PacketCrypt

Bandwidth-hard proof of work.

Caleb James DeLisle
s://github.com/cjdelisle
cjd@cjdns.fr

Abstract— Since proof of work was popularized by the Bitcoin project, there has been active research into ways to make Proof of Work (PoW) useful. Unfortunately it has proven remarkably difficult to make PoW serve humans without allowing miners to influence the nature of the work problem to their own advantage, destroying the fairness of the algorithm.

PacketCrypt takes a different approach, while the work done by PacketCrypt itself is not useful, PacketCrypt attempts to make the PoW very similar to useful work so that research and development into technologies for efficiently mining PacketCrypt will be reusable for other purposes.

PacketCrypt encourages the development of technologies for high speed encryption of internet traffic, it also contains a component using randomly generated code in order to encourage CPU mining and research on highly parallel CPUs. Most importantly, PacketCrypt is parallelizable with n-to-n communication, making it a bandwidth hard proof of work.

I. PacketCrypt Protocol Overview

PacketCrypt protocol consists of two distinct proof of work algorithms, the first is a . It consists of a memory-hard algorithm which creates a O(log(n)) size proof that can be verified without the original data.

The PacketCrypt algorithm is used in two different instances, first the algorithm is used as a simple hashcash to prove work on a small message (called an announcement), then the algorithm is used again to mine a cryptocurrency block where the miner gets a difficulty discount based on the amount of work which had been done on announcements that they managed to collect.

The demand on bandwidth is created by the need to broadcast each announcement from the device which created it to all devices which will use it for attempting to mine a block.

II. PacketCrypt Algorithm

PacketCrypt is an algorithm which takes input data (for example of a block header) and a list of data items where the merkle root of that list has been committed in the input data. It accesses 4 items from the list (determinant random based on the items and input data) and then outputs a hash of the input data and the 4 items. It also outputs a proof containing the 4 data items from the list and a partial merkle tree allowing the verifier to prove that those items existed at the given positions in that list.

Verification of the resulting proof consists using the partial merkle tree to build a sparse list with only those 4 data items and then repeating the process of hashing. If the PacketCrypt algorithm requires a data item which is not present, the verification fails. On success it results in the same hash which was produced by the original mining algorithm.

When searching for a partial preimage, as is done in cryptocurrency mining, the PacketCrypt algorithm must be run millions of times over, each time it is run, it must access data items from the list, making the result an effective proof that the data set was present in the miner's memory at the time that they were mining.

A. What is a time memory tradeoff?

A time memory tradeoff (TMTO) describes the "price" in CPU processing time of reducing memory consumed in computing the solution to a particular problem. While TMTOs can take a number of different forms, we will concentrate on two in particular.

An obvious attack on PacketCrypt is to throw away data or lie about the number of items which you have. Suppose you wanted to pretend you had twice as many items as you really had: Every time you perform a hash operation which seeks an item that is missing, you simply try again. This is what we will call a probabilistic TMTO, because it is based on faking data in the hopes that it will not be needed.

PacketCrypt requires 4 dependent memory accesses so the chance that you can mine with half of the data missing is 1 in 16. The choice of the number 4 is intended to avoid excessively bloating the size of the PacketCryptProof while still significantly penalizing the exercise of the TMTO. It's tempting to think that needing to make 16 hash operations means one needs to perform 16x the CPU effort, but because PacketCrypt performs the memory lookups sequentially, one is able to bail out part way through the hash operation. As a hash operation consists of 5 steps with 4 memory lookups in between, the steps which must be performed to test 16 possibilities with 50% of the memory missing are 8*1 (fail in the first cycle) + 4*2 (fail in the second cycle) + 2*3 (fail in the third cycle) + 1*4 (fail in the forth cycle) + 1*5 (success). This is a total of 31 encryption attempts, which compared to 1*5, the case where all memory is available, requires 6.2x the CPU usage.

There is also another type of TMTO which we'll call a regenerative TMTO because it is based on regenerating the data which was discarded. Colin Percivial's Scrypt, one of the earliest works on memory hard functions, has a fairly nasty regenerative TMTO which requires only 17% increase in processor time to halve the memory. This issue has since been formalized: https://eprint.iacr.org/2015/430.pdf and is the main reason why ASIC Litecoin miners are able to dominate.

While the properties of the probabilistic TMTO are clearly understood for PacketCrypt, the regenerative TMTO is, for PacketCrypt, dependent on the difficulty of generating the data items themselves. Therefore, as long as it takes CPU power to create a data item, we can form a unified definition of time/memory effort used for PacketCrypt. If we conjecture that we have identified all of the degenerate cases, we can informally prove that there exists no method of mining PacketCrypt hashes which is more advantageous than the best one which we identify.

B. *Hash algorithms used*

The PacketCrypt algorithm uses chacha20/poly1305 encryption algorithm to encrypt a 2048 byte buffer multiple times over, while copying each data item over the first 1024 bytes of that buffer each cycle. At the end of each cycle, the poly1305 authenticator and part of the encrypted data as the key for the next encryption cycle as well as for the index of the next data item to access. While this choice of algorithm is clearly unorthodox, the objective is to encourage the development of technology which works for encrypting and sending messages of about the same size as an internet packet.

C. *PacketCrypt as a broadcast network*

PacketCrypt can be used to form a gossip-based broadcast network, where anyone can broadcast a message as long as it contains a minimum amount of proof-of-work. In fact, any type of announcement can be sent on this broadcast network. An Announcement is created by a participant in the network who wants to get their message heard by a large number of other participants. Miners collect Announcements into what we will call an AnnouncementSet which is then used for memory hard mining. By collecting Announcements from network participants, a miner is able to get a discount on the proof of work which they must perform, effectively outsourcing some of the mining effort to those making announcements.

Parties who want to broadcast a message across the network will be able to do so by crafting an Announcement and mining it themselves before releasing it with some work done on it. The layout of an Announcement message is as follows:

TABLE I. ANNOUNCEMENT LAYOUT

Announcement Layouts		
Version	***uint8***	***Version number***
SoftNonce	[3]byte	Nonce used for mining, can be changed without regenerating the mining dataset.
HardNonce	[4]byte	Nonce for mining, any change requires regenerating the mining dataset.
WorkBits	uint32	Bitcoin format compact representation of the max hash for this Announcement
ParentBlockHeight	uint32	Number of the most recent block at the time this Announcement was made
ContentType	uint32	A value which can be used for grouping announcements of a given type (e.g. network state updates)
ContentLength	uint32	The length of the content of the announcement in bytes
ContentHash	[32]byte	If ContentLength is less than or equal to 32, this is the literal announcement content, otherwise it is the root of a

Announcement Layouts		
Version	***uint8***	***Version number***
		merkle tree of 32 byte blocks of the content.
SigningKey	[32]byte	If this is non-zero, the announcement must be signed with this ed25519 key when it is included in a block.
MerkleBranch	[14][64] byte	A branch proving one of the elements in the announcement dataset.
LastItemPrefi x	[40]byte	First 40 bytes of the last item in the dataset.

Fig. 1. This table shows the layout of an announcement. The colors show the point in the announcement hashcash mining cycle when the parts of the announcement are committed. MerkleBranch and LastItemPrefix are not committed at all and are only attached to the announcement as proof, SoftNonce is used in the hashing process but is not committed when building the dataset, the rest of the elements are committed (along with the hash of the block at ParentBlockHeight) when constructing the dataset.

D. Announcement HashCash

The instance of PacketCrypt used for hashing the individual announcements is somewhat special. First a hash h0 is created by hashing the announcement along with the block header hash of the most recent block, when performing this hash, the SoftNonce, is zeroed and HashBranch and LastItemPrefix are omitted. Then h0 is expanded into array of 215 1KiB items using a memory hard hash called RandMemoHash. Third, a Merkle tree is built from that array of items and announcement is hashed again with the Merkle root (again with SoftNonce zeroed and MerkleBranch and LastItemPrefix omitted). Fourth, the announcement is mined using a variant of PacketCrypt with RandomHash cycle in each iteration. When a hash is found which meets the requirement in WorkBits, a PacketCryptProof is created only for the fourth item. The Merkle branch for the forth item is placed in MerkleBranch and the first 40 bytes of this item are used to pad out the end of the announcement, filling LastItemPrefix.

Verification is similar but with less memory needed, the announcement is hashed as before but instead of creating an array of 215 items up-front, the items are created as needed during the PacketCrypt cycle. When the 4th item is reached, its hash and index is validated against the HashBranch entry and the result of the PacketCrypt cycle is compared to WorkBits.

E. GPU and ASIC frustration

While the PacketCrypt block mining algorithm is largely intended to work on any hardware which can encrypt and move data quickly, the announcement hashcash is designed to prefer the unused CPU cycles of existing hardware which is geographically distributed. While we acknowledge that CPU-preferring algorithms carry a higher risk of ASIC implementation, we prefer CPU for the announcement hashcash for two reasons. First, we want to prevent GPU/ASIC mined spam and novelty announcements from flooding the network by out bidding legitimate announcements. Secondly, bandwidth is only interesting if it actually goes somewhere, it would be largely self-defeating if announcement miners and block miners all placed their equipment in datacenters, or worse, colocated it in the same datacenter to eliminate transit costs.

For the announcement mining, the addition of the RandomHash is intended to maximally frustrate GPUs and ASIC implementations. RandomHash constructs a random “program” made of a set of instructions that do a random sequence of math operations. This strongly favors the flexibility of general purpose processors over GPUs or ASICs.

F. Announcement rules

In order to be valid, an Announcement will need to contain two commitments:

1. The header of the most recent block at the time the announcement was created, when hashing the announcement, the hash of block at its ParentBlockHeight will be included.
2. The amount of proof of work which the announcer intends to perform (target difficulty), this is specified in WorkBits.

An Announcement will only be valid for inclusion in a block at height ParentBlockHeight + 3. The choice of 3 is to allow one block period for network participants to mine an Announcement, one block period for them to broadcast it across the network, and the third block period when miners stop accepting new announcements because they’re mining. This rule is relaxed during the first 3 blocks of the chain for obvious reasons.

G. Compact proofs

In order to reduce blockchain forks, it is desirable to avoid sending the AnnouncementSet over the wire with a newly found block. Because the PacketCryptProof provides a random sample of the Announcement set, we can verify any property which we wish all Announcements to have with the same confidence as our belief the miner is not using probabilistic TMTO. For example, if we insist that all Announcements must begin with the letter "A", it does a miner no more good to mine an Announcement set where one starts with a "B" than he does to omit that Announcement entirely.

By requiring Announcements contain a minimum amount of CPU work, miner must commit to the minimum work of any Announcement in his AnnouncementSet to the coinbase before the the verifier checks that the 4 Announcements which he provided adhere to this commitment.

H. AnnouncementSet rules

For a miner's block to be valid, all of the Announcements which they report in their PacketCryptProof must be valid, furthermore the Merkle tree of the announcements in the PacketCryptProof must match a merkle tree which is committed in the coinbase. Also in the coinbase, there must be a commitment to the minimum announcement work (henceforth min_ann_work) and all Announcements in the PacketCryptProof must have at least this as their minimum required difficulty. Finally of course, the solved difficulty must be valid according to the target as defined by the blockchain consensus rules. This global difficulty is defined as follows:

```
Work(hash) = (2**255 / hash + 1)
ann_set_work =
Work(PacketCryptVerify(coinbase,
packet_crypt_proof))
min_ann_work =
MIN(ann.target_work, for_each
ann in ann_set)
global_work = ann_set_work *
min_ann_work * ann_count
block_is_valid = global_work >=
    Work(CompactToBig(block_head
er.nBits)) ** 3
```

III. PARTICIPANT BEHAVIOR MODELLING

We will now attempt to reason about the behavior of participants in the network. We will start with the incentives facing network participants and then finish by discussing the miner's incentives and the incentives which affect both roles.

A. Network participant incentives

Looking at existing cryptocurrencies and the emergent ecosystems around them, we can enumerate a number of different activities which network participants will want to engage in.

- Transfer of cryptocurrency
- Making and responding to OTC market offers
- Communicating the quality of their network links in order to be able to sell bandwidth leases.
- Participating in mining (e.g. in a mining pool)
- Broadcast novelty messages

The transfer of cryptocurrency is perhaps the most simple case because it is fulfilled by the blockchain and lightning network operating in their normal capacity. All other objectives are achieved through the use of announcements.

B. Types of announcements

There is no technical limitation against an announcement containing any type of data, but we can still attempt to predict the general types of announcements which will be in common use. We predict that announcements will take 4 general forms:

- Network state updates
- Market offers
- Novelty announcements
- Mined (empty) announcements

1) Network state updates

These are announcements which communicate changes in the quality and amount of available network bandwidth on a particular link. Participants in the network send them to maintain a good reputation by maintaining clear communication with those who have leased access to their bandwidth as well as those who might lease access in the future.

2) Market offers

While the Lightning Network provides a convenient way for network participants to make OTC exchanges, there remains a need for discoverability of available offers. The announcement system provides a fully decentralized way for market offers to be exchanged without any kind of "exchange" to coordinate them. The most obvious type of offer which will be exchanged is the offer of a network bandwidth lease but we foresee the emergence of many different types of assets being exchanged in emergent OTC markets.

3) Novelty announcements

From the history of arbitrary data in the Bitcoin blockchain to applications like Cryptokitties, it should be clear by now that anything which can be used to broadcast arbitrary messages to the world, will be. Participants who send novelty data act for their own amusement and thus do not act in ways considered economically rational. At times they are prepared to spend many times what a rational actor would spend for the same service for its intended use.

Because announcements are not stored in the blockchain, we consider novelty announcements to be of little negative consequence and potentially positive, as they create incentive for miners to use more bandwidth than they would have otherwise. We can imagine such applications as chat or microblogging to be built on top of novelty announcements.

4) Mined announcements

Mined announcements are announcements which are created for the sole purpose of assisting miners in winning a block. Unlike any other type of announcement discussed, the announcer does not want the announcement to be received by as many network participants as possible, he rather wants it to be received by a miner who will pay him for it.

A mined announcement will normally tend to be as close to empty as possible in order to save bandwidth and memory. Because announcements necessarily contain a 1024 byte proof, they cannot be compressed beyond this point without requiring the recipient to reconstruct the entire dataset in order to reconstruct the announcement, meaning that depending on the minimum work which the miner wishes his announcement miners to produce, announcement mining may become a bandwidth-limited problem.

C. *Announcement cost and demand*

The amount of cost which a network participant is willing to incur in order to send an announcement is defined by what is effectively a market for bandwidth on congested links. As long as there is no network congestion, the cost to an announcer converges on the minimum that is valid for an announcement to be forwarded. When some links in the network begin to become congested and nodes begin to prioritize which announcements they forward, the cost to the announcer grows to the market rate for reaching the subset of nodes who are behind congested links. Because participants have different priorities, we expect announcements to bear a wide range of different amounts of work.

While we cannot hope to accurately predict demand, we can use information that we know about Bitcoin transaction fees to estimate the distribution of work which will be done on announcements. On November 27, 2018, we collected a snapshot of fee distribution for bitcoin transactions and used that to plot a chart.

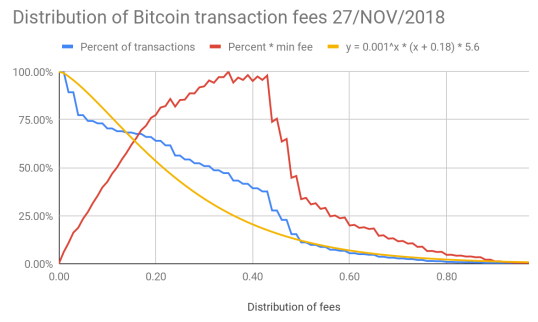


Fig. 2. Two data-points were removed from this chart: One is transactions which paid nothing at all, these are non-standard transactions and are not forwarded, the other is transactions which paid more than 99 satoshis per byte which our data source lumped together in one data point.

On the x axis is the range of fees which transactions are using and the blue line is the percentage of transactions which were paying at least x transaction fee. As you can see, about 50% of all transactions are in the bottom 30% of fee distribution. The red line shows the percent of transactions times the minimum fee from that subset, aka min_ann_work * ann_count. So if distribution of work on PacketCrypt announcements follows the same that of Bitcoin transaction fees, it is in a miner's best interest to mine the highest paying 47% of announcements (y axis of the blue line at x where the red line is at its maximum). The yellow curve is the equation y = 0.001x * (x+0.18) * 5.6 which was derived from an exponential regression with some additional tweaking to improve the fit. Importantly, y trends toward zero as x goes to infinity but the opposite is not true, this reflects the reality of a fixed minimum effort required to create an announcement.

IV. Miner Incentives

With ann_set_work, min_ann_work and ann_count all multiplied together, we expect miners expend effort on whichever number can be raised by a given percentage most cheaply.

The cost associated with min_ann_work rises linearly with a rise in ann_count because each announcement in the set of size ann_count must bear at least min_ann_work proof of work.

The cost of increasing ann_set_work rises with ann_count as well, but as a function of memory bandwidth and capacity. For example, if the size of the announcement set is less than half the size of the memory used to store it, ann_count can be doubled with a relatively low impact on the cost of mining the announcement set. However, if the announcement set is near to the limit of a cache or memory size, the cost of expanding ann_count even slightly could be more than an order of magnitude.

A miner who is collecting announcements from the network will be incentivised to choose from all announcements which he knows of, subset for which min_ann_work * ann_count is the greatest, henceforth known as the best subset, that is the peak of the red line in the chart.

A. Mining announcements

When increasing min_ann_work * ann_count is cheaper than increasing ann_set_work, a miner will tend to begin mining the announcements or paying someone else to do it. This will tend to happen when the value of the block reward is more than double the work value of the best subset.

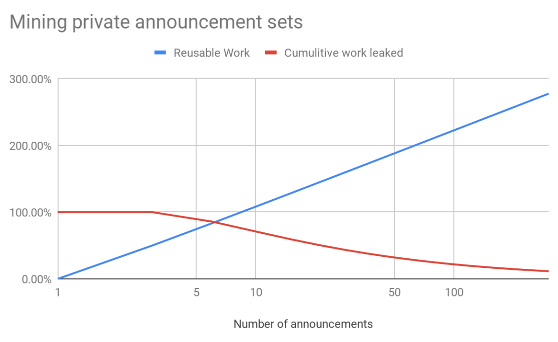
B. Expanding the best subset

When the block reward exceeds double the work value of the best subset, miners will be incentivised to expand it by mining additional announcements which have the least work that they can while still falling in the best subset.

C. Changing the best subset

When the savings in memory and bandwidth cost difference exceeds the value of the lowest value announcements in the best subset, miners are incentivized to mine announcements with more than the least possible work, in order to increase the min_ann_work of the best subset.

In the most degenerate case, the memory/bandwidth cost difference exceeds the value of all public announcements and a miner would be inclined to mine their own announcement set of one. However, in the next block he can add 50% to his min_ann_work * ann_count by increasing ann_count to 3, reusing the announcement and mining 2 new announcements at half the value. This value grows at about ½ the natural log of number of announcements which must be processed in memory per block. Meanwhile the announcement must be provided in the PacketCryptProof and is thus minable by competitors in the next block (at half the value). This "leaked work" is roughly MIN(ann_count, 4) * ann_min_work so at the small ann_count numbers where a low-memory ASIC might be competitive, it is significant.



At some point, when the benefit of increasing reusable work and reducing cumulative leaked work is outweighed by the memory/bandwidth cost of increasing ann_count, the miner will be forced to stop reusing any old work and begin making all of his announcements himself, perhaps selling the old ones to another miner with more available memory.

V. Conclusion

Here we documented the PacketCrypt proof of work algorithm which is designed to be bound by both memory latency and network bandwidth between participants. We reasoned about a number of potential failure modes using public data from the Bitcoin network and we considered different use cases of Announcements and how users might affect the network in different ways.

The code for this algorithm can be found at: https://github.com/cjdelisle/PacketCrypt/ and is currently implemented in the PKT cryptocurrency (https://pkt.cash).