PacketCrypt

Caleb James DeLisle (cjd@cjdns.fr)
Vishnu Seesahai (vjs1@cornell.edu, vishnu@gridfinity.com)

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.

1. PacketCrypt Protocol Overview

The PacketCrypt protocol consists of two distinct stages, in the first stage ("announcement mining"), the miner executes a CPU-hard algorithm which creates 1KB proof (called an *announcement*), then in the second stage ("block mining") the miner pre-commits a merkle root of collected announcements and then executes a memory-hard proof of work algorithm which accesses random announcements from that set. Upon finding a "winning hash", the miner presents the announcements which were accessed in that cycle as well as merkle branches linking them to the pre-commitment, and statistically proving that they had as many announcements as they claimed to have. This presentation is referred to as the PacketCryptProof and is needed to prove that the work was done.

The amount of work which the block miner needs to perform is reduced based on the amount of announcement work in the pre-committed set of announcements. But effective announcement work decays as time goes on so block miners require a steady supply of fresh announcements, thus creating the demand for bandwidth.

2. PacketCrypt Algorithm

PacketCrypt is an algorithm which takes input data (for example of a block header) and a list of data items (in the block mining stage, the data items are the announcements) 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 of 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 against the block mining stage of the PacketCrypt protocol is to throw away announcements or lie about the number of announcements which you have. Suppose you wanted to pretend you had twice as many announcements as you really had: Every time you perform a hash operation which seeks an announcement 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. Scrypt, one of the earliest works on memory hard functions, has a *regenerative TMTO* which requires only 17% increase in processor time to halve the memory. This issue has since been formalized[1] 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 in 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

[1] https://eprint.iacr.org/2015/430.pdf

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:

Version	uint8	Version number, currently 2
SoftNonce	[3]byte	Nonce used for mining, can be changed without regenerating the mining dataset.
HardNonce	[4]byte	Nonce used 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
UserDefined	[40]byte	Data which can be decided by the announcement miner
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.
LastItemPrefix	[40]byte	First 40 bytes of the last item in the dataset.

Table 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.

a. 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 an array of 2^{13} 1KiB items (8MB) by generating a RandomHash program (explained further below) and executing it 2^{13} times. Third, a Merkle tree is built from that array of items using 512 bit blake2b and the announcement is hashed again with the Merkle root (again with SoftNonce zeroed and MerkleBranch and LastItemPrefix omitted). Fourth, the announcement mining begins, with SoftNonce advanced to try each new hash. When a hash is found which meets the requirement in WorkBits, the Merkle branch of the forth data 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 2^{13} 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.

i) 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 because bandwidth is only interesting if it actually goes somewhere and 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 sequence of math, logic and branching operations. This favors the flexibility of general purpose processors over GPUs or ASICs.

ii) Announcement batch limits
If an announcement miner is allowed to create too many announcements with the same dataset, the announcement miner is able to avoid expanding bandwidth sending them all to the block miner because instead they can simply send the entire dataset plus a compressed representation of the winning SoftNonces.

To prevent this, we limit the number of winning announcements to around 512 per batch. The reason for the choice of 512 is because the merkle tree of all 2^{13} items in the dataset is 1MB which is twice that of 512 announcements, providing a reasonably comfortable margin given the recipient needs merkle branches and also a 40 byte prefix of one of the actual data items to reconstruct the announcements.

The way the batch limit is applied is by constraining the range of SoftNonce, but since the difficulty of winning an announcement is variable, the range of SoftNonce must also be variable. The range of SoftNonce is 0-1024 if the hash target is at the minimum (every second hash is a winner), but for every halving of the hash target, the maximum SoftNonce doubles, keeping the expected number of valid announcements stable at 512.

b. Announcement rules
In order to be valid, an announcement needs to contain two commitments:

1. The hash of the most recent block at the time the announcement was created
2. The amount of proof of work which the announcer intends to perform (target hash), 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. After block number ParentBlockHeight + 3, the value of the work done on an announcement halves each block following, this is done to ensure that block miners must continuously mine or download fresh announcements. This decayed valuation of announcements is referred to as *effective work*. After the amount of *effective work* decays below the minimum allowable work, the announcement is no longer usable in any capacity.

c. Compact proof
Since the AnnouncementSet can be many gigabytes in size, it is unacceptable to include the entire set with each block. Because the PacketCryptProof provides a random sample of the AnnouncementSet, we can verify any property which we expect 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

announcement starts with a “B” than he does to omit that announcement entirely.

We can therefore require announcements to contain a minimum amount of CPU work by requiring the block miner to specify the minimum work of any announcement in his AnnouncementSet, in his coinbase commitment. The verifier checking that the 4 announcements provided with the PacketCryptProof will also verify that he adhered to this commitment.

d. AnnouncementSet rules
For a miner’s block to be valid, all of the announcements which they provide in their PacketCryptProof must also be valid, furthermore the Merkle tree of the announcements in the PacketCryptProof must match a merkle root which is committed in the coinbase. Also in the coinbase, there must be a commitment of the minimum announcement *effective work* (henceforth `min_ann_work`) and all announcements in the PacketCryptProof must have at least this as their minimum required hash work. Finally of course, the work done by the block miner must be valid according to the target as defined by the blockchain consensus rules. However, the amount of work expected of the block miner is based on the amount of announcements that they have, so the computation is not straight forward. This global difficulty is defined as follows:

```
Work(hash) = (2**255 / hash + 1)
block_miner_work = Work(packet_crypt_hash)

min_ann_work = MIN(
        for_each ann in ann_set:
                Decay(ann.target_work)
)
global_work =
        block_miner_work *
        min_ann_work *
        (ann_count ** 2)

work_is_valid =
        global_work >=
                Work(block_header.nBits) ** 3
```

In words:

- `block_miner_work` is the amount of memory-hard work done by the block miner
- `min_ann_work` is the lowest amount of *effective work* that is done for any of the announcements in the AnnouncementSet
- `global_work` is the product of `block_miner_work` times `ann_min_work` times the `ann_count` squared

And the work is valid if the global work is greater than or equal to the cube of the difficulty as specified in the block header. The announcement count is squared because first it is multiplied by the `ann_min_work` to compute the effective value of announcement mining effort and then it is multiplied again to add a secondary equal weight to bandwidth usage.

3. 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. Announcement miner 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 can be achieved through the use of announcements.

a. Types of announcements
There is no technical limitation against an announcement containing any type of data, but we can will 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

i) 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 may 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.

ii) 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.

iii) 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 unlikely to have any negative consequence and potentially to be 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.

iv) 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 have no user defined data order to save bandwidth by compression.

b. 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.

B. Block miner incentives
With `block_miner_work`, `min_ann_work` and the square of `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 `block_miner_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 ** 2` is the greatest, henceforth known as the *best subset*.

a. Mining announcements

When increasing `min_ann_work * ann_count ** 2` is cheaper than increasing `block_miner_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*.

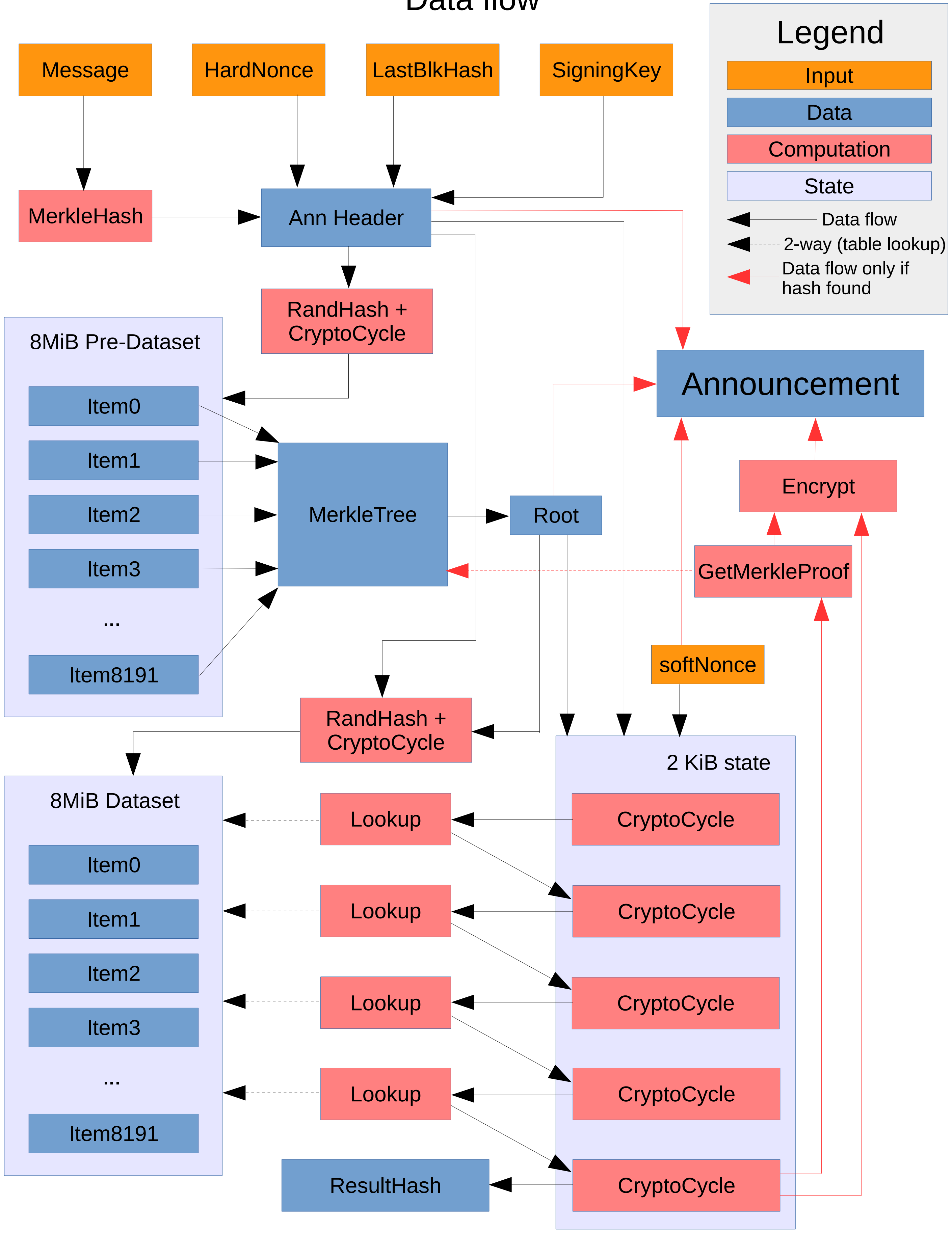
i. Changing the best subset

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.

4. 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 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).

PacketCrypt2 Announcement Hash Data flow



Legend
Input
Data
Computation
State
Data flow
2-way (table lookup)
Data flow only if hash found
Message
HardNonce
LastBlkHash
SigningKey
MerkleHash
Ann Header
RandHash + CryptoCycle
8MiB Pre-Dataset
Item0
Item1
Item2
Item3
...
Item8191
MerkleTree
Root
Announcement
Encrypt
GetMerkleProof
softNonce
RandHash + CryptoCycle
8MiB Dataset
Item0
Item1
Item2
Item3
...
Item8191
2 KiB state
Lookup
CryptoCycle
Lookup
CryptoCycle
Lookup
CryptoCycle
Lookup
CryptoCycle
ResultHash
CryptoCycle