The Proof is in the Pudding

Proofs of Work for Solving Discrete Logarithms

Marcella Hastings[1], Nadia Heninger[1], and Eric Wustrow[2]

[1] University of Pennsylvania
mhast@cis.upenn.edu, nadiah@cis.upenn.edu
[2] University of Colorado Boulder
ewust@colorado.edu

Abstract. We propose a proof of work protocol that computes the discrete logarithm of an element in a cyclic group. Individual provers generating proofs of work perform a distributed version of the Pollard rho algorithm. Such a protocol could capture the computational power expended to construct proof-of-work-based blockchains for a more useful purpose, as well as incentivize advances in hardware, software, or algorithms for an important cryptographic problem. We describe our proposed construction and elaborate on challenges and potential trade-offs that arise in designing a practical proof of work.

Keywords: Proofs of work, discrete log, Pollard rho

1 Introduction

We propose a proof of work scheme that is useful for cryptanalysis, in particular, computing discrete logarithms. The security of the ECDSA digital signature scheme is based on the hardness of the elliptic curve discrete log problem. Despite the problem's cryptographic importance, the open research community in the area is small and has limited resources for the engineering work and computation required to update cryptanalytic records; recent group sizes for elliptic curve discrete log records include 108 bits in 2002 [12], 112 bits in 2009 [10], and 113 bits in 2014 [35].

We propose a proof of work scheme to harness the gigawatts of energy spent on Bitcoin mining [34] to advance the state of the art. Jakobsson and Juels [17] call this a *bread pudding* proof of work. Like the dessert that makes stale bread delicious, individual proofs of work produce a useful computation. While memory-hard functions aim to discourage specialized hardware for cryptocurrency mining [25], we hope for the exact opposite effect: as Bitcoin has prompted significant engineering effort to develop efficient FPGAs and ASICs for SHA-256, we wish to use the lure of financial rewards from cryptocurrency mining to incentivize special-purpose hardware for cryptanalysis.

2 Background

Let G be a cyclic group with generator g of order q. We represent the group operation as multiplication, but every algorithm in our paper applies to a generic group. Every element $h \in G$ can be represented as an integer power of g, $g^a = h$, $0 \leq a < q$. We also assume that every element h of G can be mapped to a unique representation as a sequence of bits. The discrete logarithm $log_g(h)$ is a, $0 \leq a < q$ satisfying $g^a = h$. Computing discrete logs is believed to be difficult for certain groups, including multiplicative groups modulo primes and elliptic curve groups. The conjectured hardness of discrete log underlies the security of multiple important cryptographic algorithms, including the Diffie-Hellman key exchange [13, 5] and the Digital Signature Algorithm [24]. Efficient computation of a discrete log for a group used for Diffie-Hellman key exchange would allow an adversary to compute the private key from the public key exchange messages; for DSA signatures, such an adversary could compute the private signing key from the public key and forge arbitrary signatures.

2.1 Discrete Log Cryptanalysis

There are two main families of algorithms for solving the discrete log problem. The first family works over any group, and includes Shanks's baby step giant step algorithm [28], and the Pollard rho and lambda algorithms [26]. These algorithms run in time $O(\sqrt{q})$ for any group of order q. It is this family of algorithms we target in this paper. A second family of algorithms is based on index calculus and works only over finite fields; this family includes the number field sieve which has subexponential running time for large- and medium-characteristic finite fields [16] and the function field sieve which was recently improved to quasipolynomial running time for small-characteristic finite fields [4].

In practice, group parameters for Diffie-Hellman and DSA are chosen to avoid the known families of cryptanalytic attacks. To avoid index calculus-based attacks in multiplicative groups over finite fields, the minimum recommended prime modulus size is 2048 bits [19], with a minimum subgroup of order 256 or 224 bits [5, 20]. However, 1024-bit prime moduli with 160-bit subgroups remain common in practice [31]. Current best practices for elliptic curves are to use 256-bit curves [5], although 160-bit curves remain supported in some implementations [32].

Bitcoin miners currently perform around 2^{90} hashes per year and consume 0.33% of the world's electricity [34]. If this effort were instead focused on discrete log, a 180-bit curve could be broken in around a year[3]. Scaling this to discrete logs in 224-bit groups would require all current electricity production on Earth for 10,000 years.

[3] We note elliptic curve point multiplications take about 2^{10} times longer than SHA-256 on a modern CPU.

2.2 Pollard Rho with Distinguished Points

The protocols we study in this paper compute the discrete log of an element h by finding a collision $g^a h^b = g^{a'} h^{b'}$ with $b \not\equiv b' \mod q$. Given such an equivalence, the discrete log of h can be computed as $(a' - a)/(b - b') \bmod q$. In other words, given two elements whose bitwise representations collide, we can compute the discrete logarithm from their group representations. Our proof of work is based on a parallelized version of the Pollard rho algorithm due to Van Oorschot and Wiener [33], called the method of distinguished points

The Pollard rho algorithm. Pollard's rho algorithm for discrete logarithms [26] works for any cyclic group G of order q. The main idea is to take a deterministic pseudorandom walk inside of the group until the same element is encountered twice along the walk. By the birthday bound, such an element will be found with high probability after $\Theta(\sqrt{q})$ steps. The non-parallelized version of this algorithm uses a cycle-finding algorithm to discover this collision, and computes the log as above.

We base our proof of work on Van Oorschot and Wiener's [33] parallelized Pollard rho algorithm using the method of distinguished points. A distinguished point is an element whose bitwise representation matches some easily-identifiable condition, such having d leading zeros. Each individual process j independently chooses a random starting point $g^{a_j} h^{b_j}$ and generates a psuedorandom walk sequence from this starting element. When the walk reaches a distinguished point, the point is saved to a central repository and the process starts over again from a new random starting point until a collision is found.

The number of steps required to compute the discrete log is independent of d, which we call the difficulty parameter below; d only determines the storage required. We expect to find a collision after $\Theta(\sqrt{q})$ steps by all processes. With m processes running in parallel, the calendar running time is $O(\sqrt{q}/m)$.

Pseudorandom Walks. The pseudorandom walk produces a deterministic sequence within the group from some starting value. Given a group generator g and a target h, the walk generates a random starting point $x_0 = g^{a_0} h^{b_0}$ by choosing random exponents a_0, b_0. The original walk introduced by Pollard divides G into a disjoint partition T_0, T_1, T_2 and defines the function:

$$\mathcal{W}_\rho(x) = \begin{cases} gx & x \in T_0 \\ x^2 & x \in T_1 \\ hx & x \in T_2 \end{cases} \tag{1}$$

At each point $x_i = g^{a_i} h^{b_i}$, the known exponents a_i, b_i are updated according to this formula. For multiplicative groups mod p, the sets T_0, T_1, and T_2 are typically set to be $T_0 = [1, \ldots, p/3)$, $T_1 = [p/3, \ldots, 2p/3)$, and $T_2 = [2p/3, \ldots, p)$.

In practice, most implementations use the linear pseudorandom walk introduced by Teske [30]. Given a disjoint partition of G with 20 sets of equal size

$T_1, \ldots, T_{20}$ parameterized by the bitwise representation of an element, choose $m_s, n_s \in \{1, q\}$ at random and define $M_s = g^{m_s} h^{n_s}$ for $s \in [1, 20]$. Then we can define the walk $\mathcal{W}(x) = M_s * x$ for $x \in T_s$.

2.3 Discrete Log Records

The parallelized Pollard rho algorithm has been used to set a number of elliptic curve discrete log records. Monico computed a 109-bit elliptic curve discrete log from the 1997 Elliptic Curve Cryptosystem Challenge [12] in 2002 using 10,000 CPU users over 549 days. Monico also solved a separate 109-bit challenge in 2004, with 2600 CPU users over 17 months using Teske's linear walk [30].

Bos et al. [10] solved a 112-bit elliptic curve discrete logarithm in 2012 using 200 Sony Playstation 3 game consoles in around 6 months, using an optimized version of the method of distinguished points. Wenger and Wolfger used 10 FPGAs to solve a 113-bit elliptic curve discrete logarithm [35] in 2016 in around 2.5 months using a random walk due to Wiener and Zuccherato [36].

2.4 Proofs of Work

A proof of work [14, 17] protocol allows a *prover* to demonstrate to a *verifier* that they have executed an amount of work. We use the definition from [3].

Definition 1. *A $(t(n), \delta(n))$-Proof of Work (PoW) consists of three algorithms $(\mathsf{Gen}, \mathsf{Solve}, \mathsf{Verify})$ that satisfy the following properties:*

- ***Efficiency**:*
  - $\mathsf{Gen}(1^n)$ *runs in time* $\tilde{O}(n)$.
  - *For any* $c \leftarrow \mathsf{Gen}(1^n)$, $\mathsf{Solve}(c)$ *runs in time* $\tilde{O}(t(n))$.
  - *For any* $c \leftarrow \mathsf{Gen}(1^n)$ *and any* π, $\mathsf{Verify}(c, \pi)$ *runs in time* $\tilde{O}(n)$.
- ***Completeness**: For any* $c \leftarrow \mathsf{Gen}(1^n)$ *and any* $\pi \leftarrow \mathsf{Solve}(c)$,

$$\Pr[\mathsf{Verify}(c, \pi) = \mathit{accept}] = 1.$$

- ***Hardness**: For any polynomial ℓ, any constant $\epsilon > 0$, and any algorithm Solve^*_ℓ that runs in time $\ell(n)t(n)^{1-\epsilon}$ when given as input $\ell(n)$ challenges $\{c_i \leftarrow \mathsf{Gen}(1^n)\}_{i \in [\ell(n)]}$,*

$$\Pr\left[\forall i\, \mathsf{Verify}(c_i, \pi_i) = \mathit{accept} \mid (\pi_1, \ldots, \pi_{\ell(n)}) \leftarrow \mathsf{Solve}^*_\ell(c_1, \ldots, c_{\ell(n)})\right] < \delta(n)$$

We can describe the hash puzzle proof of work [2] used by Bitcoin [23] and other cryptocurrencies in this framework as follows. The challenge generated by Gen is the hash of the previous block. Solve is parameterized by a difficulty d; individual miners search for a nonce n such that SHA-256$(c, n) \leq 2^{256-d}$ when mapped to an integer. Assuming that SHA-256 acts like a random function, miners must brute force search random values of n; the probability that a random fixed-length integer is below the difficulty threshold is 2^{-d}, so the conjectured running time for Solve is $t(n) = O(2^d)$. Verify accepts if SHA-256$(c, n) \leq 2^{256-d}$; this algorithm is constant time.

There are several proposals for "useful" proofs of work. Primecoin [18] proofs contain prime chains, which may be of scientific interest. The culmination of DDoSCoin [37] proofs of work can result in a denial of service attack. TorPath [6] incentivizes miners to improve bandwidth on the Tor network. Ball et al. [3] describe theoretical proof-of-work schemes based on worst-case hardness assumptions from computational complexity theory. SpaceMint [1] and Permacoin [22] incentivize miners to consume storage or store meaningful data. Lochter [21] independently posted a preprint outlining a discrete log proof of work similar to ours.

3 Proof of work for discrete log

We aim to construct a proof of work scheme where the combined work of the proofs solve a discrete log. Bitcoin miners have executed more than 2^{80} hashes; an efficient scheme could set a computational record for a discrete log in a 160-bit group. We present the general idea behind our proposed scheme, explain limitations of the simple model, and describe possible avenues to fix the gap.

3.1 Strawman Pollard rho proof of work proposal

In our rho-inspired proof of work scheme, workers compute a pseudorandom walk from a starting point partially determined by the input challenge and produce a distinguished point. The parameters defining the group G, group generator g, discrete log target h, and deterministic pseudorandom walk function $\mathcal{W}$, are global for all workers and chosen prior to setup. A distinguished point x at difficulty d is defined as having d leading zeros in the bitwise representation, where d is a difficulty parameter provided by the challenge generator.

In the terminology of Definition 1, Gen produces a challenge bit string c; when used in a blockchain, c can be the hash of the previous block.

To execute the Solve function, miners generate a starting point for their walk, for example by generating a pair of integers $(a_0, b_0) = H(c||n)$ where n is a nonce chosen by a miner and H is a cryptographically secure hash function, and computing the starting point $P_0 = g^{a_0}h^{b_0}$. Workers then iteratively compute $P_i = \mathcal{W}(P_{i-1})$ until they encounter a distinguished point $P_D = g^{a_D}h^{b_D}$ of difficulty d, and output $\pi = (n, a_D, b_D, P_D)$. A single prover expects to take $O(2^d)$ steps before a distinguished point is encountered.

It is tempting to hope that Verify can simply verify that $P_D = g^{a_D}h^{b_D}$ and has d leading zeros. This confirms that P_D is distinguished, but does not verify that the point P_D lies on the random walk of length ℓ starting at the point determined by (a_0, b_0). Without this check, a miner can pre-mine a distinguished point and lie about its relationship to the starting point. A verifier can prevent this by verifying every step of the random walk, but this does not satisfy the efficiency constraints of Definition 1.

As described in Section 2.2, a discrete log in a group of order q takes $\sqrt{q}$ steps to compute. A single honest prover expects to perform 2^d work per proof and

store $\sqrt{q}/2^d$ proofs, for a total of $\sqrt{q}$ work. If m honest miners are working in parallel, they will perform $O(2^d)$ expected work together per proof. If all miners have equal computational power, the winning miner will find a distinguished point after expected $O(2^d/m)$ individual work. This construction expects to store $\sqrt{q}m/2^d$ distinguished points in a block chain before a collision is found; the total amount of work performed by all miners for all blocks to compute the discrete log is $\sqrt{q}m$. For each distinguished point, $(m-1)/m$ of the work performed by miners is wasted, since it does not contribute to the published distinguished point.

We next examine several modified proof-of-work schemes based on this idea that attempt to solve the problems of verification and wasted work.

3.2 Reducing the cost of wasted work

To reduce wasted work, we can allow miners that do not achieve the first block to announce their blocks and receive a partial block reward. One technique is to use the Greedy Heaviest-Observed Sub-Tree method [29] to determine consensus, which has been adopted by Ethereum in the form of Uncle block rewards [15].

In this consensus method, the main (heaviest) chain is defined as the sub-tree of blocks containing the most work, rather than the longest chain. This allows stale blocks to contribute to the security of a single chain, and allocates rewards to their producers. In Ethereum, this supports faster block times and lowers orphan rates [11], but we could use it to incentivize miners to publish their useful work rather than discard it when each new block is found.

3.3 Limiting the length of the pseudorandom walk

We attempt to reduce the cost of the Verify function by limiting the length of the random walk in a proof to at most 2^ℓ steps for some integer ℓ. Individual miners derive a starting point from the challenge c and a random nonce n. They walk until they either find a distinguished point or pass 2^ℓ steps. If no distinguished point has been found within 2^ℓ steps, the miner chooses another random nonce n and restarts the walk from the corresponding new starting point.

Solve requires miners to produce a proof $\pi = (n, \mathcal{L}, a_D, b_D)$ that satisfies four criteria: (1) the walk begins at the point found by hashing the nonce with the hash of the previous block ($(a_0, b_0) = H(c||n)$) (2) walking from this initial point for $\mathcal{L}$ steps leads to the specified endpoint ($\mathcal{W}^{\mathcal{L}}(g^{a_0}h^{b_0}) = g^{a_D}h^{b_D}$) (3) the bitwise representation of the endpoint $g^{a_D}h^{b_D}$ is distinguished and (4) the walk does not exceed the maximum walk length ($\mathcal{L} < 2^\ell$). An individual miner expects Solve to run in time $O(2^d)$.

Verify retraces the short walk and runs in $O(2^\ell)$ steps.

Analysis. Overall, fixing a maximum walk length forces more total work to be done, since walks over 2^ℓ steps are never published. The probability that a length 2^ℓ random walk contains a distinguished point of difficulty d is $2^{\ell-d}$, so

a prover expects to perform $2^{d-\ell}$ random walks before finding a distinguished point. An individual prover in a group of order q can expect to store $O(\sqrt{q}/2^{\ell})$ distinguished points before a collision is found. With 2^d work performed per distinguished point stored, the total amount of work is $O(2^{d-\ell}\sqrt{q})$. For $m \ll 2^{d-\ell}$ miners working in parallel, the work wasted by parallel mining is subsumed by that of discarded long walks.

Bitcoin miners currently compute around 2^{90} hashes per year To target a 160-bit group, the total amount of work performed by miners would be $2^{90} < 2^{d-\ell}2^{80}$, or $10 < d - \ell$, with a total of $2^{80-\ell}$ distinguished points. If we allow 1 GB = $8 \cdot 10^9$ storage[4], this allows up to 2^{25} 160-bit distinguished points, so we have $\ell = 55$, and thus we set the difficulty $d = 65$. This amount of work is is feasible: at Bitcoin's current hash rate, miners produce nearly 2^{75} hashes per block.

3.4 Efficiently verifying pseudorandom walks

In theory, a SNARK [7] solves the efficient verification problem for the proof of work. Provers would compute the SNARK alongside the pseudorandom walk, and include the SNARK with the proof of work. Verification can be done in constant time. Unfortunately, generating a SNARK is thousands of times more expensive than performing the original computation. Verifiable delay functions [9] could also be used to solve this problem, but existing solutions appear to take advantage of algebraic structure that we do not have in our pseudorandom walk.

We attempted to emulate a verifiable delay function by defining an alternate pseudorandom walk. We experimented with several possibilities, for example a "rotating" walk that performs a set of multiplications and exponentiations in sequence. A walk of this type has the convenient algebraic property that it is simple to verify for a given start point, end point, and length $\mathcal{L}$, that the end point is $\mathcal{L}$ steps from the start. Unfortunately, this walk has terrible pseudorandom properties: collisions are either trivial or occur after $O(q)$ steps.

There appears to be a tension between the pseudorandomness properties required for the Pollard rho algorithm to achieve $O(\sqrt{q})$ running time and the algebraic structure that would allow for efficient verification of the walk. The random walk has the property that each step is determined by the bitwise representation of a given element independent of its group element representation $g^{a_i}h^{b_i}$, but this independence makes it difficult to reconstruct or efficiently summarize the group steps without repeating the entire computation. We leave the discovery of a pseudorandom walk that satisfies these criteria to future work.

3.5 Distributed verification

An alternate block chain formulation has miners accept blocks unless they see a proof that it is invalid, and incentivizes other validators to produce such proofs. This technique has been proposed for verifying off-chain transactions in

[4] Bitcoin's blockchain is roughly 183 GB as of Sep 2018

Ethereum Plasma [27]. We extend this idea to allow validators to prove a miner has submitted an invalid block and offer rewards for such discoveries.

In this scheme, the Verify function accompanies a *reject* decision with a proof of falsification f, and can take as long as mining: $\tilde{O}(t(n))$. We define a function $\mathsf{Check}(c, f)$ to check whether this proof of falsification is accurate, which runs in time $\tilde{O}(n)$. In a block chain, miners Solve proofs of work and dedicated verifiers Verify. If a verifier produces a proof of falsification f (that is, finds an invalid block) it broadcasts (c, f) to all participants, who must Check the falsification.

To increase verification cost, there must be a matching increase in incentive. One option has miners offer a bounty when they produce a new block. After a specified amount of time, the miner receives both the bounty and the block reward. A verifier who publishes a valid falsification before the time limit can collect a portion of the miner's bounty; the remaining bounty and the entire block reward are destroyed. This scheme aims to prevent collusion between miners and verifiers to collect rewards and bounty for no useful work. Next, we present two ideas for distributed verification and outline the limitations of each approach.

Walk summaries. A first idea modifies the proof of work π to include intermediate points spaced at regular intervals along the walk. The Verify function picks a random subset of these intermediate points and retraces the shorter walks between them. An invalid proof must have the property that at least one interval does not have a valid path between the endpoints. For a walk with I intervals of length ℓ, a verifier that checks k intervals has probability k/I of detecting an invalid proof with work kI. However, checking a claimed falsification f requires ℓ work. A lying verifier can force other participants to perform arbitrary amounts of work by reporting incorrect falsifications. To fix this, we need to have more succinct or efficiently checkable falsifications.

Bloom filters for secondary validation. One approach to efficiently checkable proof falsifications uses Bloom filters [8], a probabilistic data structure that tests set membership. It may return false positives, but never false negatives. We modify our walk summary proof of work π above to also include a Bloom filter containing every point on the walk. The Verify function chooses a random interval and takes ℓ walk steps. If an element e_i on the walk is not present in the filter, the verifier broadcasts the sequence of points $f = (e_{i-k}, \ldots, e_i)$. Verification takes work ℓ. The Check function confirms that the broadcast points are a correctly generated random walk and that all points except e_i are contained in the Bloom filter. This takes time k. The short sequence prevents a malicious verifier from invalidating a correct block by taking advantage of false positives in Bloom filters.

A Bloom filter containing every element in a random walk for a reasonable difficulty value will be too large (we estimate at least 150 TB for a walk of length 2^{60}). To shrink the filter, we could store hashes of short sub-walks of length ℓ', rather than every step. To Check, a participant must walk ℓ' steps for each of the k broadcast sub-walks. This increases the runtime to $k\ell'$, but decreases Bloom filter size by a factor of ℓ'.

Acknowledgements

Joseph Bonneau, Brett Hemenway, Michael Rudow, Terry Sun, and Luke Valenta contributed to early versions of this work. This work was supported by the National Science Foundation under grants no. CNS-1651344 and CNS-1513671 and by the Office of Naval Research under grant no. 568751.

References

1. SpaceMint: A cryptocurrency based on proofs of space. In: FC'18. Springer (2018)
2. Back, A.: Hashcash-a denial of service counter-measure (2002)
3. Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Proofs of work from worst-case assumptions. In: CRYPTO 2018. Springer International Publishing (2018)
4. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In: EUROCRYPT'14 (2014)
5. Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R.: SP 800-56A Revision 3. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. National Institute of Standards & Technology (2018)
6. Biryukov, A., Pustogarov, I.: Proof-of-work as anonymous micropayment: Rewarding a Tor relay. In: FC'15. Springer (2015)
7. Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinstein, A., Tromer, E.: The hunting of the SNARK. Journal of Cryptology **30**(4) (2017)
8. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (Jul 1970). https://doi.org/10.1145/362686.362692
9. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Annual International Cryptology Conference. pp. 757–788. Springer (2018)
10. Bos, J.W., Kaihara, M.E., Kleinjung, T., Lenstra, A.K., Montgomery, P.L.: Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. International Journal of Applied Cryptography **2**(3) (2012)
11. Buterin, V.: Uncle rate and transaction fee analysis, `https://blog.ethereum.org/2016/10/31/uncle-rate-transaction-fee-analysis/`
12. Certicom ECC challenge (1997), `http://certicom.com/images/pdfs/challenge-2009.pdf`, Updated 10 Nov 2009. Accessed via Web Archive
13. Diffie, W., Hellman, M.: New directions in cryptography. IEEE transactions on Information Theory **22**(6), 644–654 (1976)
14. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Annual International Cryptology Conference. pp. 139–147. Springer (1992)
15. Ethereum Project: Ethereum white paper, `https://github.com/ethereum/wiki/wiki/White-Paper\#modified-ghost-implementation`
16. Gordon, D.M.: Discrete logarithms in GF(P) using the number field sieve. SIAM J. Discret. Math. **6**(1), 124–138 (Feb 1993). https://doi.org/10.1137/0406010
17. Jakobsson, M., Juels, A.: Proofs of work and bread pudding protocols. In: Secure Information Networks, pp. 258–272. Springer (1999)
18. King, S.: Primecoin: Cryptocurrency with prime number proof-of-work (2013)
19. Kleinjung, T., Diem, C., Lenstra, A.K., Priplata, C., Stahlke, C.: Computation of a 768-bit prime field discrete logarithm. In: EUROCRYPT'17. Springer (2017)
20. Lepinski, M., Kent, S.: Additional Diffie-Hellman groups for use with IETF standards. RFC 5114, RFC Editor (2008), `http://rfc-editor.org/rfc/rfc5114.txt`

21. Lochter, M.: Blockchain as cryptanalytic tool. Cryptology ePrint Archive, Report 2018/893 (2018), https://eprint.iacr.org/2018/893.pdf
22. Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: Repurposing Bitcoin work for data preservation. In: 2014 IEEE S&P. pp. 475–490. IEEE (2014)
23. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. White paper (2008)
24. National Institute of Standards and Technology: FIPS PUB 186-4: Digital Signature Standard (DSS). National Institute of Standards and Technology (Jul 2013)
25. Percival, C., Josefsson, S.: The scrypt password-based key derivation function. RFC 7914, RFC Editor (Aug 2016), http://rfc-editor.org/rfc/rfc7914.txt
26. Pollard, J.M.: Monte carlo methods for index computation (mod p). In: Mathematics of Computation. vol. 32 (1978)
27. Poon, J., Buterin, V.: Plasma: Scalable autonomous smart contracts (2017)
28. Shanks, D.: Class number, a theory of factorization, and genera. In: Proc. of Symp. Math. Soc., 1971. vol. 20, pp. 41–440 (1971)
29. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. In: FC'15. pp. 507–527. Springer (2015)
30. Teske, E.: Speeding up Pollard's rho method for computing discrete logarithms. In: ANTS-III. pp. 541–554. Springer-Verlag, Berlin, Heidelberg (1998)
31. Valenta, L., Adrian, D., Sanso, A., Cohney, S., Fried, J., Hastings, M., Halderman, J.A., Heninger, N.: Measuring small subgroup attacks against Diffie-Hellman. In: NDSS (2017)
32. Valenta, L., Sullivan, N., Sanso, A., Heninger, N.: In search of CurveSwap: Measuring elliptic curve implementations in the wild. In: EuroS&P. IEEE (2018)
33. Van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of cryptology **12**(1), 1–28 (1999)
34. de Vries, A.: Bitcoin's growing energy problem. Joule **2**(5), 801–805 (2018)
35. Wenger, E., Wolfger, P.: Harder, better, faster, stronger: elliptic curve discrete logarithm computations on FPGAs. Journal of Cryptographic Engineering (2016)
36. Wiener, M.J., Zuccherato, R.J.: Faster attacks on elliptic curve cryptosystems. In: International workshop on selected areas in cryptography. Springer (1998)
37. Wustrow, E., VanderSloot, B.: DDoSCoin: Cryptocurrency with a malicious proof-of-work. In: WOOT (2016)