DDoSCoin: Cryptocurrency with a Malicious Proof-of-Work

Eric Wustrow
University of Colorado Boulder
ewust@colorado.edu

Benjamin VanderSloot
University of Michigan
benvds@umich.edu

Abstract

Since its creation in 2009, Bitcoin has used a hash-based proof-of-work to generate new blocks, and create a single public ledger of transactions. The hash-based computational puzzle employed by Bitcoin is instrumental to its security, preventing Sybil attacks and making double-spending attacks more difficult. However, there have been concerns over the efficiency of this proof-of-work puzzle, and alternative "useful" proofs have been proposed.

In this paper, we present DDoSCoin, which is a cryptocurrency with a *malicious* proof-of-work. DDoSCoin allows miners to prove that they have contributed to a distributed denial of service attack against specific target servers. This proof involves making a large number of TLS connections to a target server, and using cryptographic responses to prove that a large number of connections has been made. Like proof-of-work puzzles, these proofs are inexpensive to verify, and can be made arbitrarily difficult to solve.

1 Introduction

Cryptocurrencies rely on proofs-of-work that require miners to spend a large amount of effort to solve a specific puzzle. Solutions to these puzzles, on the other hand, are designed to be inexpensive to verify. In Bitcoin, the proof-of-work is a computational puzzle based on the SHA256 hash function, and miners are tasked with finding a partial preimage. The best way to do this is to iterate over a large number of inputs to the hash function, and check if the output hash satisfies the partial preimage target. In simplified terms, to find an input that hashes to N bits of zeros, the miner must perform on average 2^N hashes. Once such an input is found, other miners and Bitcoin nodes can verify the puzzle solution with a single hash [1].

[1]Bitcoin uses double-SHA256 in its design, but this detail is not important for the purposes of this paper.

Although the proof-of-work used in Bitcoin gives it resistance to sybil attacks, the amount of computational effort carried out collectively by its miners does not contribute to any useful problems besides securing the currency from attack. Indeed, this computational effort is substantial: As of 2016, miners collectively perform roughly 10^{18} (or about 2^{60}) hashes every second [2].

Many see this distributed computation as a colossal waste of CPU resources, and researchers have proposed alternative cryptocurrencies, or altcoins, that aim to have more beneficial proofs-of-work [13, 16] that provide utility beyond securing the underlying currency: finding chains of large primes or proving the archive of data. In this paper however, we investigate going in the opposite direction, and propose an altcoin that has a *malicious* proof-of-work that is externally detrimental.

In particular, we propose a proof-of-work that allows miners to prove they have participated in a distributed denial of service, or DDoS, attack against a particular target. Miners are incentivized to send and receive large amounts of network traffic to and from the target in order to produce a valid proof-of-work. As in other cryptocurrencies, these proofs can be inexpensively verified by others, and the original miner can collect a reward. This reward can be sold for other currencies, including Bitcoin or even traditional currencies, allowing botnet owners and other attacks to directly collect revenue for their assistance in a decentralized DDoS attack.

The malicious "proof-of-DDoS" operates by having miners create a large number of TLS connections to a target webserver, and using the server's signed responses as a proof of connection. In modern versions of TLS, the server signs a client-provided parameter during the handshake, along with server-provided values used in the key exchange of the connection. This allows the client to prove to others that it has communicated with the server. In addition, the signed value returned by the server is not predictable to the client, and is randomly distributed. Thus, clients can use a similar trick as in Bitcoin, and

only report connections that match some rare threshold, such as the signature from the server starts with N bits of zeros. On average, it will take clients 2^N connections to produce such a proof.

Although the malicious proof-of-DDoS only works against websites that support TLS 1.2, as of April 2016, over 56% of the Alexa top million websites support this version of TLS [10]. Furthermore, we expect this number to increase as TLS support becomes more widespread [11].

Using our proof-of-DDoS, we conceptualize a cryptocurrency that uses it, which we call *DDoSCoin*. In addition to using proof-of-DDoS, DDoSCoin allows miners to select the victim servers by consensus using a proof-of-stake protocol. Rather than specify a single website or static list that DDoSCoin miners target, choosing them by consensus allows the choice of who is attacked to be made collectively and fairly by DDoSCoin participants.

Contributions:

- Propose a novel conceptual cryptocurrency, DDoSCoin, whose proof-of-work incentives miners to participate in a DDoS attack, and allows them to prove they have done so
- Describe a way for target victims to be chosen by consensus of its participants
- Implement our proof-of-work function and evaluate its performance and impact
- Discuss several defenses against such a cryptocurrency that potential victims can employ

2 Related Work

Many alternate cryptocurrency protocols have been proposed since the creation of Bitcoin. Although none have yet eclipsed the popularity of Bitcoin, many so-called altcoins have interesting innovations, often in the form of alternate proof-of-work. For example, Litecoin uses scrypt in place of SHA256 in its proof-of-work, with the intention of being "ASIC-resistant" in order to allow anyone to mine competitively with only a CPU [1].

Permacoin is a proposed altcoin that uses a novel *proof-of-storage* [16]. Rather than waste computational resources solving a useless proof-of-work puzzle (such as in Bitcoin), miners are instead incentivized to store parts of a large agreed-upon file. In order to be competitive in Permacoin, miners prove that they can retrieve an assigned portion of the file.

In Primecoin, miners search for special chains of prime numbers [13]. Although Primecoin has found several large Cunningham and bi-twin prime chains, it remains unclear if such primes have any practical use.

TorPath proposes a proof-of-bandwidth altcoin that incentivizes users to participate in the Tor Network [12]. However, it relies on a set of semi-trusted centralized servers to assign Tor circuits to clients. It is also susceptible to collusion and Sybil attacks by participants, however many of these attacks are outside the threat model of TorPath, as it requires performing those attacks against the underlying Tor network. In contrast, DDoSCoin does not require any centralized parties for the proof-of-DDoS, besides trusting existing TLS certificate authorities to validate domains.

Although Bitcoin uses a scripting language for transactions (see Section 3.1), the allowed operations are very limited [17]. Ethereum allows transactions to be *programmable contracts*, whereby a transaction can be specified in a custom scripting language with arbitrary logic, including conditional payments or fee-collection. While the Ethereum Virtual Machine that executes these contracts does not provide external network access to the scripts, it could still be used to implement some parts of DDoSCoin. For example, it is possible to create an Ethereum contract that pays a bounty to anyone that can provide a proof-of-DDoS for a specified target. This would effectively accomplish the same outcome as PAY_TO_DDOS (described in Section 4.3.1) by implementing DDoSCoin's proof-of-DDoS verification.

3 Background

In this section, we review relevant basics of cryptocurrency. For more detail, we refer the reader to other sources [3, 17, 18].

3.1 Cryptocurrency

Bitcoin employs a hash-based proof-of-work protocol to construct a public ledger of transactions. In this protocol, miners attempt to solve a simple hash-based computational puzzle in order to mine blocks. Mining a block nets the miner a reward and the right to specify what new, valid transactions should be included in the public ledger. To mine a block, a miner must find a hash of a block header that is less than an adjustable target. The block header includes the hash of the last block found, the Merkle root of the list of transactions the miner wants to include in this block, and a nonce. The easiest way to solve this puzzle is to iterate the nonce[2] until the hash satisfies the target difficulty. Then, the miner publishes the found block, and miners use this block as the latest block.

Each block commits to a set of transactions, where each must be valid for the block to be considered valid. Transactions in Bitcoin are scripts, and each new transaction specifies an *output script* that must be satisfied in order for

[2] or modify transactions, thereby changing the Merkle root

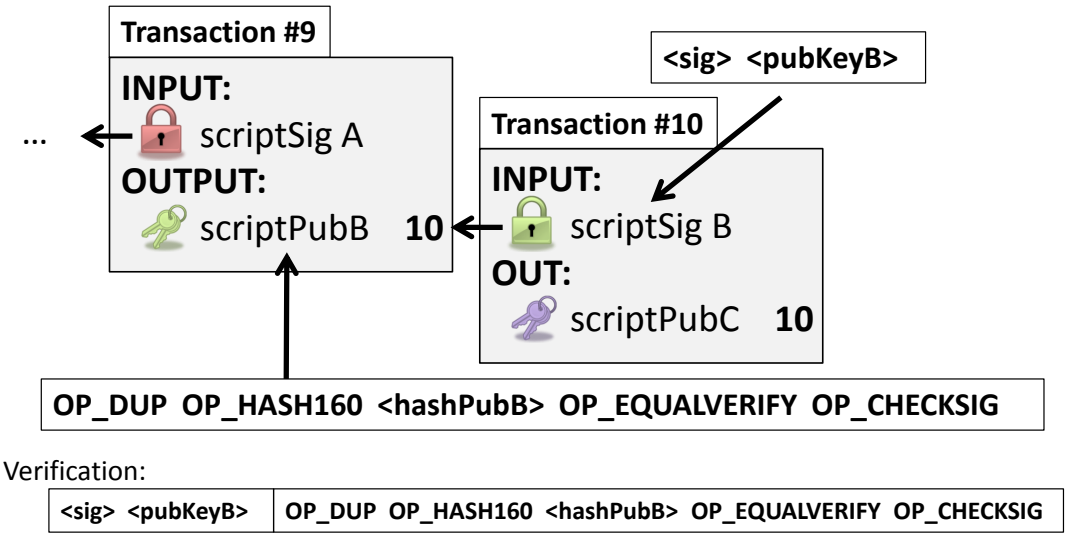

Figure 1: **Standard Bitcoin transaction script** — Transaction #10 spends #9 by providing a valid input script that when prepended to Transaction #9's output script, executes to produce a valid output. The output script stores a hash of a public key (verified using the OP_DUP OP_HASH160 and OP_EQUALVERIFY opcodes), and requires a valid signature over the spending transaction (verified via OP_CHECKSIG).

the coins to be spent. For example, an output script might specify that it requires a spending transaction to provide a valid signature, with the output script specifying the public key to validate it with. To spend such a transaction requires creating a new transaction that references the previous one, and provides an input (scriptSig) that "solves" the previous transactions' output (scriptPubKey). In this case, a valid signature over the new transaction. This new transaction can then specify a new output (scriptPubKey) for these coins. Figure 1 shows an example of a standard Bitcoin transaction script.

3.2 Proof-of-Stake

Proof-of-Stake is an alternate way of generating new blocks in a cryptocurrency. In this system, blocks are "minted" (rather than mined) based on how much stake the miner has in the currency (rather than their fraction of computational power). Proof-of-stake assumes that the underlying cryptocurrency is already somewhat distributed, for example through prior use of a traditional proof-of-work system. There are several variants on proof-of-stake; in this subsection, we will describe Peercoin, an active alternate cryptocurrency that employs a hybrid proof-of-work and proof-of-stake algorithm to mint blocks. For more details on proof-of-stake, we refer the reader to the Peercoin [14] and NXT [5] whitepapers.

Peercoin's proof-of-stake allows currency holders to mint new blocks based on how many coin-days they are able and willing to consume. "Coin-days consumed" is a measure of the product of the amount of coins spent in a transaction, and the number of days since they were last spent. For example, to consume 10 coin-days, a currency holder can spend 10 Peercoin that they have held for a day, or equivalently, spend 5 Peercoins that were last spent 2 days ago. Note that the holder can send these coins to anyone—including themselves—in order to consume the coin age.

To mint a new block, a minter creates a special transaction, presumably one where the minter pays themself, where one of the inputs has been unspent for at least 30 days. The resulting transaction, along with a timestamp, must meet a certain hash target that gets easier proportional to how many coin-days are consumed by the transaction. For example, if the current difficulty gives a minter consuming 100 coin-days a 1% chance of minting a block per second, a minter that was able to consume 200 coin-days would have a 2% chance per second.

The input to the hash includes information about the "aged" transaction, as well as a current timestamp[3]. If this hash is less than the current target multiplied by the number of coin-days consumed in the transaction, then the transaction meets the proof-of-stake difficulty and a new block is minted. Although this looks similar to a proof-of-work check where a hash is being compared to a target, here the only varying inputs are the long-lived

[3] In recent versions of Peercoin, the hash also includes a stakeModifier which is computed from recent blocks; however, these details are not important for our purposes

input transaction, and a course-grained timestamp with a resolution of 1 second. Thus, for each aged unspent transaction output they control, minters perform only one hash per second in order to solve the proof-of-stake.

In Peercoin, coins cannot age beyond 90-days; coins that have not been consumed after 90-days are counted as 90-day-old coins. This prevents very old coin-holders from being able to disrupt the blockchain at a future date, potentially causing a large fork of the blockchain and destabilizing it.

When a proof-of-stake block is minted, the minter is allowed to collect a small subsidy proportional to the coin-age consumed in their proof-of-stake transaction. This serves as the only reward for a minter, as transaction fees are not collected by minters in Peercoin.

4 DDoSCoin Design

Miners in DDoSCoin repeatedly create connections to a TLS victim server, and check for a response that satisfies a target difficulty decided by the network. If the response satisfies this condition, then parameters of the TLS handshake can be published by the miner to create a new valid block. This connection is depicted in Figure 2.

To begin, a miner generates a random 32-byte secret nonce, N. The miner also selects the latest block in the chain it is mining on, and the set of transactions that the miner intends to include in this block, both as is done in Bitcoin. The miner then computes:

$$SHA256(prev\_block||merkle\_root||N) \qquad (1)$$

to generate the 32-byte `client_random`, where *prev_block* is the SHA256 hash of the previous block header, *merkle_root* is the Merkle root of transactions as computed in Bitcoin, and N is the random nonce.

The miner initiates a TLS version 1.2 [8] connection with the victim server, using the `client_random`, and choosing a set of cipher suites that will ensure the server will send a server key exchange message. This includes ephemeral Diffie-Hellman (DHE), and ephemeral elliptic curve Diffie-Hellman (ECDHE). We note that for the default cipher suite list sent by Google Chrome, over 94% of TLS 1.2 servers in the Alexa top million choose a cipher suite with a server key exchange message [10], compatible with DDoSCoin.

The server will respond with a server hello message, containing a 32-byte `server_random`, the cipher suite that will be used, and other parameters that are not needed for the purposes of DDoSCoin.

Next, the server sends its certificate chain, and a server key exchange message. The certificate chain contains the server's signed TLS certificate, as well as intermediate certificate authorities that create a signature chain to a browser-trusted root CA. Inside the server's certificate is a public key, as well as the identity of the server represented by it's domain. The miner stores these for verification in case a block is found.

In the server key exchange message, the server sends key exchange parameters such as the Diffie-Hellman parameters and the ephemeral public value generated by the server, or the curve parameters and a public curve point in ECDHE. The server signs these key exchange parameters, along with the `client_random` and `server_random`. This signature can be verified with the public key in the certificate.

Since the server key exchange signature contains the `client_random` and `server_random`, the signature value will be unpredictable to the client ahead of time. In addition, each signature will be a random value that depends on the miner-provided nonce. Thus, it can be used as a proof-of-work.

If the SHA256 hash of the server key exchange parameters, signature, and miner-chosen nonce N is less than the current target difficulty, then this connection satisfies the proof-of-work threshold, and can be used to form the next block. We hash N a second time in this way to prevent the victim server from being able to tell if their response would satisfy the target difficulty. Otherwise, servers could simply withhold such infrequent responses, and no blocks could be published for them.

To make a block, the miner publishes the previous block's hash, the transactions (and their Merkle root), the nonce N, the `server_random` value, the server's certificate chain, the server key exchange parameters, and the server key exchange signature.

4.1 Validating Blocks

To verify that a block is valid, miners must validate several items. First, they must validate that the previous block hash and Merkle roots are valid, as in Bitcoin. Next, they must re-create the `client_random`, by hashing the previous block hash, transaction Merkle root, and provided nonce. Then, they must validate the certificate actually belongs to a valid victim server. This is done by checking that the certificate chain is valid, rooted in a trusted certificate authority, and the domain name is in a list of acceptable victims. This list of valid victim domains is generated by consensus, and details for its generation are discussed in Section 4.3.

After the victim has been verified to be a valid one, the public key is used to verify the server key exchange signature. This signature is over the `client_random`, `server_random`, and server key exchange parameters.

Finally, the block must be verified to meet the current target difficulty. The server key exchange parameters, signature, and the block's nonce (N) are hashed using

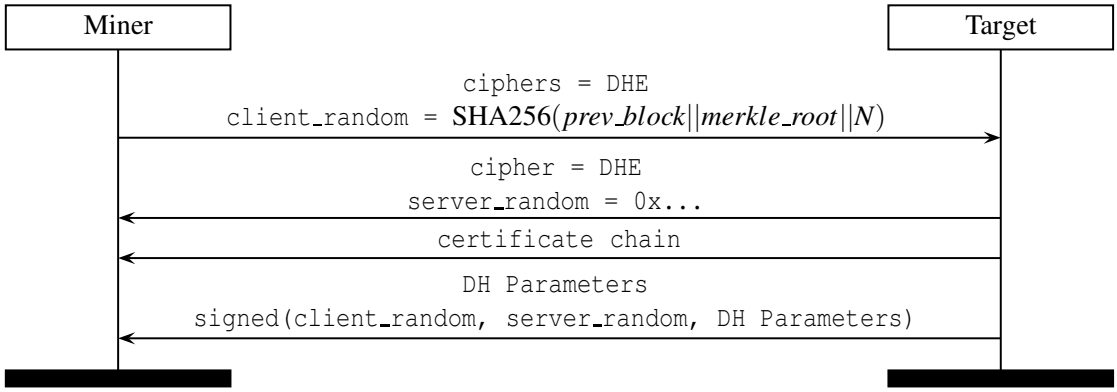

Figure 2: **Miner–Target Interaction** — A single round-trip is required for each attempt to solve the proof-of-work. The client computes the `client_random` to commit to a given previous block and set of transactions, and only provides the server the option to use Diffie-Hellman Ephemeral cipher-suites. The server then calculates a signature dependent on the `client_random` and verifiable by the certificate the server provides.

SHA256, and compared to the current target difficulty. If it is less than the target, the block is a valid one and becomes the latest block in this chain.

4.2 Earlier TLS Versions

DDoSCoin is incompatible with versions of TLS before 1.2 (including SSL) because the server key exchange signature does not include the `client_random` (or any client-provided values). In earlier versions, the only value that provably comes from the server (i.e. is signed by a key tied to the victim's identity) does not contain any commitments from the client about the previous block, or what transactions should be included in the block. If we were to accept blocks where there was only a server signature that met some target difficulty, any miner could steal another miner's found block (and the contained block reward) by changing the transactions and forwarding the block.

If we were to somehow tie the previous block's hash and Merkle root into this proof, for example, by saying that the hash of the signature and a client-provided nonce had to be less than some target value, the proof-of-work becomes a computational puzzle: a miner only has to collect a single signature from the server, and try different nonces until the hash of signature and nonce meets the target difficulty.

Similarly, proof-of-work target difficulties can not be based off of the purported shared secret that results from a connection, because the client can always choose different key exchange values to ensure that the resulting shared secret is less than than the threshold difficulty, resulting in a computational puzzle. Even values sent by the server—such as the Finished message that contains a MAC dependent on the shared secret—can be (re)computed by the client.

As an aside, this interesting property gives rise to an observation that with respect to clients, the contents of TLS connections are *deniable*. That is, it is not possible for a client to record a TLS connection, including its chosen secrets, and report this to a third party in a way that cryptographically proves what the server says. This property has been accomplished in different more efficient ways in various secure messaging protocols such as Off-the-Record (OTR) [4].

Besides TLS 1.2, there are other cryptographic protocols that involve the server signing a client-chosen value. For example, the SSH protocol key exchange involves the server signing a hash over the previous handshake messages, including the client-provided identification (version) string, and Diffie-Hellman public value [21]. Either of these fields could be used to encode the necessary commitments (previous block, Merkle root, and nonce) used in DDoSCoin. However, SSH server identities are generally not signed by a trusted party, making verification that a miner has communicated with a particular server more challenging. Nonetheless, identities in an SSH-based DDoSCoin could simply be the server's host key rather than a domain in a CA-signed certificate.

4.3 Victim Selection

If DDoSCoin accepted proofs against any TLS 1.2 server, miners would be incentivized to set up their own TLS servers locally, and focus exclusively on "attacking" those servers instead of remote ones. Since the miner would have access to the private key for that server, they would not be required to even create network connections, reducing the proof-of-work to a computational puzzle rather than one based on participation in a denial of service over

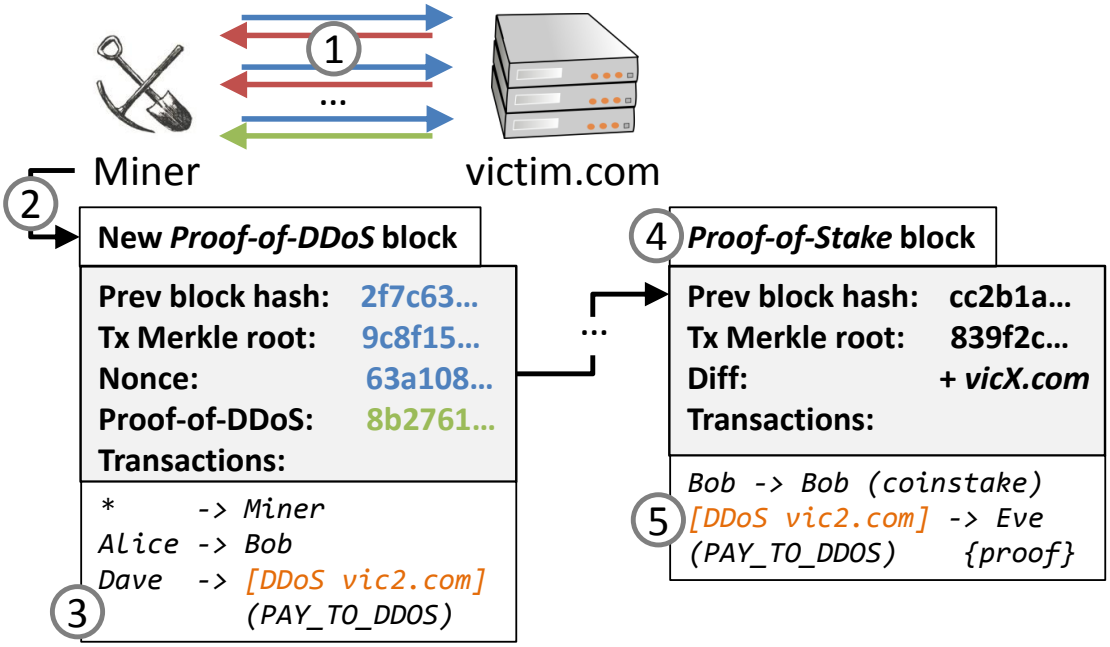

Figure 3: **DDoSCoin Design** — DDoSCoin miners make repeated connections to a victim server running TLSv1.2 ①. In each handshake, the miner commits to the previous block, a transaction merkle root, and a secret nonce. Eventually the victim server will respond with a signed message that meets the current proof-of-DDoS target difficulty, and the miner can create a new block ② that includes this proof. Victim targets can be selected in two ways: First, participants can pay into one-time bounties for specific victims ③, which can be redeemed by anyone in a special PAY_TO_DDOS transaction ⑤ if they can provide a similar proof-of-DDoS against that victim. Second, the list of valid victims that blocks can be mined for can be updated by proof-of-stake blocks ④.

a network. To prevent this, DDoSCoin must have a way to agree on which victims are acceptable targets for the proofs-of-DDoS to be considered valid. Similarly, it is important to focus the denial-of-service effort toward a small number of victims in order to have the most impact.

One simple solution is to hardcode a set of victim identities into the protocol. For example, DDoSCoin's published code could contain the identities (domain names) of Facebook's servers, and only allow and incentivize attacks against them. However, there are many attackers that might feel uncomfortable attacking Facebook, but are willing to participate in attacks against other websites. In addition, having only a single or small set of victim websites is a risk to DDoSCoin: if those websites all go offline or successfully mitigate the attacks, no future DDoSCoin blocks can be mined, and the currency will stall. If this happened, all DDoSCoins in circulation would become nearly worthless, as there would be no way to safely transact them.

These issues lead to a few subtle requirements for choosing valid victims. First, there must be *limits* on who is allowed to be a victim, otherwise miners can create local computational puzzles. At the same time, there must be *enough* targets in order to be robust against stalled blockchains. Similarly, the list of victims must be *flexible* in order to allow removing targets as they become impractical to attack (either through successful attack or mitigation), and adding new ones to take their place.

To address these issues, we propose two mechanisms that DDoSCoin can employ to allow victims to be selected by consensus. The first mechanism allows anyone to commit DDoSCoins to whoever can prove they sent an amount of traffic to a target (PAY_TO_DDOS). The second allows for updating the list of victim domains that miners are allowed to attack when mining for blocks.

4.3.1 Pay-to-DDoS

In order to allow victims to be (temporarily) selected for DoS, DDoSCoin allows "bounties" for targeting specific servers. To accomplish this, DDoSCoin introduces a new payment opcode, PAY_TO_DDOS, that can be used in transactions subject to certain constraints. This opcode takes two arguments in an output script: a string representing a domain name that the payer wishes to have attacked, and a target difficulty corresponding to the amount of connections the payer wishes to be made. Once this transaction is stored in a valid block, anyone can collect its

reward by creating a connection to the specified victim server and obtaining a response that meets the given target difficulty. These connections are made similar to how miners connect to victim servers to create blocks, but with a few important differences. First, the `client_random` is generated as follows:

$$SHA256(txid||output\_script||N) \quad (2)$$

Where *txid* is the transaction ID of the PAY_TO_DDOS transaction, *output_script* is the SHA256 hash of the output script the payee wants to collect the payment with, and *N* is a 32-byte random nonce. The client then makes connections to the specified victim domain until it gets a response that meets the target difficulty specified by the PAY_TO_DDOS transaction, i.e. the hash of the key exchange parameters, signature, and nonce are less than the target hash value.

The bounty poster must select a difficulty that is appropriate for the target and reward. If the difficulty is too high, no miner will attack the target, because the miners do not want to waste effort. Also, if the difficulty is too low, the reward will be claimed for little degradation of service.

To collect the bounty reward, the client creates an input script that will satisfy the PAY_TO_DDOS output. This includes the parameters of the connection: the nonce, the `server_random` value, the server's certificate chain, the server key exchange parameters, and the server key exchange signature. To validate the transaction, a miner must recompute the `client_random` as above. Then, the miner must verify that the certificate is valid for the provided domain (and rooted in a trusted CA), and that the certificate's signature over the server key exchange message is valid. Finally, the server must check that the server's response meets the target difficulty specified by the PAY_TO_DDOS transaction. If it does, then the transaction is valid and can be included in a block, letting the client collect the reward.

There are a couple important features of this transaction. First, the reason the transaction ID of the PAY_TO_DDOS transaction is committed to in the `client_random` is to prevent a client from "double-claiming" multiple bounty rewards with the same proof-of-DoS. Similarly, the client commits to the output script of the claiming transaction to prevent others from being able to swap it for their own, thereby "stealing" the proof-of-DoS and collecting the reward themselves. These properties combined allow two mutually-distrusting parties to directly and safely transact currency in exchange for a specified-amount of work in a DoS attack.

Since PAY_TO_DDOS transactions can be claimed by anyone, there is a risk to the collector that by the time they complete their DoS attack, someone else will have collected the payment. Any target specified by a PAY_TO_DDOS transaction could use their privileged position to turn the proof-of-work into a computational challenge. Since the challenge is no longer bound by the network, the target can claim the reward for themselves. Quickly solved PAY_TO_DDOS transactions with no degradation of service and sufficiently difficult proof-of-work could indicate to the bounty provider that the target is claiming its bounty.

In addition, a potential victim website could disincentivize attacks by submitting PAY_TO_DDOS transactions that they can quickly solve locally. This would cause attackers to try to connect to them, and eventually even collect a response that meets the target difficulty. However, when the attacker went to publish this to collect the reward, the victim could publish their proof, essentially double-spending the PAY_TO_DDOS transaction. With some probability, the victim's transaction will "win" and prevent the attacker from collecting the reward. Over the long term, attackers may be wary of attacking such targets, and instead require higher payouts for attacking them to cover such risks.

While this new payment type allows a one-time payment for attacking a particular target, it cannot be used to update the long-term victim list used by miners to decide valid victims for blocks. Otherwise, miners could make a PAY_TO_DDOS transaction specifying their own local domain in order to always have a local-computation puzzle that allowed them to create new blocks in the future. Instead, a second method for choosing victims must be employed.

4.3.2 Alternate Proof-of-Work

In order to allow the victim list to be updated dynamically, DDoSCoin uses *Proof-of-Stake* to let currency holders make small changes to the list of acceptable victim domains. Using proof-of-stake ensures that updates are only done by actors that already have a substantial stake in the currency, either by purchasing it or by previously participating in the mining themselves.

To update the list, a currency holder "mints" a proof-of-stake transaction similar to Peercoin. First, they must hold a number of coins that have been unspent for more than 30 days. After the 30 days have passed, each second the minter hashes the aged transaction, a recent stake-modifier block, and a current seconds-resolution timestamp. If this hash is less than the current proof-of-stake target difficulty multiplied by the coin-age being consumed in this transaction, then the currency holder is allowed to make a new proof-of-stake block. Inside this block, the currency holder can make one addition to or one deletion from the current victim list.

Proof-of-stake blocks contains no coinbase or coinstake reward, and transaction fees are not collected. Thus, there

is no direct short-term monetary incentive to produce such a block. The intention is that this will encourage only those with a vested interest in the long-term success of DDoSCoin to participate in updating the victim list.

After this block has been published, miners must wait until 10 proof-of-DoS blocks have been mined with the old list of valid domains before the specified delta to the change list is applied. This discourages malicious miners from forking the chain at an earlier block with a proof-of-stake block adding their local domain to the victim list, and immediately mining blocks on top of it to catch up to the main chain.

4.3.3 Difficulty Adjustment

The difficulty of proof-of-work changes dynamically in DDoSCoin, as it does in other crypto-currencies, in order to limit the rate at which new blocks are mined. However, in DDoSCoin, the difficulty is adjusted separately for each target domain. Each victim domain has its own difficulty since domains could have different maximum throughputs, allowing more or fewer connections from all miners over the same time period. Consider a single domain that handles many connections per second added to the target list. The miners would connect to that server more often, since it will give more chances at mining a block, and thus relieve traffic from the other targets.

DDoSCoin adjusts difficulty for a given domain by looking at the previous BLOCK_ADJUST blocks, and counting the number of blocks mined against the given domain. This ratio is compared to the expected ratio based on the current size of the victim list and the time over which these blocks were mined (based on a timestamp included in the blocks). If more blocks were mined than expected, the difficulty for this domain is increased. Similarly, if fewer blocks were mined, the difficulty for the domain is decreased. This mechanism allows target domains that somehow become more difficult to reach to quickly return to a more appropriate difficulty, without having to wait for many blocks mined at the high difficulty.

A target's initial difficulty in DDoSCoin is decided by the stakeholder that introduces the target in a proof-of-stake block. The stakeholder is incentivized to not set the difficulty too low, as many coins would be mined quickly, devaluing the stakeholder's own coins. If the difficulty is set too high, there will be a delay as other domains are mined, and the difficulty adjusts down to the appropriate level.

4.4 Implementation

To evaluate the performance and effectiveness of DDoSCoin, we implemented our proof-of-DDoS function in 582 lines of C. Our mining code uses libevent [15] and OpenSSL [19] to create concurrent connections toward a specific target, crafting a `client_random` in the specified way and testing each response to see if it satisfies a hard-coded proof-of-DDoS target difficulty.

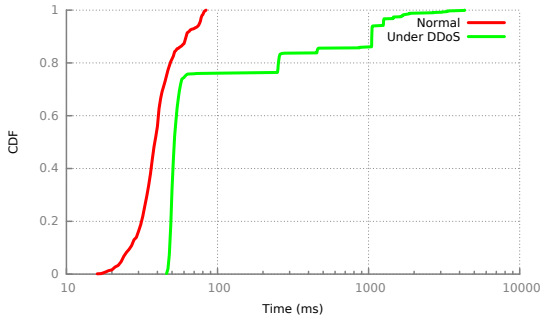

Figure 4: **DDoSCoin effect** — We set up a quad-core nginx HTTPS server on a local network, and used Apache benchmark (ab) to measure page load times under normal conditions (no DDoSCoin miner running) and with a single DDoSCoin instance. DDoSCoin increases the average page-load time 6-fold from 43ms to over 291ms.

We tested our proof-of-DDoS function (miner) against a quad-core TLS server under our control, running nginx serving the default page and connected to our attacker over a local Gigibit Ethernet network. When running our miner, the CPU consumption of the server quickly reaches 100% utilization on all 4 cores, while the attacker only requires approximately 30% on a single core. This produces over 3000 TLS connections per second, and slows response times of the webserver considerably. We used Apache benchmark (ab) with 1000 connections with a concurrency of 50 to quantify this, and tested the server under normal conditions (no miner running) and under DoS (miner running). As shown in Figure 4, we find that the average server response time increases by more than a factor of 6 when our miner is running.

5 Discussion

Many positive results have come from cryptocurrencies. Bitcoin was invented with the goal of creating a decentralized digital currency, and to that end has been more sucessful than most imagined. Banks are turning to "Blockchain technology" to solve issues regarding public ledgers. Primecoin and TorPath use cryptocurrencies to incentivize positive actions. However, this positivity is not inherent to cryptocurrencies. DDoSCoin demonstrates maliciousness inherent to a currency's design, turning the notion of what constitutes a "useful" proof-of-work around.

DDoSCoin uses a proof that the miner has connected to a given server a sufficient number of times as its proof-

of-work. This is an indirect measurement of how much bandwidth or resources are being spent by the victim. Although miners only receive DDoSCoins in return for this effort, these can be traded in for other currency via cryptocurrency exchanges. In addition, because DDoSCoin uses transactions similar to Bitcoin, trust-less (atomic) cross-chain transactions can allow DDoSCoins to be traded for Bitcoins without requiring a trusted 3rd party intermediary. Thus, even outlawing DDoSCoins from exchanges does not prevent a black-market from getting around these sanctions.

An interesting choice in the design of DDoSCoin is target-setting is done dynamically, and by consensus. A given target can go offline, or in some other way defend against DDoSCoin at any point. When this happens, the currency cannot stall, as then no coins could be transacted. This necessitates dynamic target-setting.

While stakeholders have control of the target list, a majority of miners could use their choice of previous block to veto a given proof-of-stake block. This would be a fork of the blockchain. If a clear majority treat a block as malformed because it introduces an undesired domain, then the longer chain wins, and the undesired domain is not included in the target set, and any blocks mined on the chain with the vetoed target are not part of the blockchain. This allows the miners to coordinate and self-enforce a blacklist, or whitelist, of target domains, so long as they have sufficient participation in the policy.

Although not yet implemented, the current TLS 1.3 proposal appears to also have the necessary properties that enable DDoSCoin [20]. In particular, the Certificate Verify message includes a signature from the server over essentially all prior handshake messages, including the Client Hello that contains the `client_random`. Thus, DDoSCoin could be used against TLS 1.3 servers with only minor modifications. This should be considered in the design of TLS 1.3 due to the long life of TLS implementations.

5.1 Ethical Considerations

While we acknowledge this work introduces an idea that could be used to incentivize malicious behavior, we have taken precautions to limit harm. First, in demonstrating our proof-of-concept and evaluating our proof-of-DDoS code, we have only "attacked" websites we have ownership and authority over. Furthermore, we are not publishing a working alt-coin that uses this proof-of-DDoS, but rather a conceptual description of one. We believe it is important to fully disclose potential attacks, even those that require the development of an altcoin to execute. This is especially important in the face of the impending commitment to the design of TLS 1.3, and compounded by how long TLS/SSL protocol versions stay in active use.

5.2 Defenses

Victim websites can use several methods to defend themselves from DDoSCoin miners. First, websites can simply disable TLS 1.2 entirely, and only support earlier versions such as TLS 1.0 [6] and TLS 1.1 [7]. While older versions of TLS 1.0 clients may still be vulnerable to the BEAST attack [9], most modern clients have mitigations for this attack, and it likely remains the safest short-term defense against DDoSCoin. The downside of disabling support for TLS 1.2 is losing the ability to negotiate authenticated encryption cipher modes with clients, which may decrease performance of implementations that have hardware acceleration for such modes.

Alternatively, web servers can disable cipher modes that require signatures from the server. This would remove support for forward secure cipher suites, meaning that as long as the server's private key is accessible, any previously recorded TLS connections could be decrypted. We note that the current drafts of TLS 1.3 propose to remove non-forward secure cipher suites entirely, possibly making it more difficult to defend against DDoSCoin in TLS 1.3.

Although dramatically changing existing deployed versions of TLS is unrealistic, it may still be possible to influence the design of the TLS 1.3 standard to make it incompatible with DDoSCoin. For example, rather than having the server sign all of the handshake messages (including those from the client), the server could sign only the handshake messages which it sent. This may not fully mitigate DDoSCoin, as some of the parameters (such as the selected cipher suite or elliptic curves) may have originally been provided by the client. We note that while the server signing only its own messages still mitigates many previous known bugs such as those revealed in the FREAK TLS attack, there may be other subtle bugs that are exposed by servers not signing client's messages.

Websites could also attempt to thwart DDoSCoin by participating in the mining themselves. Because the victim website will have local access to its private key, they will have an advantage over remote clients. With this advantage, the victim might be able to mine enough DDoSCoins to mint a proof-of-stake block that removes itself from the list, or raise the difficulty of mining a block high enough that a remote miner has a negligible chance of succeeding.

Finally, websites could attempt to go after mining operations legally. Publishing a proof-of-DDoS block in the DDoSCoin blockchain may reveal the intent of a miner to attack a service for financial profit. If the victim webserver is able to log the traffic they receive, they can discover the IP address of the party given a published proof-of-DDoS block. However, as many attackers may use proxies, botnets, or bullet-proof hosting to carry out

their attacks, finding an attacker's IP address may not be sufficient for legal action.

6 Conclusion

In this paper, we have presented a proof-of-DDoS cryptocurrency that allows miners to prove their involvement in sending a large amount of requests to a specified victim webserver. Proof-of-DDoS can be used to replace proof-of-work in a cryptocurrency setting, provided that there is consensus around what victims are valid targets. Our conceptual altcoin, DDoSCoin, provides such a consensus through two mechanisms: PAY_TO_DDOS, whereby a bounty can be set for DDoSing a given victim domain, and proof-of-stake updates to the list of valid victims.

Cryptocurrency innovation continues to produce new and useful proof-of-work replacements. Still, proving access to arbitrary resources remains a difficult challenge. In this direction, DDoSCoin delivers a novel technique for proving the use of bandwidth to a (potentially unwilling) target domain. We hope that this work encourages others to discover and innovate on novel proof-of-resource puzzles.

Acknowledgements

The authors are grateful for the support and various discussions from our colleagues on DDoSCoin. In particular, the authors wish to thank David Adrian, Matt Bernhard, and Drew Springall for their discussions concerning the feasibility of the proof-of-DDoS idea. We also thank the anonymous reviewers for their helpful feedback.

References

[1] Litecoin - open souce P2P digital currency. https://litecoin.org, 2011.

[2] Blockchain.info. Bitcoin hash rate. https://blockchain.info/charts/hash-rate.

[3] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.

[4] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM, 2004.

[5] Nxt Community. Whitepaper:nxt. http://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt, 2013.

[6] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507.

[7] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507.

[8] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685.

[9] T Duong and J Rizzo. Here come the $\oplus$ ninjas. *BEAST attack*, 2011.

[10] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 542–553. ACM, 2015.

[11] Let's Encrypt. Let's Encrypt - free SSL/TLS certificates. https://letsencrypt.org/.

[12] Mainak Ghosh, Miles Richardson, Bryan Ford, and Rob Jansen. A TorPath to TorCoin: Proof-of-bandwidth altcoins for compensating relays. 2014.

[13] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work. *July 7th*, 2013.

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

[15] Nick Mathewson and Niels Provos. libevent: An event notification library. http://libevent.org/.

[16] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing Bitcoin work for data preservation. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 475–490. IEEE, 2014.

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

[18] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies*. 2 2016.

[19] OpenSSL Project. OpenSSL: Cryptography and SSL/TLS toolkit. https://www.openssl.org/.

[20] E. Rescorla. The transport layer security (TLS) protocol version 1.3 draft-ietf-tls-tls13-12, March 2016.

[21] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. Updated by RFC 6668.