Valuable Puzzles for Proofs-of-Work

Colin Boyd(✉) and Christopher Carr

Norwegian University of Science and Technology, Trondheim, Norway
{colin.boyd,chris.carr}@ntnu.no

Abstract. Proof-of-work (PoW) is used as the consensus mechanism in most cryptocurrencies. PoW-based puzzles play an important part in the operation and security of a cryptocurrency, but come at a considerable energy cost. One approach to the problem of energy wastage is to find ways to build PoW schemes from valuable computational problems. This work proposes calibration of public key cryptographic systems as a suitable source of PoW puzzles. We describe the properties needed to adapt public key cryptosystems as PoW functions suitable for decentralised cryptocurrencies and provide a candidate example.

Keywords: Proof-of-work · Useful computation · Blockchain

1 Introduction

Proof-of-work (PoW) mechanisms are an integral part of modern cryptocurrencies, such as Bitcoin and the numerous altcoin variants [20], where they are used to maintain consensus. Despite their successful employment for this task, a source of contention for proofs-of-work is the energy wastage associated with their use [14,16]. On the other hand, the developers of Bitcoin claim that the waste of energy is analogous to the energy expenditure of other financial institutions, such as banks and credit card companies [20]. Even so, the high energy consumption of PoW systems is a concern, and one that is not easily avoided. A main purpose of PoW is to manage the Sybil vulnerability problem [7]. Devising an authority-free decentralised cryptocurrency, that does not suffer from Sybil vulnerabilities and does not use PoW, remains an open problem.

The approach of this work is to design a PoW mechanism that is useful outside of the cryptocurrency it is intended to support. While the energy expenditure would still continue, there would at least be some other value in the execution of the PoW function. The intention is to provide insight into the construction of PoW functions from arbitrary computational puzzles. To illustrate the applicability of this idea, a particular focus is placed on public-key schemes.

Adapting public-key schemes for use as PoW has potential advantages:

1. It can incentivise calibration techniques in software and hardware through the reward structure.

J. Garcia-Alfaro et al. (Eds.): DPM 2018/CBT 2018, LNCS 11025, pp. 130–139, 2018.
https://doi.org/10.1007/978-3-030-00305-0_10

2. It can provide data points to more accurately set safe parameter choices for public key schemes.
3. It can be used for specific public-key schemes to encourage their analysis.

While some public key schemes have undergone considerable practical analysis in the past, this is not true in general. The level of scrutiny applied to any specific scheme is sometimes unclear, especially when the underlying problem has been recently introduced, as in the case of the ring learning with errors problem [5]. Public, large-scale analysis of cryptosystems is not without precedent. RSA Laboratories famously offered cash prizes for factoring large composite numbers [17]. The approach was relatively successful, as many of the challenges remain un-factored, and general understanding of factoring algorithms increased.

Building PoW puzzles from public key schemes has the potential to increase the awareness and level of scrutiny applied to them, and to encourage analysis – both valuable to the cryptographic community. In fact, using a market-driven approach, inherent in competitive proof-of-work schemes, can have the effect of incentivising clever cryptanalytic techniques as well as smart specific hardware designs that target weaknesses in a given scheme. If public key based puzzles stand up to scrutiny for a period of time, without major speedups or breakthroughs, the level of confidence in the scheme's security will grow.

Related Work. The idea of utilising the computational work carried out in PoW schemes for some useful purpose has been around for a while and was first proposed by Dwork and Naor [8]. Despite useful puzzles being addressed early on, there are still relatively few candidates. It seems that a significant problem lies in finding candidate puzzles that can be moulded into a PoW puzzle.

The cryptocurrency Primecoin [11] uses the search for Cunningham Prime Chains as the PoW function. This example demonstrates the possibility to find puzzles that satisfy some of the conditions for adaptability into a proof-of-work mechanism. Gridcoin [22] rewards users for their attempts to solve @home puzzles, for example folding@home [21]. But Gridcoin does not offer decentralisation, equating to simply handing out tokens for the effort of solving certain puzzles. Ball et al. [1] demonstrate the adaptability of specific problems, known as the Orthogonal Vectors and 3SUM problems, into a PoW framework. They rely on a distributed problem board, where specified delegated parties issue problems that can be used to create challenges. We aim to devise authority free, decentralised, proofs-of-work, and so no delegated party or problem board is required.

There are other works that examine the energy expenditure problem in PoW systems. Most solutions rely on removing the competitive computational aspect of proof-of-work, replacing it with some different method, such as proof-of-stake [10], proof-of-activity [2] or proof-of-commitment [6]. Tschorsch and Scheuermann [19] give a concise overview of these alternatives.

Contributions. The primary goal is to give some insight into the possibility of adapting generic computational puzzles into a PoW framework. This is achieved by stating and explaining the reasoning behind the requirements for this adaptation, and providing definitions and formalism where necessary. Using the

Schnorr signature scheme as an example, a transformation into a PoW scheme is described.

2 Puzzles and Their Properties

Most decentralised cryptocurrencies use a consensus mechanism which relies on the partial pre-image resistance of a chosen hash function.

A notable gain in understanding from the use of the SHA-256 hash function in Bitcoin is that, despite the speedups and development of efficient hardware, there is no evidence of a solution finding method that is any better than brute force search. Another useful insight gained is the ability to quantify the time it would take to find a full preimage of a message digest. This ability is crucial when selecting the difficulty parameter for cryptocurrencies that use PoW.

In order to derive such benefits for more general problems, we need to fit them into a cryptocurrency PoW framework. Puzzles used for Bitcoin-like consensus have certain characteristics which are fundamental to the smooth operation of the cryptocurrency. In order to use alternative puzzles for the PoW mechanism, it is necessary to construct them with these characteristics in mind. We would like to retain the Bitcoin structures, such as blocks and transactions, and identify the abstract interface to the PoW puzzle. We start by defining a puzzle set.

Definition 1 (Puzzle Set). *A puzzle set PS is a tuple of three efficient algorithms* `Setup`, `GenPuz`, `FindSol` *and a deterministic algorithm* `VerSol`. *Let λ be the setup parameter, $\mathcal{D}$ the difficulty space, $\mathcal{S}tr$ the message space, $\mathcal{P}$ the puzzle space and $\mathcal{S}ol$ be the solution space.*

1. $\texttt{Setup}(1^\lambda)$: *Select $\mathcal{D}, \mathcal{S}tr, \mathcal{P}, \mathcal{S}ol$ and return $(\mathcal{D}, \mathcal{S}tr, \mathcal{P}, \mathcal{S}ol)$.*
2. $\texttt{GenPuz}(d \in \mathcal{D}, m \in \mathcal{S}tr)$: *Return $p \in \mathcal{P}$ or $\perp$.*
3. $\texttt{FindSol}(m \in \mathcal{S}tr, p \in \mathcal{P}, t \in \mathbb{N})$: *Return $s \in \mathcal{S}ol$ after at most t steps.*
4. $\texttt{VerSol}(m \in \mathcal{S}tr, p \in \mathcal{P}, s \in \mathcal{S}ol)$: *Return* `true` *or* `false`.

A puzzle set may be defined without a solution finding algorithm. It is included here only for completeness. From now on the `FindSol` algorithm is purposefully omitted. If a solution finding algorithm is included, then there is a correctness requirement as follows: Let $params \leftarrow \texttt{Setup}(1^\lambda)$ and $p \leftarrow \texttt{GenPuz}(d, m)$, where $d \in \mathcal{D}$ and $m \in \mathcal{S}tr$, then there exists $t \in \mathbb{N}$ where

$$\Pr[\texttt{VerSol}(m, p, s) = \texttt{true} \mid s \leftarrow \texttt{FindSol}(m, p, t)] = 1.$$

The Bitcoin puzzle fits the structure of Definition 1 where: $\mathcal{D}$ is the set of valid difficulty levels; $\mathcal{S}tr$ is the combination of hash of the previous block header and the set of valid user inputs (nonce, transactions and other parameters); $\mathcal{P}$ is just the concatenation of the difficulty and valid input strings; and $\mathcal{S}ol$ is the set of hash inputs that hash below the current target.

A new puzzle must have the interfaces of Definition 1, but must also satisfy some properties to ensure that the incentive properties of Bitcoin are retained. We call these *fairness requirements* (FRs). It is not possible to prove what are the

correct fairness requirements without extensive real-world trials, because they depend on human behaviour. Thus we define properties based on the perceived critical properties of the Bitcoin puzzle. We can also take guidance from previous efforts to define puzzle properties, including those of Miller et al. [12,13], Narayanan et al. [14], and Biryukov and Khovratovich [3].

FR.1 *Creator Free:* Finding a solution to one puzzle must not give any advantage in solving of any other.

Once a Bitcoin puzzle is solved, the solution is distributed to all participants and used to form a new puzzle. Specifically, the header information from a previous block is used as input to the next block. The header data is unpredictable until a solution is found, so even the solver will have no extra information to help make a start on finding the next puzzle solution.

In essence, this requirement aims to ensures that no party has an advantage in finding the solution to the new puzzle, even if they have solved the previous one. We note however, that this is not satisfied in existing implementations. It has been shown that it is possible to perform *selfish mining* [9,18], where the solver of the previous block does not distribute the solution immediately in order to gain some time advantage on solving the next one.

To formally define FR.1 we first describe two security experiments in Fig. 1. In both experiments the goal of the adversary $\mathcal{A}$ is to solve any one of the set of puzzles defined using the inputs $m_i = (m_{1,i}, m_{2,i})$. The difference between the two experiments is that in the first $\mathcal{A}$ selects both $m_{1,i}$ and $m_{2,i}$ for input into the `GenPuz` algorithm, and in the second $m_{1,i}$ is selected at random from the $\mathcal{S}tr$ set. This reflects the Bitcoin puzzle set where the input string consists of two parts: one coming from the previous block and one which can be influenced by the miner. The ability to influence the first part should not help an adversary.

$\mathbf{Exp}^{\mathsf{PzSol}}_{\mathcal{A},d,n,t}$:

$m_{1,1}, m_{1,2}, \ldots, m_{1,n} \leftarrow \mathcal{A}$
$m_{2,1}, m_{2,2}, \ldots, m_{2,n} \leftarrow \mathcal{A}$
$\{m_i = (m_{1,i}, m_{2,i}) | \forall i \in \{1, 2, \ldots, n\}\}$
$p_1 \leftarrow \mathtt{GenPuz}(d, m_1), p_2 \leftarrow \mathtt{GenPuz}(d, m_2), \ldots, p_n \leftarrow \mathtt{GenPuz}(d, m_n)$
$s \leftarrow \mathcal{A}$
return (m_i, p_i, s) for some $i \in \{1, 2, \ldots, n\}$

$\mathbf{Exp}^{\mathsf{PzSolR}}_{\mathcal{A},d,n,t}$:

$m_{1,1}, m_{1,2}, \ldots, m_{1,n} \xleftarrow{\$} \mathcal{S}tr$
$m_{2,1}, m_{2,2}, \ldots, m_{2,n} \leftarrow \mathcal{A}$
$\{m_i = (m_{1,i}, m_{2,i}) | \forall i \in \{1, 2, \ldots, n\}\}$
$p_1 \leftarrow \mathtt{GenPuz}(d, m_1), p_2 \leftarrow \mathtt{GenPuz}(d, m_2), \ldots, p_n \leftarrow \mathtt{GenPuz}(d, m_n)$
$s \leftarrow \mathcal{A}$
return (m_i, p_i, s) for some $i \in \{1, 2, \ldots, n\}$

Fig. 1. Creator free experiments

For the experiments in Fig. 1 we define the game G to **win**, for some difficulty d, fixed n, for some efficient $\mathcal{A}$ returning (m_i, p_i, s), $i \in \{1, 2, \ldots, n\}$ and running in at most time t, if $\texttt{VerSol}(m_i, p_i, s)$ returns true. Succinctly we write $\mathbf{Exp}^{\mathsf{G}}_{\mathcal{A},d,n,t} = 1$, else we write $\mathbf{Exp}^{\mathsf{G}}_{\mathcal{A},d,n,t} = 0$.

Definition 2 (Creator Free). *Let PS be a puzzle set with setup parameter λ. We say that PS is* creator free *if for any d, n and any efficient $\mathcal{A}$ running in time t we can define an efficient $\mathcal{B}$ running in approximately the same time $t' \approx t$, such that*

$$\Pr[\mathbf{Exp}^{\mathsf{PzSol}}_{\mathcal{A},d,n,t} = 1] - \Pr[\mathbf{Exp}^{\mathsf{PzSolR}}_{\mathcal{B},d,n,t'} = 1] \leq \mathsf{negl}(\lambda).$$

FR.2 *Puzzle independence:* It should not be possible to use the effort expended to solve one puzzle, to solve another.

Puzzle independence requires that even if you can create multiple puzzles, all the effort expended towards solving any specific one of them will not give any advantage in solving another distinct puzzle. In Bitcoin, puzzles are independent as one cannot use the work directed towards solving one block, to help with the solution to another. This is because each new puzzle is formed by an unpredictable pseudo-random string each time, for each block.

$\underline{\mathbf{Exp}^{\mathsf{PzIndR}}_{\mathcal{A},d,n,t}}$:

$m_{1,1}, m_{1,2}, \ldots, m_{1,n} \xleftarrow{\$} \mathcal{S}tr$
$m_{2,1}, m_{2,2}, \ldots, m_{2,n} \leftarrow \mathcal{A}$
$\{m_i = (m_{1,i}, m_{2,i}) | \forall i \in \{1, 2, \ldots, n\}\}$
$p_1 \leftarrow \texttt{GenPuz}(d, m_1), p_2 \leftarrow \texttt{GenPuz}(d, m_2), \ldots, p_n \leftarrow \texttt{GenPuz}(d, m_n)$
return $(m_i, p_i, s_i) \forall i \in \{1, 2, \ldots, n\}$

Fig. 2. Puzzle independence experiment

For Fig. 2, as in Fig. 1, we define the game G to **win**, for some difficulty d, fixed n, for some algorithm $\mathcal{A}$ returning (m_i, p_i, s_i), $\forall i \in \{1, 2, \ldots, n\}$ and running in at most time t, if $\texttt{VerSol}(m_i, p_i, s_i)$ returns true for every i.

Definition 3 (Puzzle Independence). *Let PS be a puzzle set with setup parameter λ. We say that PS has* puzzle independence *if for any d, n and any efficient $\mathcal{A}$ running in time t we can define an efficient $\mathcal{B}$ running in at most time t'/n, where $t' \approx t$ such that*

$$|\Pr[\mathbf{Exp}^{\mathsf{PzIndR}}_{\mathcal{A},d,n,t} = 1] - (\Pr[\mathbf{Exp}^{\mathsf{PzIndR}}_{\mathcal{B},d,1,t/n} = 1])^n| \leq \mathsf{negl}(\lambda).$$

FR.3 *Chance to win:* Every participant should have some non-negligible chance of solving a puzzle before any other.

In Bitcoin, the probability of being the first to solve the puzzle is directly proportional to one's share of the computational power directed towards the puzzle at a given time. Note that FR.3 only asks for some non-negligible chance that a participant can win, it does not require any specific probability distribution. Previous authors [3,14] have proposed a related property called *progress-free* which states that solving a puzzle should be a Poisson process. Such a definition may be too strict; it excludes some useful examples, while the concrete parameters used will determine what is sufficient incentive for a small user to participate.

In addition to the fairness requirements, there are practical requirements (PRs) that can be identified to ensure that any new puzzle is useable in a real system. We mostly give these informally, since usability is not easy to quantify.

PR.1 *Linkable puzzles:* A previous puzzle solution can be used to form a new puzzle.

The security of transactions within a PoW based distributed ledger relies on encoding the transactional data along with the puzzle. For this data to persist, the solution of each puzzle is used to form a new puzzle, so the puzzle solution acts as a pointer to the previous transactions. This forms the ledger. Specifically in Bitcoin, each new block contains information relating to the previous block.

Definition 4 (Linkable). *Let PS be a puzzle set with setup parameter λ, then we say that a PS is linkable if $\mathcal{S}ol \subseteq \mathcal{S}tr$.*

PR.2 *Efficiently Verifiable:* The solution must be efficient and quick to verify by all parties.

PR.3 *Tunable:* The difficulty, or expected number of computational steps, of finding a puzzle solution must be adjustable in order to increase and decrease the difficulty of finding a solution to a puzzle.

PR.4 *Valuable:* Puzzles should provide some useful function in the finding of their solution, other than their purpose within the PoW scheme.

3 Generic Bitcoin-Like Construction

We can now describe a generic construction for a Bitcoin-like puzzle in Definition 5. This is an abstract version of the Bitcoin puzzle construction, where each puzzle instance is generated by a hash output where the input has two parts, in addition to a difficulty parameter.

Definition 5 (Bitcoin-like puzzle). *A Bitcoin-like puzzle generation algorithm is a puzzle set, described by three algorithms:*

- $\texttt{Setup}(1^\lambda) : \mathcal{D} = \mathbb{Z}, \mathcal{S}tr = \{0,1\}^*, \mathcal{P} = \{0,1\}^{n_1}, \mathcal{S}ol = \{0,1\}^{n_2}$ *for some* $n_1, n_2 \in \mathbb{N}$.
- $\texttt{GenPuz}(d, m = (m_1, m_2))$ *computes* $\tilde{p} \leftarrow H(m_1||m_2)||d$ *for* $H : \{0,1\}^* \rightarrow \{0,1\}^{n_3}$*, where H is a pseudo-random function with* $n_3 \in \mathbb{N}$*, and returns* $p \leftarrow \texttt{CreatePuz}(\tilde{p})||\tilde{p}$*, where* `CreatePuz` *is a deterministic algorithm, with running time parameterised by* λ.

– $\texttt{VerSol}(m, p, s)$ *returns* **true** *if* $p = p' \leftarrow \texttt{GenPuz}(d, m)$ *and* $\texttt{CheckSol}(p, s)$ *returns* **true**, *else it returns* **false**, *where* `CheckSol` *is a deterministic boolean algorithm.*

The next result shows that any Bitcoin-like puzzle satisfies FR.1 and FR.2. The proof is omitted due to space constraints.

Theorem 1. *Let PS be a Bitcoin-like puzzle generation algorithm with setup parameter λ. If H is a random oracle, then for a fixed $d \in \mathcal{D}$, a fast and efficient* `GenPuz` *algorithm, PS is creator free and is linkable.*

Moreover, PS is efficiently verifiable if both H and `CheckSol` combined terminate in time significantly less than the time required to find a solution. The puzzle set PS is tunable if when d is increased, it takes on average more computational steps to find a corresponding puzzle and solution p, s that satisfies `VerSol`, and vice-versa.

Figure 3 further describes the puzzle chaining process. This process explicitly defines the solution to a previous puzzle as part of the message, which is used to generate the next puzzle. This links the puzzles and the solutions together. Figure 3 is in practice how one would expect a PoW mechanism to operate, though there may be different variations.

Bitcoin-like Chained Puzzle:
Let PS be a Bitcoin-like puzzle generation algorithm as in Defn. 5. For any $i > 0, i \in \mathbb{N}$, with predefined constant $s_0 \in \mathcal{S}ol$, the Bitcoin-like chained puzzle is defined by:

1: $s = s_{i-1}$.
2: $a_i = \mathsf{input}()$. \\collect auxiliary inputs
3: $m = (s, a_i)$.
4: $p_i = \texttt{GenPuz}(d, m)$.
5: $s_i = \mathsf{input}()$. \\attempts to find the puzzle solution
6: **-If:** $\texttt{VerSol}(m, p_i, s_i)$ returns true, $s = s_i$, **goto** 2.
7: **Else: goto** 5.

Fig. 3. Bitcoin-like chained puzzle

4 Schnorr Signature Puzzles

The Schnorr signature scheme public key generation procedure, as described by Boneh [4], selects random primes p and q such that $q|p-1$, an element $g \in \mathbb{Z}_p^*$ of order q, an element $a \in \mathbb{Z}_q$ and computes $y = g^a \in \mathbb{Z}_p^*$. The scheme also uses some public hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$. The public parameters are (p, q, g, y, H), with a as the private key.

The goal is to create Bitcoin-like puzzles by describing a method for generating random public keys, without corresponding private keys. The puzzle is then to find a corresponding private key, or otherwise form a signature on the input

$\texttt{Setup}(1^\lambda)$:

1: **Return:** $\mathcal{D} = \mathbb{N}, \mathcal{S}tr = \{0,1\}^*, \mathcal{P} = (\mathbb{Z}^4, H : \mathcal{S}tr \to \mathbb{Z}), \mathcal{S}ol = \mathbb{Z}^2$, for some pseudo-random function H.

$\texttt{GenPuz}(d \in \mathcal{D}, m \in \mathcal{S}tr)$:

1: **Select:** $\hat{m}$.

2: **Return:** $p = \hat{m}||d$.

$\texttt{VerSol}(m \in \mathcal{S}tr, p \in \mathcal{P}, s \in \mathcal{S}ol)$:

1: **Input:** $(m, F(p), s = (\sigma, \gamma))$.

2: –**If:** $p = p' \leftarrow \texttt{GenPuz}(d, m)$, continue, else return false.

3: **Run:** $v = g^\sigma y^{-\gamma} \mod \mathbb{Z}_p$.

4: –**If:** $H(m||v) = \gamma, \pi =$ true, else $\pi =$ false.

5: **Return:** π.

Fig. 4. Schnorr signature puzzle algorithms

message. A puzzle set for the Schnorr signature forgery puzzle is described in Fig. 4.

To complete the puzzle definition, we need to define the function F used in the VerSol algorithm of Fig. 4. Due to space constraints we omit the details, but the general idea is to use a deterministic version of the parameter generation process from the FIPS digital signature standard [15, Appendix A]. Using the value $\hat{m}$ as the parameter seed, first q, then p, then g and finally the public verification key y, are all generated. This method generates the public key without a corresponding secret key. Finding the secret key, or otherwise signing the message m becomes the PoW challenge. By relying on the randomness provided by the hash function H, this puzzle set is linkable and creator free by Theorem 1.

We are not able to prove that puzzle independence (FR.2) holds for the Schnorr puzzle due to the nature of the puzzle generation algorithm. If two distinct puzzles are generated with the same initial primes p and q, then this could give an advantage to a potential solver who has retained some computation for the number field sieve algorithm. We conjecture that in practical cases the puzzles will have the FR.2 property, since selecting a p and q that have been used before is very unlikely.

5 Conclusion

An abstract puzzle construction has been demonstrated as well as describing how the Schnorr signature scheme can be used for a stand-in PoW scheme. Moreover, the parameter generation is applicable to DSA and ElGamal signatures with only minor alterations. The clear route for future work is to adapt different types of public-key schemes, or puzzles in general, for use in PoW systems using the requirements here. A wider variety of puzzles may not only prove to be

more valuable in terms of the actual puzzle, but could also potentially help with resistance to the design of ASICs for specific fixed puzzles.

References

1. Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Average-case fine-grained hardness. Cryptology ePrint Archive, Report 2017/202 (2017). http://eprint.iacr.org/2017/202
2. Bentov, I., Lee, C., Mizrahi, A., Rosenfeld, M.: Proof of activity: extending Bitcoin's proof of work via proof of stake [extended abstract]. SIGMETRICS Perform. Eval. Rev. **42**(3), 34–37 (2014)
3. Biryukov, A., Khovratovich, D.: Equihash: asymmetric proof-of-work based on the generalized birthday problem. In: NDSS 2016. The Internet Society, February 2016
4. Boneh, D.: Schnorr digital signature scheme. In: van Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security, 2nd edn., pp. 1082–1083. Springer, Heidelberg (2011)
5. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: take off the ring! Practical, quantum-secure key exchange from LWE. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1006–1018. ACM Press, October 2016
6. Clark, J., Essex, A.: CommitCoin: carbon dating commitments with bitcoin. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 390–398. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_28
7. Douceur, J.R.: The Sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A.I.T. (eds.) Peer-to-Peer Systems, First International Workshop, IPTPS (2002)
8. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_10
9. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_28
10. Kiayias, A., Konstantinou, I., Russell, A., David, B., Oliynykov, R.: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889 (2016). http://eprint.iacr.org/2016/889
11. King, S.: Primecoin: a cryptocurrency using the search for Cunningham prime chains as the proof-of-work mechanism (2013). http://primecoin.io/. Accessed Jan 2018
12. Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: repurposing bitcoin work for data preservation. In: 2014 IEEE Symposium on Security and Privacy, pp. 475–490. IEEE Computer Society Press, May 2014. https://doi.org/10.1109/SP.2014.37
13. Miller, A., Kosba, A.E., Katz, J., Shi, E.: Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 680–691. ACM Press, October 2015
14. Narayanan, A., Bonneau, J., Felten, E.W., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction. Princeton University Press (2016). ISBN: 978-0-691-17169-2
15. National Institute of Standards and Technology: Digital Signature Standard (DSS), July 2013. http://dx.doi.org/10.6028/NIST.FIPS.186-4

16. O'Dwyer, K.J., Malone, D.: Bitcoin mining and its energy footprint. In: Irish Signals and Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). IET (2014)
17. RSA-Laboratories: RSA factoring challenges. http://www.isiloniq.com/emc-plus/rsa-labs/historical/the-rsa-challenge-numbers.htm. Accessed Jan 2017
18. Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in bitcoin. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 515–532. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_30
19. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: a technical survey on decentralized digital currencies. IEEE Commun. Surv. Tutorials **18**(3), 2084–2123 (2016)
20. Web: Bitcoin Wiki. https://en.bitcoin.it/wiki/Main_Page. Accessed Jan 2018
21. Web: Folding@home. http://folding.stanford.edu/. Accessed Feb 2018
22. Web: Gridcoin. https://gridcoin.us (2017). Accessed June 2018