VRF-based mining: How to make pooled mining impossible?

Runchao Han
Monash University and
CSIRO-Data61
runchao.han@monash.edu

Haoyu Lin
chris.haoyul@gmail.com

Jiangshan Yu
Monash University
jiangshan.yu@monash.edu

1 INTRODUCTION

Bitcoin [17] started the era of cryptocurrency. Bitcoin's main novelty is Nakamoto consensus - the first consensus protocol that can work in permissionless settings. Due to the similarity, participants of Nakamoto consensus are called miners, the process of solving PoW puzzles called mining, and the computation power used for mining is known as mining power.

Today, however, mining pools dominate the mining power of most blockchains using Nakamoto consensus, which contradicts the design objective of Bitcoin - the decentralisation. Mining pool is a kind of service that gathers mining power from solo miners and rewards solo miners in a more fine-grained way. To date (01/12/2019), four largest mining pools control more than 51% Bitcoin mining power [2], which can be a lurking threat of Bitcoin.

In this work, we introduce *VRF-based mining*, a novel idea of using Verifiable Random Functions (VRFs) rather than hash functions for mining in PoW-based consensus. With VRF, a miner should use his private key to mine blocks, so a pool operator should reveal his private key in order to outsource the mining process to miners. In this way, no one can outsource the mining process, otherwise any miner can steal all cryptocurrency in his wallet anonymously.

2 VERIFIABLE RANDOM FUNCTIONS

Verifiable Random Function (VRF) [15] is a public-key version of cryptographic hash function. In addition to the input string, VRF involves a pair of a secret key and a public key. Given an input string and a secret key, one can compute a hash. Anyone knowing the associated public key can verify the correctness of the hash, and can also verify the hash is generated by the owner of the secret key. Formally, a VRF consists of four algorithms: VRFKeyGen, VRFHash, VRFProve and VRFVerify.

- $(sk, pk) \leftarrow \mathsf{VRFKeyGen}(1^\lambda)$: on input a security parameter 1^λ, outputs the secret/public key pair (sk, pk).
- $\beta \leftarrow \mathsf{VRFHash}(sk, \alpha)$: on input sk and an arbitrary-length string α, outputs a fixed-length hash β.
- $\pi \leftarrow \mathsf{VRFProve}(sk, \alpha)$: on input sk and α, outputs the proof π for β.
- $\{0, 1\} \leftarrow \mathsf{VRFVerify}(pk, \alpha, \beta, \pi)$: on input pk, α, β, π, outputs the verification result 0 or 1.

3 VRF-BASED MINING

Instead of using a hash function, VRF-based mining uses a VRF to produce hashes that satisfy the difficulty requirement. Different from hash functions, VRF takes both an input and a secret key to produce a hash. In addition, the owner of this secret key can produce a proof proving that the hash is generated by himself. Thus, to outsource the mining process, the pool operator should show his secret key to miners. However, with the secret key, any miner in the mining pool can steal all coins of the pool operator.

Algorithm 1: Work($sk, t, Target$).

```
Input: The secret key $sk$, the block template $t$, and the
       difficulty parameter $Target$
Output: The block $blk$, the VRF output $h$, and the VRF
        proof $\pi$
Initialise $n, h, blk$                       ▷ Initialise variables
while $n \leftarrow$ NextNonce() do          ▷ Refresh the nonce
    $blk \leftarrow$ ConstructBlock($t, n$)     ▷ Assemble the block
    $h \leftarrow$ VRFHash($sk, blk$)           ▷ Produce the VRF output
    if $h < Target$ then                     ▷ If meeting difficulty
        break                                ▷ Mining successful
    end
end
$\pi \leftarrow$ VRFProve($sk, blk$)            ▷ Produce the proof
return $blk, h, \pi$                         ▷ Return block, hash and proof
```

Algorithm 2: Verify($blk, h, \pi, Target$)

```
$pk \leftarrow blk.txs[0].scriptPubKey$   ▷ Find pubkey in coinbase tx
require($h < Target$)                    ▷ Hash should meet diff requirement
/* Here VRFVerify(·) ensures:                                  */
/* 1. $h$ is generated by the owner of $sk$                    */
/* 2. $h$ is the valid output of VRFHash($sk, blk$)            */
require(VRFVerify($pk, blk, h, \pi$))
...  ▷ Verify other fields
...  ▷ Verify transactions
```

Cryptocurrency mining consists of two components, namely mining blocks and verifying blocks. We call the process of mining a block Work, and the process of verifying a block Verify. Algorithm 1 and 2 describe Work and Verify, respectively.

Work. Miners run Work to mine new blocks. More specifically, a miner - with his private key sk the block template (a complete block without nonce) t - keeps searching for a nonce n that can make the (VRF) hash h of the block blk to meet the difficulty requirement $Target$. Once finding a valid n, the miner produces the proof π of h, and appends blk (with n), h and π to the blockchain.

Verify. Upon incoming blocks, miners run Verify to verify their validity. While other verifications are the same as in hash-based mining, Verify in VRF-based mining should additionally run VRFVerify($\cdot$) to verify 1) whether h is produced by the owner of sk, and 2) whether h is a valid output of VRFHash(sk, blk).

Block structure. Different from hash-based mining, in VRF-based mining the blockchain should additionally attach h and π, but does not need the signature of a block. Other miners without knowing sk cannot produce h, but can use π to verify h is generated by someone knowing sk. Through proving the authorship of h, π also proves the authorship of the block. Thus, miners only need to sign h, but do not need to sign blocks.

4 NON-OUTSOURCEABILITY ANALYSIS

4.1 Revised definitions

Miller et al. [16] first formally defines cryptocurrency mining and non-outsourceability. In particular, they define two levels of non-outsourceability, namely **Weak Non-outsourceability** and **Strong Non-outsourceability**. **Weak Non-outsourceability** defines the punishment of outsourcing, while **Strong Non-outsourceability** covers both the punishment and the anonymity of malicious miners. We call the property defining the punishment of outsourcing **Punish-mining-reward**. The anonymity of malicious miners defined in [16] is equivalent to **Transaction Unlinkability** [22]: given two arbitrary transactions, distinguishing whether they have the same output address requires knowing the private keys holding their outputs. To steal the mining reward, the miner should create a transaction, of which the input is the mining reward and the output is his owned address. Identifying who steals the mining reward is equivalent to finding out who holds the output address, which can be further reduced to **Transaction Unlinkability**.

4.2 Punish-secret-key-leakage

We introduce **Punish-secret-key-leakage**, a new property of punishment on outsourcing that achieves stronger non-outsourceability than **Punish-mining-reward**. A cryptocurrency mining protocol achieves **Punish-secret-key-leakage** in the following sense: the pool operator should reveal 1) the block template and 2) the private key receiving the coinbase transaction, so that miners can mine in the name of the pool operator.

4.3 Non-outsourceability of VRF-based mining

VRF-based mining achieves both **Punish-secret-key-leakage** and **Transaction Unlinkability**. In VRF-based mining, the pool operator outsources mining requires revealing his private key to miners, which leads to **Punish-secret-key-leakage**. Similar with the construction achieving **Strong Non-outsourceability** in [16], VRF-based mining achieves **Transaction Unlinkability** by allowing a malicious miner to use a freshly generated address to steal cryptocurrency. To receive the stolen cryptocurrency of the pool operator, the malicious miner can create a new address, and construct a transaction of which the stolen cryptocurrency directs to this address. As this new address has no historical transactions, linking the transaction stealing cryptocurrency with other transactions can be impossible. Then the malicious miner can then spend his stolen cryptocurrency anonymously using numerous techniques, such as mixing services [14][9][18][12] and stealth addresses [22].

5 INSTANTIATING VRF

To implement VRF-based mining, one needs to first instantiate the VRF. VRF in [11] has four configurable components: the elliptic curve and three hash functions. In this section, we discuss considerations on choosing these four components for VRF-based mining.

5.1 Elliptic curve

As neither blockchains and VRF limits the choice of elliptic curves, any elliptic curve can be adapted. Among prominent elliptic curves, Curve25519 [4] can be a promising choice. First, Curve25519 works on a prime field with the prime number $2^{255} - 19$, which provides sufficient enumeration space for VRF. Second, Curve25519 supports Ed25519 [5], a fast and secure digital signature algorithm. Last, Curve25519 is compatible with existing blockchains: numerous blockchains and projects using VRF adapt Curve25519 as their underlying elliptic curve.

5.2 $H_1(\cdot)$ (Hashing a string to an elliptic curve point)

$H_1(\cdot)$ is a hash-to-curve function, which should prevent distinguishing behaviours: adversaries cannot learn any pattern of the input from its hash. In addition, the hash-to-curve function used in VRF should be deterministic, otherwise the hash will be unreproducible.

A standardisation document [19] specifies several hash-to-curve functions that fit into different elliptic curves and satisfy our requirements: Icart Method [13], Shallue-Woestijne-Ulas Method [21], Simplified SWU Method [10] and Elligator2 [6]. Within these hash-to-curve functions, Elligator2 is the recommended one for Curve25519.

5.3 $H_2(\cdot)$ (Hashing an elliptic curve point to a string)

$H_2(\cdot)$ hashes an elliptic curve point to a fixed-length string. It can be divided to two steps: 1) encoding an elliptic curve point to a string, and 2) hashing the string using a normal hash function. The encoding step can be bidirectional and unencrypted, so can be done simply by converting a big integer to a string. The hashing step should be cryptographically secure, so can adapt any existing cryptographically secure hash functions.

For ASIC-resistant cryptocurrencies using memory-hard hash functions (e.g., Ethereum [24] and Monero [1]), there are some concerns on choosing hash functions in $H_2(\cdot)$. To make mining memory-hard, VRFHash should be memory-hard. VRFHash of the standardised VRF consists of one $H_1(\cdot)$ hashing, one scalar-point multiplication and one $H_2(\cdot)$ hashing. $H_1(\cdot)$ and the encoding step in $H_2(\cdot)$ can be fast so less possible to become the performance hotspot. The scalar-point multiplication is slow and computation (rather than memory)-intensive. The hash function in $H_2(\cdot)$ is the only possibility to make VRFHash memory-hard, and it should be overwhelmingly slower than the scalar-point multiplication, otherwise the scalar-point multiplication will make VRFHash computation-intensive. This requirement can be achieved by repetitively executing one or multiple different memory-hard hash functions (such as Ethash [23], Equihash [8], and CryptoNight [20]).

5.4 $H_3(\cdot)$ (Normal hash function)

$H_3(\cdot)$ is only used in VRFProve (proving the authorship of hashes) and VRFVerify (verifying the authorship of hashes). The overhead of proving and verification should be minimised, so cryptographically secure hash functions that are designed to be fast (such as Keccak [7] and BLAKE [3]) are suitable for $H_3(\cdot)$.

REFERENCES

[1] Monero - secure, private, untraceable.

[2] Pool stats - btc.com, 2019-11-24.

[3] Aumasson, J.-P., Henzen, L., Meier, W., and Phan, R. C.-W. Sha-3 proposal blake. *Submission to NIST 92* (2008).

[4] Bernstein, D. J. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography* (2006), Springer, pp. 207–228.

[5] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y. High-speed high-security signatures. *Journal of Cryptographic Engineering 2*, 2 (2012), 77–89.

[6] Bernstein, D. J., Hamburg, M., Krasnova, A., and Lange, T. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 967–980.

[7] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques* (2013), Springer, pp. 313–314.

[8] Biryukov, A., and Khovratovich, D. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. *Ledger 2* (2017), 1–30.

[9] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J. A., and Felten, E. W. Mixcoin: Anonymity for bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security* (2014), Springer, pp. 486–504.

[10] Brier, E., Coron, J.-S., Icart, T., Madore, D., Randriam, H., and Tibouchi, M. Efficient indifferentiable hashing into ordinary elliptic curves. In *Annual Cryptology Conference* (2010), Springer, pp. 237–254.

[11] Goldberg, S., Papadopoulos, D., and Vcelak, J. draft-goldbe-vrf: Verifiable random functions.(2017), 2017.

[12] Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., and Goldberg, S. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium* (2017).

[13] Icart, T. How to hash into elliptic curves. In *Annual International Cryptology Conference* (2009), Springer, pp. 303–316.

[14] Maxwell, G. Coinjoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum* (2013).

[15] Micali, S., Rabin, M., and Vadhan, S. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)* (1999), IEEE, pp. 120–130.

[16] Miller, A., Kosba, A., Katz, J., and Shi, E. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 680–691.

[17] Nakamoto, S., et al. Bitcoin: A peer-to-peer electronic cash system.

[18] Ruffing, T., Moreno-Sanchez, P., and Kate, A. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *European Symposium on Research in Computer Security* (2014), Springer, pp. 345–364.

[19] Scott, S., Sullivan, N., and Wood, C. A. Hashing to elliptic curves. Tech. rep., Internet-Draft draft-irtf-cfrg-hash-to-curve-03. Internet Engineering Task …, 2019.

[20] Seigen, M. J., and Nieminen, T. Neocortex, and antonio m. juarez. cryptonight hash function. cryptonote, march 2013.

[21] Ulas, M. Rational points on certain hyperelliptic curves over finite fields. *arXiv preprint arXiv:0706.1448* (2007).

[22] Van Saberhagen, N. Cryptonote v 2.0, 2013.

[23] Wiki, E. Ethash. *GitHub Ethereum Wiki. https://github.com/ethereum/wiki/wiki/Ethash* (2017).

[24] Wood, G., et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper 151*, 2014 (2014), 1–32.