin system is no longer safe to decentralized system. Third, energy waste. In Bitcoin, in order to mine new blocks, miners run PoW algorithms, constantly doing hash calculations, thus causing a lot of unnecessary computational effort.

As mentioned above, a good SoP is crucial to the consensus mechanism and the extensibility of the entire blockchain. In this paper, we propose new a SoP - Proof of Transaction(PoT) puzzle. Our PoT puzzle is based on sequential aggregate signatures[LMRS04]. sequential aggregate signature is a very interesting model, the source of applying this model is as follows: ① Consideration of the transaction data in the blockchain. Obviously, a lot of data in the blockchain is the transaction. In addition to the currency transfer function, these transaction data have other attributes such as unforgeability, authenticity, traceability, etc.

How can we use these data functions and property once again perfect use? ② Who will generate the next block? Needless to say, the next block is generated by Bitcoin miner in Bitcoin. In the PPcoin[KN12], the miner with large amount of coins generates the next block. While in Ethereum [But14], the next block is produced by miners who have both large computation power and lots of coins. These computation power and coins can then easily be increased by external resources (such as nation state power), seriously affecting the security of the blockchain and increasing the instability of the blockchain. Based on the above two ideas, we used to sequential aggregate signature model. We use the transaction data in the blockchain so that those users who have initiated those transactions generate the next block. In other words, we give the mining right to those who really use the blockchain users, so that these users really use the blockchain system as a "master". Unlike other cryptocurrencies, as computational power and coins increase, nodes will dominate the block generation. On the contrary, in the system we constructed, the number of transactions is not easily increased in a short period of time. For the details of the sequential aggregate signature, see section 3.

We construct proof of transaction puzzle using sequential aggregate signatures, and we show that proof of transaction puzzle satisfy the basic conditions for constructing Scratch-off puzzle[MKKS15]. Based on proof of transaction puzzle, we designed PoTcoin that have good performance, such as strengthening the network topology and facilitating the circulate of PoTcoin.

The contribution of this article is as follows:

- We successfully constructed a new Proof of Transcation puzzle using sequential aggregate signature and proved that the basic conditions for constructing Scratch-off puzzle are satisfied.
- We accelerate signature verification and reduce network load by using sequential aggregate signatures.
- Application for proof of transaction puzzle-PoTcoin. PoTcoin has the good performance of strengthening the network topology, promoting the circulate of PoTcoin, resistance to outsourcing computation and environment-friendly.

1.1 Related Work

As mentioned above, Bitcoin PoW puzzles have been severely challenged in performance. In order to solve these problems, a large number of researchers have proposed many new Scratch-off puzzles. These puzzles can be classified in three main classes.

Scratch-off puzzle based on physical resources. Which is divided into three categories, the first category of puzzles are based on the computation of the hard puzzle. Mining depends on the CPU computability. Bitcoin is the use of such computability continue to do SHA-256 calculations. Miller et al.[MJSP14] found that it takes approximately 2^{55} hashes to mine a new block(which is equivalent to the amount of work required to crack a DES password), resulting in a waste of computing and natural resources. And solving this puzzle has no real value

to society. Therefore, Scratch-off puzzles are considered based on useful computational puzzle (eg, protein folding problems[Wol05]). In 2013, King [Kin13] proposed Primecoin, using the huge computational power of the whole network to find prime numbers, the disadvantage of which is that the complete proof of security is ungiven. In 2014, Miller [MJSP14] proposed a Scratch-off puzzle based on Proofs-of-Retrievability (PoR) [JK07]. The main feature of this mechanism is the use of Bitcoin mining resources for distributed storage of archives, reducing the overall waste of Bitcoin. The main drawback is that it takes longer time to verify the results of the puzzle. The second type of scratch-off puzzle is based on storage problems. The capability of mining new block is dependents on miners storagability. Dziembowski [DFKP15] proposed Proof-of-Space (PoS) and later PoS was improved [PPA$^+$15][RD16][Dzi13]. The third type of scratch-off puzzle is based on the CHPTCHA problem[ABHL03]. This type of scratch-off puzzle can make miners more equally during successful mining a new blocks. Blocki's Proof-of-Human (PoH)[BZ16] puzzle is based on CHPTCHA[ABHL03] and indistinguishability obfuscation (IO)[GGH$^+$13] and uses human-machine interaction to solve artificial intelligence problems (for example, reading distorted letters). The solution to the problem must rely on human participation, making the probability of each miners successfully mining new blocks more equally. However, the development achievements of IO [GGH$^+$13] can not meet the demand of mechanism of PoH, therefore the PoH is unable to practical implement.

Scratch-off puzzle based on virtual resources. The main advantage of this type of Scratch-off puzzle is the reduction of consumption and the transfer of physical resources needed for mining to virtual resources. For example, the Nextcoin [Com16] which based on the Proof-of-Stake (PoS) [Kwo][But16], the one who owns the large amount of coins during the mining process, decide to produce the next block. At the same time, it also brought a significant flaw, the centralization of the coin break the decentralization of the system. In other words, with the more coin the miners, the greater the probability of mining new blocks, resulting in a few number of the miners who have most of the coins, more and more easy to mine the new block, making the system centralized.

Hybrid scratch-off puzzle based on physical resource and virtual resource. This type of scratch-off puzzle increases security and increases the cost of attacks. Duong[DFZ16] and Bentov[Ben] show that the advantage of this mechanism is that honest nodes still have the chance to use stakes to prevent the blockchain even if the malicious nodes have more than 50% computability. Typical cryptocurrencies are PPcoin [KN12], Ethereum [But14], TwinsCoin[CDFZ17].

Section 2 gives the basic knowledge and sequential aggregate signature are given in section 3, then we define the PoT puzzle and show its security. In section 5 we give application of the PoT puzzle, and its advantage and we conclude the whole article in last section.

2 Preliminaries

2.1 Assumption

PoT protocol is based on the Bitcoin protocol. In the PoT protocol, we assume that the resources (the number of transactions that have been initiated and recorded in the blockchain, hereinafter referred to as the number of transactions) owned by each PoT user are equal, and we also assume that the number of users in the PoT protocol is n, denoted as u_i ($i \in [1, n]$), where the number of online users is m, where $m \leq n$. Note that this is an "ideal assumption". In reality, each different user u_i has a different number of transactions. However, this ideal assumption is not loss of generality, because in reality user u_i is a combination of arbitrary users under ideal assumptions. We pointed out that in the protocol, the number of users who really participate in the operation of the protocol can not be determined. That is, we can not identify the number of users in this protocol that are participating in the operational protocol. In short, this is a static model under our assumption that the number of users is fixed while running the protocol.

2.2 Scratch-off puzzle

As mentioned in the introduction, the Bitcoin protocol is based on a computationally moderate puzzle that all miners compete with each other to solve it. However, it is common to call Bitcoin's puzzle as proof of work puzzle, and the basic requirements for building such a puzzle are somewhat different[CMSW09] [DN92][GW14][SKR+11]. Miller et al.[MJSP14][MKKS15] shows some requirements that a Bitcoin puzzle(aka scratch-off puzzle) should satisfies.The following gives Miller[MKKS15] for the definition of scratch-off puzzle and a scratch-off puzzle must meet the three requirements.

In what follows, let λ denote a security parameter. A scratch-off puzzle is parameterized by parameters$(\underline{t}, \mu, d, t_0)$ where, informally speaking, $\underline{t}$ denotes the amount of work needed to attempt a single puzzle solution, μ refers to the maximum amount by which an adversary can speed up the process of finding solutions, d affects the average number of attempts to find a solution, and t_0 denotes the initialiazation overhead of the algorithm.

Definition 1. *A scratch-off puzzle is parameterized by parameters $(\underline{t}, \mu, d, t_0)$, and consists of the following algorithms (satisfying properties explained shortly):*

1) $\mathcal{G}(1^\lambda) \rightarrow$ *puz: generates a puzzle instance.*
2) *Work(puz,m,t)* $\rightarrow$ *ticket: The Work algorithm takes a puzzle instance puz, some payload m, and time parameter t. It makes t unit scratch attempts, using $t \cdot \underline{t} + t_0$ time steps in total. Here $\underline{t} = ploy(\lambda)$ is the unit scratch time, and t_0 can be thought of as the initialization and finalization cost of Work.*
3) *Verify(puz,m,ticket)* $\rightarrow \{0,1\}$*: checks if a ticket is valid for a specific instance puz, and payload m. If ticket passes this check, we refer to it as a winning ticket for (puz,m).*

Intuitively, the honest Work algorithm makes t unit scratch attempts, and each attempt has probability 2^{-d} of finding a winning ticket, where d is called the puzzles difficulty parameter. For simplicity, we will henceforth use the notation $\zeta(t,d) := 1-(1-2^{-d})^t$ to refer to the probability of finding a winning ticket using t scratch attempts. For technical reasons that will become apparent later, we additionally define the shorthand $\zeta^+(t,d) = \zeta(t+1,d)$.

A scratch-off puzzle must satisfy three requirements:

1) Correctness. For any (puz,m,t), if Work(puz,m,t) outputs ticket $\neq \perp$, then Verify(puz,m,ticket) = 1.

2) Feasibility and parallelizability. Solving a scratch-off puzzle is feasible, and can be parallelized. More formally, for any $\ell = poly(\lambda)$, for any $t_1, t_2, ..., t_\ell = ploy(\lambda)$, let $t := \sum_{i\in[\ell]} t_i$.

$$Pr\begin{bmatrix} puz \leftarrow \mathcal{G}(1^\lambda), \\ m \leftarrow \{0,1\}^\lambda, \\ \forall i \in [\ell] : ticket_i \leftarrow Work(puz, m, t_i), \\ \forall i \in [\ell] : Verify(puz; m; ticket_i) \end{bmatrix} \geq \zeta(t) - negl(\lambda)$$

Intuitively, each unit scratch attempt, taking time $\underline{t}$, has probability 2^{-d} of finding a winning ticket. Therefore, if ℓ potentially parallel processes each makes $t_1, t_2, ..., t_\ell$ attempts, the probability of finding one winning ticket overall is $\zeta(t) \pm negl(\lambda)$ where $t = \sum_{i\in[\ell]} t_i$.

3) μ-Incompressibility. Roughly speaking, the work for solving a puzzle must be incompressible in the sense that even the best adversary can speed up the finding of a puzzle solution by at most a factor of μ.More formally, a scratchoff puzzle is μ-incompressible (where $\mu \geq 1$) if for any probabilistic poly-nomial-time adversary $\mathcal{A}$ taking at most $t \cdot \underline{t}$ steps,

$$Pr\begin{bmatrix} puz \leftarrow (1^\lambda), \\ (m, ticket) \leftarrow \mathcal{A}(puz) : \\ Verify(puz, m, ticket) = 1 \end{bmatrix} \leq \zeta^+(\mu t) \pm negl(\lambda)$$

Note that $\zeta^+(t) = 1-(1-2^{-d})^{t+1}$ is roughly the probability of outputting a winning ticket after t unit scratch attempts, though we additionally allow the adversary to make a final guess at the end (as in [SKR+11]), and hence the t+1 in the exponent instead of just t. Ideally, we would like the compressibility factor μ to be as close to 1 as possible. When μ =1, the honest Work algorithm is the optimal way to solve a puzzle.

3 Sequential Aggregate Signature Scheme

Aggregate signature[BGLS03](based on pairing) is a generalization of multi-signature in which several users sign on distinct messages. In aggregate signature

those signatures are generated by individuals and aggregate them. Note that the aggregating party may malicious. A sequential aggregate signature(SAS)[LMRS04] is very same as aggregate signature but the every signer will sign the message by some order. In a SAS scheme the signer may take the secret key and a message to be signed plus a SAS signature so far as input and output a SAS signature. Note that every signer will sign the message and aggregate then too. After all the SAS signature should be valid corresponding to all the signers public keys.

In this paper our purpose is to construct a secure proof of puzzle, So we didn't go any further. The following definition and the security experiment are very similar to [LOS+13] and the definition is as follows.

Definition 2. *An aggregate signature is consisted of three PPT algorithms: (KeyGen,AggregateSign, AggregateVerify) such that:*

KeyGen(1^n) *input a security parameter* 1^n *and output public-secret key pair (pk,sk).*

AggregateSign$(\sigma_k, ((m_1, pk_1), \cdots, (m_k, pk_k)))$ *inputs an aggregate signature* σ_k *and k tuple of message-public key pairs* $((m_1, pk_1), \cdots, (m_k, pk_k))$ *. Outputs an aggregate signature* σ_{k+1} *and k+1 tuple of message-public key pairs* $((m_1, pk_1), \cdots, (m_{k+1}, pk_{k+1}))$

AggregateVerify$(\sigma_n, ((m_1, pk_1), \cdots, (m_n, pk_n)))$ *Checks aggregate signature against the all messages and returns a boolean bit b. b=1 means* σ_n *match all the messages, b=0 implies* σ_n *is a invalid signature.*

A trivial sequential aggregate signature scheme can be constructed from ordinary signature scheme by putting all the signatures together. Namely, suppose $(Keygen, Sign, Verify)$ is a ordinary signature scheme , then we can obtain a sequential signature scheme by letting $AggregateSign(M_i, \mathrm{M}, sk_i, \sigma_i) = ((\mathbf{M}, m_i), (\sigma_i, \sigma))$, and $AggregateVerify$ on the fly, where $\sigma = Sign(m_i, sk_i)$.

the security of sequential aggregate signature schemes(SAS) is defined as the nonexistence of an adversary capable, within the restrict of a certain game, of existentially forging a sequential aggregate signature. Existential forgery here means that the adversary attempts to forge a sequential aggregate signature, on messages of his choice, by some set of users not all of whose private keys are known to the forger. We formalize this intuition as the sequential aggregate chosen-key security model. In this model, the adversary $\mathcal{A}$ is given a single public key. His goal is the existential forgery of a sequential aggregate signature. We give the adversary power to choose all public keys except the challenge public key. The adversary is also given access to a sequential aggregate signing oracle on the challenge key. His advantage, $Adv^{\mathcal{A}}_{AggSig}$, is defined to be his probability of success in the following experiment between a challenger and a PPT adversary $\mathcal{A}$:

Setup choose $(pk, sk) = Keygen(1^n)$, and give pk to $\mathcal{A}$ as a challenge.

Certification Query $\mathcal{A}$ provides key pairs (pk', sk') to C for certifying his public key pk'. $C = C, pk'$, if sk' is matching pk'.

Signature Query $\mathcal{A}$ can query a sequential aggregate signature under the challenge public key pk, on a message M of his own choice. furthermore, $\mathcal{A}$ provide an aggregate signature σ' so far on message vector $\mathbf{M}$ under public key $\mathbf{pk}$. Challenger checks that the validity of σ'; that $pk \notin \mathbf{pk}$; that $|\mathbf{pk}| < n$ (n is upper bound on the length of sequential signature); that $\mathbf{pk} \subset C$. If any of them fails the return $\perp$, otherwise respond with $\sigma = AggregateSign(sk, M, \sigma', \mathbf{M}, \mathbf{pk})$.

Output After polynomially many time querying the AggregateSign() algorithm, $\mathcal{A}$ outputs a forgery $(\sigma^*, \mathbf{M}, \mathbf{pk})$ and this forgery must be valid under AggregateVerify(); $pk \in \mathbf{pk}$ and $\mathbf{pk} \backslash \{pk\} \subset C$; $|\mathbf{pk}| \leq n$;

We will denote the advantage of adversary successes in the above game by $AggSignForge_{\mathcal{A}}^{SAS}$ and upper bound on the length of sequential aggregate signature by a positive integer n. ϵ and t positive reals , and q_C, q_S are polynomials in security parameter. The security of sequential aggregate signature scheme is given below:

Definition 3. *A sequential aggregate signature scheme is* $(t, q_C, q_S, n, \epsilon)-$*secure if there not exists a t-time adversary making* q_C *certification queries and making* q_S *queries to Signing algorithm and win the above game with advantage more than* ϵ*,that:*

$$Prob[AggSignForge_{\mathcal{A}}^{SAS}(n) = 1] \leq \epsilon.$$

The probability is taken over the randomness used in the experiment and adversary.

secure SAS schemes can be constructed permutations[LMRS04]. There is lattice based SAS scheme[BB14] which is secure in the random oracle model[BR93]. It can be constructed by any collection of preimage sampleable trapdoor functions like [GPV08] or more efficient one[MP12]. When we say SAS scheme we mean by that a secure SAS scheme. We will use a simplified sequential signature scheme in our PoT puzzle, that all the messages which will be signed are same, that $m_1 = m_2 = \cdots = m_n$ in the definition 2. So we have a very short message-signature pair which would be diffused to the peer to peer network, and it essentially reduced the network load. So, when we are reducing the computation waste, we didn't increase network load. In the next section we will describe a new scratch-off puzzle using sequential aggregate signature scheme in detail and prove its security.

4 Proof of Transaction Puzzle

In this section, we define the syntax and security of the proof of transaction puzzle and use the sequential aggregate signature to illustrate the structure of the proof of transaction puzzle.

4.1 Definition

In the proof of work puzzle, all the nodes in the entire network compete with each other to solve the puzzle in each a epoch. The node that first provides the correct answer indicates that it effectively solves the puzzle and obtains the block reward. The proof of work puzzle is composed of a set of algorithms: setup algorithm Setup(), puzzle instance generation algorithm G(), puzzle solution algorithm C() and verification algorithm V(). In the setting algorithm, it is mainly used to design public parameters. In the puzzle instance generation algorithm, it mainly uses the parameters and data of the setting algorithm to generate a puzzle instance. In the puzzle solving algorithm, the node keeps doing SHA-256 computation and tries to find the answer. It is worth noting that this is a non-deterministic algorithm. Because each time a user try an answer, the user do not know whether this answer will solve the puzzle; in the verification algorithm, the node verifies the answers to the puzzle received in the network. Note that this is a deterministic algorithm. Because each node receives the puzzle and answers, through a verification computation the node can verify the correctness of the answer. In order to reach a consensus, the proof of work puzzle must meet the basic conditions for constructing Scratch-off puzzled.

Our proof of transaction puzzle and proof of work puzzle are similar, but the main differences are as follows: (1) Unequal mining resources. In the proof of work puzzle, the ability of miners to mining new block is proportional to the computation power. The greater the calculated power of miners, the greater the probability of mining new block. In proof of transaction puzzle, the probability of users mining a new block is proportional to the number of transactions they own. The more transactions they have, the greater the probability of mining new block. (2) The form of the puzzle is different. In the proof of work puzzle, the miners solve an inequality problem; in the proof of transaction puzzle , the user is solving an equation problem; (3) The way to solve the puzzle are different. In the proof of work puzzle, the miners constantly change the random numbers so that the hash values of the random numbers and the public initial parameters are less than the difficulty values. In proof of transaction puzzle , the user needs to find a chain of sequential signatures whose length is equal to the difficulty value. the syntax is as follow:

Definition 4 (Proof of Transaction Puzzle). *The proof of transaction puzzle consists of a set of algorithms (Setup, G, $u^{\mathcal{O}(\cdot)}$, V) as follows:*

Setup: *Setup is a system random setting algorithm that inputs the parameter 1^λ (λ is a security parameter) and outputs a system common parameter $PP \leftarrow Setup(1^\lambda)$, which includes a puzzle size parameter is $\omega = ploy(\lambda)$.*

G: *G is a probabilistic puzzle generation algorithm, input the common parameter PP, and output the puzzle instance $puz = \sum_{i\in[1,\omega]} puz_i \leftarrow G(PP)$.*

$u^{\mathcal{O}(\cdot)}$: *$u^{\mathcal{O}(\cdot)}$ is a puzzle-solving algorithm that outputs a answer $\sigma \leftarrow u^{\mathcal{O}(\cdot)}(PP, puz)$ of length ω where $\mathcal{O}(\cdot)$ is a signing oracle which input an online transaction [4], output the signature of the owner of the transaction[5].*

V: *V is a deterministic puzzle verification algorithm that inputs public parameters PP and a pair of puzzle-answer (puz, σ) and outputs a bit $b := V(puz, \sigma, PP)$, which also includes signature verification. $b = 1$ means that σ is the valid signature of the puzzle puz, otherwise $b = 0$.*

We require Setup, G, $u^{\mathcal{O}(\cdot)}$ are a probabilistic polynomial time algorithm, Verify is a deterministic polynomial time algorithm.

Following notation of Miller et at.[MKKS15] we will let $\epsilon(k, \omega) = 1 - (1 - m^{-\omega})^k$, where m represents the number of online users in the network and m^{-1} represents probability of a user obtaining a valid signature after calling signing oracle once. In simple terms, $\epsilon(k, \omega)$ represents the probability of the user calling k times of signing oracle to get a valid answer to the puzzle.

Definition 5 (Honest User Solvability). *If a proof of transaction puzzle system (Setup, G, $u^{\mathcal{O}(\cdot)}$, V) to be honest user-solvable for each polynomial $k = ploy(\lambda)$ honest user $u^{\mathcal{O}(\cdot)}$ who controls k work unit, the success probability of user is expressed as follows:*

$$Prob\begin{bmatrix} PP \leftarrow Setup(1^\lambda); \\ puz^\star \leftarrow G(PP)); \\ \sigma^\star \leftarrow u^{\mathcal{O}(\cdot)}(PP, puz^\star); \\ V(PP, puz^\star, \sigma^\star) = 1; \end{bmatrix} \geq \epsilon(k, \omega) - negl(\lambda)$$

Definition 6 (Adversarial User Unsolvability). *If a proof of transaction puzzle system (Setup, G, $u^{\mathcal{O}(\cdot)}$, V) to be adversarial user unsolvability for each polynomial $k = ploy(\lambda)$ adversary $\mathcal{A}$ who controls at most k work unit, the success probability of this adversary $\mathcal{A}$ is expressed as follows:*

$$Prob\begin{bmatrix} PP \leftarrow Setup(1^\lambda); \\ puz^\star \leftarrow G(PP)); \\ \sigma^\star \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(PP, puz^\star); \\ V(PP, puz^\star, \sigma^\star) = 1; \end{bmatrix} \leq \epsilon(k+1, \omega) + negl(\lambda)$$

Remark 1) in the proof of transaction system, the adversary can try to guess the signature value he needs without calling the signing oracle, just as in[SKR$^+$11], after calling the signing oracle k times, allowing the adversary to guess the

[4] This transaction is in the longest block chain

[5] Note that this signature is for public parameters and the data which the user received from former user

signature value. Therefore, the probability that $\epsilon(k+1,\omega) = 1-(1-m^{-\omega})^{k+1}$ is approximately equal to probability of calling k times the signing oracle and then obtain a valid answer. 2) Like Miller's incompressibility factor μ in[MKKS15], we have $\mu = 1$ here. This is because our proof of transaction puzzle system is optimal. As in Definition 3, honest users need at least k work units to solve the puzzle.

4.2 Structure

In this part, we will present the structure of proof of transaction puzzle on Bitcoin system. In Bitcoin system, the proof of work puzzle generates a puzzle instance $puz \leftarrow G(s)$ with the most recent public parameter s, and the miners try each of them to a different randomness by constantly calling the random oracle(e.g.,the SHA256 hash function). The random number x makes the hash value of public parameter s and random number x less than the difficulty value. Namely, every random number x selected by miners compute $y_i = RO(s, x_i)$. If $y_i < T_\omega$, the corresponding x_i is regarded as the answer to the proof of work. Given a random oracle RO: $\{0,1\}^\star \rightarrow \{0,1\}^n$ we will use the notation $T_\omega = 2^{n-\omega}$. Intutitively, this ensures that $RO(s, x_i) < T_\omega$ with probability is $2^{-\omega}$.

In our proof of transaction, we first give a security parameter λ, where $\omega = ploy(\lambda)$ indicates the difficulty of the puzzle puz. The proof of transaction system generates the puzzle instance puz $\leftarrow G(s)$ with the latest public parameter s, where s represents the hash of the block header of the packed block constructed by the user, which contains the hash of the block header of the previous block, and transaction information in the packed block. In order to solve the proof of transaction puzzle, the user needs to call the signature oracle $u^{\mathcal{O}(\cdot)}$ to obtain the signature of the next user, so that the length of a sequential signature chain is equal to the difficulty value. First, each user u_i computes $\sigma_i = Hash(s)$, modifies σ_i off the height H of the current blockchain to obtain a certain block $H_i = \sigma_i$ mod H. The purpose of which is to randomly select a block from the blockchain; Then compute $Tx_i = \sigma_i mod \phi(H_i)$ and broadcast Tx_i to the network. Here the Tx_i is a transaction in block H_i, the purpose of this step is to randomly selected a transaction from the block H_i. That is to say, the whole process is equivalent to each user randomly select a transaction from the blockchain. Secondly, the user who received the transaction verify that whether he is the owner(who launched the transaction) of this transaction. If the user(u_{i+1}) is the owner of this transaction, the user u_{i+1} use his private key which used to sign on the transaction Tx_i, to sign on s and σ_i and get a signature $\sigma'_{i+1} = Sign(\sigma_i, s)$. Further, do a Hash on σ'_{i+1} to get σ_{i+1}, modulo σ_{i+1} by height H of current block chain and get a block of $H_{i+1} = \sigma_{i+1}$ mod H. Then compute $Tx_{i+1} = \sigma_{i+1} mod \phi(H_{i+1})$ and broadcast Tx_{i+1} to the network. Similarly, the user who received the transaction Tx_{i+1} and verifies herself as the owner of the transaction Tx_{i+1}, if he is the owner(u_{i+2}) of the transaction . The user u_{i+2} use his private key that has been signed for this transaction Tx_{i+1} and sign on s and σ_{i+1} and get $\sigma_{i+2} = Sign(\sigma_{i+1}, s)$. And so on, a series of signature values $(\sigma_{i+3}, \sigma_{i+4}, ..., \sigma_{i+k})$ will be obtained; Finally, the user calculates length

τ of the series of signature values $(\sigma_{i+1}, \sigma_{i+2}, ..., \sigma_{i+k}\ )$. If $\tau = k$, it means that the user successfully finds the answer $Ticket := \{s, [\sigma_i, \sigma_{i+1}, \sigma_{i+2}, ..., \sigma_{i+k}]\}$ to the puzzle puz. It is noteworthy that in order to reduce the load on the network, we aggregate the signatures $(\sigma_{i+1}, \sigma_{i+2}, ..., \sigma_{i+k})$ into a short signature σ using a sequential aggregate signature. The validation phase is to compute the validity of $(puz, Ticket, \omega)$. If the result of the calculation is 1, then the puzzle is solved successfully. Otherwise, the result is 0.

Structure Our proof of transaction puzzle structure consists of the following three phases (Setup, Scratch-off, Verify).

Setup $s \leftarrow Setup(1^\lambda)$. λ is a security parameter, $\omega = ploy(\lambda)$ indicates the difficulty of puzzle puz. s is a parameter used to generate the puzzle $puz \leftarrow G(s)$, which indicates the header hash value of the packed block constructed by the user itself.

Scratch-off puzzle $puz := \sum_{i\in[1,\omega]} puz_i$ is solved during the Scratch-off phase. Each user can generate the parameter s, and then compute Hash value of s and export the next user. After that, every user needs to call the signature oracle to get the next user's signature.

procedure PoT(s) ▷ s is block information
  $\sigma_1 \leftarrow Hash(s)$
  $H_1 = \sigma_1 \text{mod H}$
  $Tx_1 = \sigma_1 \text{ mod } \phi(H_1)$
  $u_1 \leftarrow Tx_1$
  for $i = 1 \dots k$ **do**
    $\sigma'_{i+1} \leftarrow u_i^{\mathcal{O}(\cdot)}(\sigma||s)$
    $\sigma_{i+1} = Hash(\sigma'_{i+1})$
    $H_{i+1} = \sigma_{i+1} \text{mod H}$
    $Tx_{i+1} = \sigma_{i+1} \text{mod} \phi(H_{i+1})$
    $u_{i+1} \leftarrow Tx_{i+1}$
  end for
  return (m, σ) ▷ a sequential aggregate signature on m
end procedure

The answer Ticket is defined as follows:

$$Ticket := \{s, [\sigma_1, \sigma_2, , ..., \sigma_{k+1}]\}$$

Verify Calculate $b := V(puz, Ticket, \omega)$. If b = 1, then Ticket is the correct answer to puz. That is, $\tau = \omega$. Otherwise, b = 0. The verification is necessary to repeat the Scratch-off process to verify that each Scratch-off is performed correctly.

To prove the security of the proof of transaction, we mimic Blocki's proof idea in[BZ16]. If the Definition 3 is true in the sequential aggregate signature, we can easily verify that the proof of transaction puzzle is honest user solvable. Next, we give the security theorem of our proof of transaction puzzle.

Theorem 1. *If the sequential aggregate signature used in the PoT puzzle is secure and Hash is random oracle, then PoT puzzle is adversarial user unsolvable.*

Proof. Suppose there exists an adversary $\mathcal{A}$ which controls m work-unit and the success probability in definition 6 is greater than $\epsilon(m+1) + negl(\lambda)$, then we can construct another adversary $\mathcal{A}'$ who can (controls all secret keys except the challenged one) success in the SAS security game with noticeably.

Let q_H be the number of queries that $\mathcal{A}$ make to random oracle. Without lose of generality, we assume that the adversary $\mathcal{A}$ query an input on the random oracle only once. $\mathcal{A}'$ works as follows:

Algorithm $\mathcal{A}'$

The algorithm input a $(pk, publicparam)$

1 Choose uniform $j \in \{1, ..., q_H\}$.

2 Run $\mathcal{A}$ in input $publicparam$,

3 When $\mathcal{A}$ makes ith random oracle query $Hash(\sigma_i)$, answer it as follows:
- If $i = j$, choose any feasible number r such that $pk = (r \bmod H) \bmod \phi(H_i)$, and return it to $\mathcal{A}$ as a response.
- If $i \neq j$, choose a random number r, and return r as a respond to the query.

4 When $\mathcal{A}$ makes SAS query on $(m, \mathbf{pk}, \sigma_{i-1})$ for pk_i, answers the query as follows:
- if $i = j$, query $(m, \mathbf{pk}, \sigma_{i-1})$ to SAS scheme and obtain σ_i.
- if $i \neq j$, produce a SAS signature σ_i by his own.
- return the SAS signature σ_i.

Now we define two games in the view of adversary $\mathcal{A}$.

Game0: This is the real game. In this game $\mathcal{A}$ will communicate with a challenger who works as same as in the Procedure $PoT(s)$. Namely, the challenger take a SAS signature σ_{i-1} so far as an input, then outputs a SAS signature σ_i corresponding to pk_i; taking random oracle query and responses with randomly. The adversary may query the random oracle and does two modulo computation to decide the next signer's public key. The $\mathcal{A}$ will query the random oracle at most q_H times and query on signing oracle at most m times, then it outputs a valid SAS signature of length (m+1).

Game1: In this game $\mathcal{A}$ will communicate with algorithm $\mathcal{A}'$. All the random oracle and signing queries are responds by $\mathcal{A}'$ as above. $\mathcal{A}$ may make q_H times random oracle query and m times SAS queries. After all it will outputs a SAS signature of length m+1 .

Lemma 1. ***Game0*** *and* ***Game1*** *are computationally indistinguishable.*

Proof. Note that in **Game0**, the way to decide next signer in the SAS chain is random due to the randomness returned by random oracle and two modulo functions. In **Game1** , all the queries are similar to **Game0** but ith random oracle query. Because the challenge public key is generated randomly, then the next signer decision procedure too. The SAS queries in these two games are completely indifferent. In the view of adversary $\mathcal{A}$, these two games are indistinguishable. □

Lemma 2. *If $\mathcal{A}$ wins in **Game1** with noticeably then $\mathcal{A}'$ wins in experiment $AggSignforge_{\mathcal{A}'}^{SAS}$ with non negligibly.*

Proof. If the adversary $\mathcal{A}$ successfully generate valid SAS signature of length m+1 by querying signing oracle at most m times, then it should forge one of his SAS signature. Without lose of generality, we assume that the adversary only successfully generate an valid signature corresponding to a public key which is decided by querying on random oracle. We can evaluate the success probability of $\mathcal{A}'$ as follows:

$$Prob[\mathcal{A}'] = Prob[\mathcal{A} \wedge k = i] \tag{1}$$
$$= Prob[\mathcal{A}|k = i] \quad Prob[k = i] \tag{2}$$
$$= \frac{1}{q_H} Prob[\mathcal{A}|k = i] \tag{3}$$

Thus, if adversary $\mathcal{A}$ success with noticeably, then the adversary $\mathcal{A}$ will break the SAS scheme non negligibly. □

The proof of theorem1 is straight based on the above two lemmas. □

5 Application—PoTcoin

In this section, we show how to use proof of transaction to construct a new cryptocurrency– PoTcoin. As mentioned earlier, our PoT protocol is very similar to the PoW protocol, except that PoT is used instead of PoW. In PoTcoin, we do not intend to detail the details of PoTcoin, we mainly focus on the differences between them. In the following discussion, we use bitcoin (or PoTcoin) to represent coin units in the Bitcoin protocol (or PoT protocol).

5.1 Backgrand of Bitcoin

There are many interesting innovations and features in the Bitcoin protocol. However, in order to give a better description our PoTcoin, we have given the corresponding knowledge.

Blockchain. All transactions in Bitcoin are on blockchain. Blockchain is stored in with a special cryptographic structure, and we represent the blockchain as $C = b_0, ..., b_N$. C is valid if and only if $b_i (i \leq N)$ is valid. A single block $b_i = (Tx_i, Nonce_i, h_{i-1})$ is valid if and only if the following three conditions must be met: Firstly, all transactions recorded in the block Tx_i are valid. That is to say, each transaction is signed by the sender and the spent money is not less than the expenditure; Secondly, the cryptographic hash value $H_{i-1} = SHA256(b_{i-1})$ must be the hash value of the previous block b_{i-1}; third, the random number $Nonce_i$ contained in the block b_i satisfies $SHA256(b_i) < 2^{256-\omega}$, where ω represents the difficulty parameter ,which we will discuss in detail below. The first condition ensures that the user can not afford to spend other user's money. The second condition ensures that the user can not forge a new blockchain

$C' = b_0, ..., b_{i-1}, b'_i, b'_{i+1}$. The third condition ensures that it is moderated difficult to mine a new block on the blockchain.

Reward,epoch bitcoin is issued in a predetermined ratio in the Bitcoin protocol. At this writing, 12.5 bitcoins are distributed about every 10 minutes (a epoch). When a new time epoch begins, the node generates a puzzle puz by calculating the latest block in the current blockchain. then, the nodes compete with each other to solve the puzzle puz for this epoch. Whenever which node firstly submits a valid answer, that node will get the reward newly mined within that epoch.

5.2 PoTcoin

Similar to bitcoin, all PoTcoin transactions are recorded in the blockchain, denoted as $C = b_0, ..., b_N$, where each block $b_i = (Tx_i, Ticket_i, H_{i-1})$ contains three pieces of data, namely, all transactions Tx_i in the block b_i, answer $Ticket_i$ of the proof of transaction puzzle and hash $H_{i-1} = SHA256(b_{i-1})$ of the previous block $b_{(i-1)}$. In block b_i, all transactions Tx_i must be valid and must contain the hash value $H_{i-1} = SHA256(b_{i-1})$ of the previous block b_{i-1}. In our PoT protocol, each user can find answers to PoT puzzle and verify the validity of the answer. In detail, suppose a given PoT puzzle system (Setup, G, $u^{\mathcal{O}(\cdot)}$, V), each user can get the parameters s and ω by running the algorithm Setup, a valid block b_i must contain a valid $Ticket_i$. Let the verifier output 1 after running algorithm $V(puz, Ticket, \omega)$, ticket is the valid answer to the puz. Given an effective blockchain $C = b_0, ..., b_N$, the user can construct a valid block by constructing a valid block $b_{N+1} = (Tx_{N+1}, Ticket_{N+1}, H_N)$ to get PoTcoin. In order to find $Ticket_{N+1}$, users must stay online and sign certain data to form a sequential signature chain of length ω. In the process of forming a sequential signature chain, if the next user to be exported not online or signature is not available, users need to start again to form a new chain of sequential signatures. Otherwise, the user finds a valid chain of sequential signatures and successfully constructs a valid block b_{N+1}.

The choice of parameters In Bitcoin, ω is a difficulty parameter. In general, it takes about 10 minutes for a miners to generate a block [NBF+16], that is, 2^ω times can be compute to effectively mining new block. In a new difficulty cycle, initially it takes 10 minutes or more to generate a block; however, as time goes on and the change of the computability, the time to generate the block is slowly less than 10 minutes ,until near the end, it may take less time to generate blocks (such as 6 minutes). We can clearly recognize that the dynamic difficulty is conducive to stability. If the difficulty is fixed, as more and more miners join the Bitcoin system , the time to generate blocks will be reduced. Therefore, the difficulty value ω must be periodically adjustable. In Bitcoin, the difficulty value ω is adjusted every 2016 blocks for about two weeks and the difficult parameter $\omega = \omega_{old} - log(\frac{t_{elapsed}}{2016 \times 10min})$ [NBF+16].

In the PoT protocol, our difficulty parameter $\omega = ploy(\lambda)$ is very easy to adjust. We can adjust the difficulty parameter ω directly by adjusting the security parameter λ. When PoT system, generating the block time becomes smaller

or larger, we can adjust the security parameter λ, making the block generation time stable at a certain time.

Reward distribute In the PoT protocol, on the one hand, we rewarded the users who mined the block, on the other hand, we motivated the users to stay online. The production of a new block requires a group of users to work together to complete. Each user is likely to be the first miner to form an effective chain of sequential signatures, but it is not so easy to become a second, third, etc. miner. Therefore, we allocate rewards and transaction fees to the block according to a certain percentage. We distribute the reward and the transaction fee to the miners who participated to mine new block by a certain proportion to their order in the sequential signature. For example, it takes k users to form a valid chain of sequential signatures, and all the reward the users receive is denoted as M. We divide k users into three equal batches of users. The first k / 3 users of the sequential signature chain are share the M / 5 of reward. The second k / 3 users share 3M / 10 reward; The last k / 3 part of users share M / 2 of reward. Of course, this ratio is not necessarily fixed and may change as the actual situation of the operation of the PoT protocol.

6 Advantage of PoTcoin

PoTcoin has the following advantages over bitcoin and other cryptocurrencies:

1. Enhancing network topology, facilitating PoTcoin circulation. In PoTcoin, in order to get the reward, firstly, users must stay online; Secondly users should initiate more transactions to increase their share of total transactions; Finally, online user rate and transaction numbers increase in the network , and it will strengthen the network topology and facilitate the circulation of PoTcoin .
2. Environmentally friendly. We know that cryptocurrencies such as Bitcoin consume a large amount of useful computing resources such as energy or storage space during mining [PPA+15]. And in our PoTcoin mining process, the user stays mostly online and does some signature calculations and verification, where resources are consumed as much as a normal computer consumes. So we think PoTcoin is environment-friendly.
3. Resistance to outsourcing computation. In Bitcoin, some "rational" miners outsource their mining resources to one or more large mining pool in order to expand their revenues and form Hosted mining, such as Alydian[DFKP15]. Hosted mining is very attractive, as it reduces the cost of miners mining due to economies of scale. In the PoT protocol, it is clear that if a user outsources his own mining resources (the ownership of the transaction – the private key) to several large mining pool, then, the user reveals his private key, and the large mining pool will able to take a way the PoTcoin in users account.

7 Conclusion

Currently, most cryptocurrencies are based on proof of work puzzle and proof of stake puzzle. However, these cryptocurrencies are faced with a very serious challenge. Bitcoin based on the proof of work face the problem of resource waste and security. The Peercoin based on the proof of stake faces the centralization of the currency.

In this paper, inspired by the challenges faced by cryptocurrencies, we construct a novel proof of transcation puzzle for the first time using sequential aggregation signatures. We show that proof of transcation puzzle satisfies the basic conditions of constructing scratch-off puzzle. We also designed a new cryptocurrency – PoTcoin, based on the proof of transcation puzzle. Our PoTcoin has good performance, for example, strengthening the network topology, facilitating the circulation of PoTcoin, resistance to outsourcing and environment-friendly. We leave behind a public challenge that how many transactions the user owns when the users will dominate the generation of blocks in the network? Just as Eyal et al.[ES13] analyzed in Bitcoin, selfish miners will dominate the generation of blocks in the network through selfish mining strategy, when selfish miners own mining power more than 25% of the total network.

References

[ABHL03] Luis Von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. Captcha: Using hard ai problems for security. *Lecture Notes in Computer Science*, 2656:294–311, 2003.

[Bac02] Adam Back. Hashcash - a denial of service counter-measure. In *USENIX Technical Conference*, 2002.

[BB14] Rachid El Bansarkhani and Johannes A. Buchmann. Towards lattice based aggregate signatures. In *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, pages 336–355, 2014.

[Ben] Proof of activity: Extending bitcoin's proof of work via proof , author=Bentov, Iddo and Lee, Charles and Mizrahi, Alex and Rosenfeld, Meni, year=2014,.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 416–432, 2003.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[But14] Vitalik Buterin. A next-generation smart contract and decentralized application platform. 2014.

[But16] Vitalik Buterin. Proof of stake faq. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ, 2016/ , 2016.

[BZ16] Jeremiah Blocki and Hong Sheng Zhou. Designing proof of human-work puzzles for cryptocurrency and beyond. In *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography - Volume 9986*, pages 517–546, 2016.

[CDFZ17] Alexander Chepurnoy, Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake. In *In Cryptology ePrint Archive*, 2017.

[CMSW09] Liqun Chen, Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. Security notions and generic constructions for client puzzles. In *International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, pages 505–523, 2009.

[Com16] NXT Community. Nxt whitepaper. https://www.dropbox.com/s/cbuwrorf672c0yy/ NxtWhitepaper v122 rev4.pdf/ , 2016.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. 9216:585–605, 2015.

[DFZ16] Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. In *In Cryptology ePrint Archive*, 2016.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *International Cryptology Conference on Advances in Cryptology*, pages 139–147, 1992.

[Dzi13] Stefan Dziembowski. Proofs of space and a greener bitcoin. 2013.

[ES13] Ittay Eyal and Emin Gn Sirer. Majority is not enough: Bitcoin mining is vulnerable. 8437:436–454, 2013.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits (extended abstract). *Annual IEEE Symposium on Foundations of Computer Science*, 311(2):40–49, 2013.

[GHa14] http://arstechnica:com/security/2014/06/bitcoin/security/guarantee/ shattered/by/anonymousminer/with/51/network/power/ , 2014.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 197–206, 2008.

[GW14] Bogdan Groza and Bogdan Warinschi. Cryptographic puzzles and dos resilience, revisited. 2014.

[JK07] Ari Juels and Burton S. Kaliski. Pors:proofs of retrievability for large files. In *ACM Conference on Computer and Communications Security*, pages 584–597, 2007.

[Kin13] S. King. Primecoin: Cryptocurrency with prime number proof-of-work. 2013.

[KN12] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012.

[Kwo] Jae Kwon. Tendermint: Consensus without mining.

[LK16] Kevin Liao and Jonathan Katz. Incentivizing blockchain forks via whale transactions. 2016.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *Advances*

in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings, pages 74–90, 2004.

[LOS+13] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures, multisignatures, and verifiably encrypted signatures without random oracles. *J. Cryptology*, 26(2):340–373, 2013.

[MJSP14] A Miller, A Juels, E Shi, and B Parno. Permacoin: Repurposing bitcoin work for data preservation. In *IEEE Symposium on Security and Privacy*, pages 475–490, 2014.

[MKKS15] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *ACM Sigsac Conference on Computer and Communications Security*, pages 680–691, 2015.

[Moo59] Frederick T Moore. Economies of scale: Some statistical evidence. *Quarterly Journal of Economics*, 73(2):232–245, 1959.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 2008.

[NBF+16] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[PPA+15] Sunoo Park, Krzysztof Pietrzak, Joel Alwen, Georg Fuchsbauer, and Peter Gazi. Spacecoin : A cryptocurrency based on proofs of space. 2015.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *Theory of Cryptography Conference*, pages 262–285, 2016.

[SKR+11] Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Gonzalez Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In *International Conference on Topics in Cryptology: Ct-Rsa*, pages 284–301, 2011.

[SSZ16] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532, 2016.

[Vit13] VitalikButerin. Bitcoin network shaken by blockchain fork. 2013.

[Wol05] Peter G. Wolynes. Energy landscapes and solved protein-folding problems. *Philosophical Transactions*, 363(1827):453, 2005.