DigitalNote

XDN-project

http://digitalnote.org

June 6, 2015

Abstract

In this whitepaper we present DigitalNote XDN a brand new technology for securely transferring money and messages between anonymous peers. The new version of the XDN protocol also provides deposits with interest rate. Our solution is based on the CryptoNote code base and keys/addresses system including multi-signatures outputs and ring signatures. [1] [2]

1 Intro*duck*tion. Launch story.

DigitalNote XDN was launched and announced as the duckNote cryptocurrency on May 30, 2014. It's aim was to spread the hashrate and information about the anonymous CryptoNote based XDN among ordinary users and the cryptocurrency community. Libertarian economy with an original supply curve, proved fair decentralization with CPU-efficient ASIC-resistant mining process, brilliantly scaled network specifications, user friendly cross platform GUI wallet and lots of network improvements was brought by XDN in its duckNote appearance. [3]

In mid September 2014 XDN was big enough to go to the next step. duckNote was renamed to DarkNote, as assumed by many of XDN community members. With the new name XDN received new DarkNote only unique features, like untraceable encrypted messages, transfers aggregation, other source code and network improvements.

2 DigitalNote features:

- Unlinkable transactions (Appendix A)
- Untraceable payments (Appendix B)
- Blockchain analysis resistance (Appendix C)
- CPU-efficient ASIC resistant Proof-of-work (Appendix D)
- Adaptive and scalable (Appendix E)
- Untraceable encrypted messages (3)
- Multi-signatures support (4)
- Deposits based on blockchain technology (5)

In the next sections we will present DigitalNote unique features.

3 Encrypted messages

3.1 Motivation

Imagine, an encrypted untraceable messaging platform where even the fact of message sending is unknown, only recipient can decrypt his message and message can be stored in blockchain forever. XDN made it.

Cryptocurrency is about digital money and how the money flows between parties. However bare transfers are sometimes just inconvenient, the payee needs to get some additional information from the payer, like secret text messages. For example, an on-line store supports a "money-back feature" and its customer should be able to attach his refund address to his transaction. Alternatively, when you make a donation you may want to specify how your funds should be distributed between all possible charities (just like Humble Bundle allows you to divide your purchase between games' developers).

The urgency of this feature is supported by the fact that some cryptocurrencies have already implemented it: Bitcoin [4], Florin [5], Cosmoscoin [6] etc. But their developers did not attend to convenience of private communication. Whilst there is no problem with attaching a plain-text message, it is tricky to provide an easy-to-use way of handling of secret data.

Both parties can share some *symmetric private key* and use it for encrypting and decrypting messages. But this method is only suitable for a long-time communication and/or repeated transactions. Another way: they can use a *public key encryption scheme*, but such algorithms are less effective in case of arbitrary-length messages (when compared with symmetric crypto).

We propose a protocol for transferring encrypted messages within transactions, which does not require any preliminary data exchange. Also it is based on the modern symmetric stream cipher — ChaCha20 [7] — and results in excellent performance.

3.2 The protocol

We are now to describe step-by-step the protocol for encrypting and decrypting a message. The message is being sent from Alice to Bob. Firstly we will provide the high level scheme:

1. Alice generates common secret via Diffie-Hellman key exchange protocol in the same CryptoNote one-time keys are generated.
2. She encrypts her message under this key and sends the transaction.
3. Bob re-generates the common secret, just like he recovers CryptoNote outputs private keys.
4. He tries to decrypt all available messages in the transaction and determines (via a checksum) those which were sent to him.

Encryption

Let (A, B) be Bob's CryptoNote address and $\mathcal{H}()$ to be cryptographic hash function Keccak.

Alice generates a random value r and computes the common secret $x = H(r \cdot B)$. Additionally she stores $R = r \cdot G$.

Then Alice takes the plain-text message M and adds four zero bytes at the end. The motivation of this step will be shown later. After that she uses x as a key for stream cipher ChaCha20 and gets a pseudo-random bit sequence S.

The resulting encrypted message is $E = M \oplus S$. It is stored in the transaction along with R.

Decryption

Bob receives the transaction and re-generates the common secret as $x = H(b \cdot R)$. With x he recovers the same sequence S.

Then for every encrypted message E_i he computes $M_i = E_i \oplus S$. M_i which have the last four bytes zeroed indicated they were sent to him, i.e. decrypted correctly. The others may belong to other recipients of the transaction or even be Alice's comments for herself.

3.3 Separate messages

Our solution does not rely on an output properties: payee, amount, any other content. That means that transaction comments may be used separately with money transfer, i.e. like just private messages (without payments). Alice can send many messages to different addresses in a single transaction which sends some funds to herself. Due to CryptoNote's unlinkabile one-time keys no one can prove that there were no money transfer at all: i.e. private message via transaction is indistinguishable from ordinary transactions.

DigitalNote XDN can serve as a service of private encrypted communication, as well as secure money transfers.

3.4 Other applications

- **Greetings** A digital postcard with personal "X-mas greetings" and some present inside.
- **Moneyback** A customer can specify his public address (with occasional purchases as he may not have a permanent account on a site), so that only the merchant can see it.
- **Payment ID** A merchant may not provide newly created address for each customer's transaction, but instead demand to transfer the payment ID inside.
- **Donations** A donator can specify (in public or private manner) how to distribute his pledge.

4 DigitalNote Multi-signatures

A multi-signature address is an address that is associated with more than one ECDSA private key. The simplest type is an m-of-n address - it is associated with n private keys, and sending bitcoins from this address requires signatures from at least m keys. A multi-signature transaction is one that sends funds from a multi-signature address. There are several Multi-signatures use cases. One is to greatly increase the difficulty of stealing the XDN. With a 2-of-2 address, you can keep the two keys on separate machines, and then theft will require compromising both, which is very difficult - especially if the machines are as different as possible (e.g., one pc and one dedicated device, or two hosted machines with a different host and OS). It can also be used for more advanced scenarios such as an address shared by multiple people, where a majority vote is required to use the funds. It can also be used for redundancy to protect against loss - with a 2-of-3 address, not only does theft require obtaining 2 different keys, but you can still use the coins if you forget any single key. This allows for more flexible options than just backups.[8]

5 Deposits with interest rate based on XDN blockchain

You can safely deposit your DigitalNote with some interest rate. In June 2015 XDN introduced new blockchain based feature: time-locked deposits with variable annual interest rate. In general it allows users to "lock" some of their XDN for a while (from one month to a ten years) and after some time, withdraw it back with some interest as a part of main DigitalNote supply. Longer period gives users a bigger interest rate.

Deposits are implemented via new types of transaction output/outputs. It includes amount, destination key (or keys) and time (expressed in blocks) to lock. Transaction itself contains the field `unlock_time` but output-specific parameters is much convenient, because user may want to sent some money back as change (and surely doesn't want them to be locked). The transaction is included in the blockchain as usual and the counter starts.

When the lock expires user can spend this output as usual, but the new transaction volume will be increased with the interest. For example, if user makes a deposit of 1000 XDN for about 1 year an annual interest rate will be about 1%. It also means that deposits act as new source of emission.

DigitalNote **multi-signature** core feature enables a common ownership for any XDN units and deposits in particular. A family can store their XDN savings in N-of-N deposits, which means that only all members together can withdraw the money. Company can keep its capital in M-of-N address, which is redeemable only by at least M members out of N.

Interest rate depends deposit time and varies from one month to ten years. The latter case is the most profitable (gives user maximum possible profit — 11% of his deposit). The relation takes form of hyperbolic function, as shown below.

6 Proof-of-Activity

Security is the highest priority of any money-related system. DigitalNote will be the first cryptocurrency ever implemented the truly hybrid PoW+PoS scheme, suggested in [10].

Proof-of-Activity (PoA) is the two-phase protocol:

1. PoW miner generates *empty* block header and broadcasts it to the network. N pseudo-random numbers are derived from the result hash. These numbers correspond to the N duckoshi[1] selected from all coins currently deposited. Since this information is public, it is possible to map every number to a specific output and its public key.

2. These N public keys belong to selected stakeholders, who should sign the block. First $N-1$ stakeholders just provide their signatures, whilst the N^{th} user fills the block with transactions (including the generating transaction), forms the block and signs all data. The reward and transaction fees are divided between PoW and PoS participants.

If someone from N stakeholders is offline, eventually the next PoW empty block appears and the process continues. PoW difficulty is adjusted to the current fraction of on-line holders, so the blockchain grows evenly.

The main benefit of PoA is the greater protection from 51% attack, because the adversary needs to have both PoW and PoS powers to make his own alternative blocks to double spend. Original CryptoNote code base would not allow to apply the native PoA algorithm because of *too anonymous* transactions. Public deposits in DigitalNote made it possible to use the peer-reviewed secure PoA scheme without any changes.

[1] Duckoshi is the smallest XDN unit

7 Other DigitalNote specific features

7.1 Block time, transaction processing and orphans

DigitalNote has a 4-minute block interval. In general it is 2.5x times faster than Bitcoin. It provides almost a negligible quantity of orphan blocks compared with other CryptoNote currencies. Less orphans lead to less losses and much profits for miners. Greater incentive for miners ensures bigger hashrate and most secure network.

7.2 Emission process

The total number of XDN is 8589869056, which is a 6th perfect number [9]. Whilst it may look weird in a decimal numeration, it's binary form looks great: $11111111111111110000000000000000_2$.

7.3 DigitalNote emission

DigitalNote XDN has 2 sources of main emission:

1. Main source of XDN supply is a CPU-efficient ASIC-resistant Proof-of-work mining with a constant base block reward = 150 XDN

2. Additional source of emission is a XDN deposit with about 1% annual interest rate that may act as a Proof-of-activity element in future.

7.4 duckNote and DarkNote emission

Initial base block reward is 320000 XDN. Every 11000 blocks (approx. 1 month) it halves until reaching 150 XDN. It will occur in roughly one year after first block. That means that 80% of ever made coins will be available for free market use after just one year of fair Proof-of-work mining.

After 132000 block, base block reward remains 150 coins until the max number of coin will be reached. It implies that the emission process will last about 77 years, supporting mining incentive and increasing money supply.

7.5 Block rewards

Every block reward is a round number: 320k, 160k, ..., 150 notes. Round numbers are more convenient for human beings: they are easy to remember and easy to deal with. One can accurately predict his mathematical expectation of his mining revenue for arbitrary period of time and therefore his profits.

As a result, there is no dust (millicents of coins) in mining transaction, and this fact has two benefits:

- **Reducing the size** of blocks and transactions (and the whole blockchain).

- **Increasing privacy**, because dust is hard to use in ring signature (one needs to find outputs with the same amount), so you must spend it not anonymously.

7.6 "Spacious" blocks

All CryptoNote-based currencies has smart and neat mechanism of limiting the size of blocks, deterring transaction flood attack. If current block is greater that median value of last

N blocks, it's reward is decreasing with exponential speed. The default CryptoNote "free block size" (maximum size of block which is not affected by the rule above) is 10 KB.

We increased this value up to 32 KB, which allows more transaction to fit in a block "for free". Thrice more transactions compensate twice less frequent blocks and create opportunity for healthy economy growth.

8 Conclusion

In this whitepaper we presented new cryptocurrency DigitalNote XDN. We took all the good things from the CryptoNote (untraceability, unlinkability, egalitarianism and multi-signatures) and added several unique blockchain features.

One is untraceable encrypted messages and encrypted transaction comments. Data is encrypted by the state-of-art stream cipher and all the keys are automatically derived from a transaction content (no need of preliminary key exchange).

Another feature is time-locked deposits with interest rate: user can keep his DigitalNote safe, while DigitalNote amount is increasing. Also deposits allow to apply secure Proof-of-Activity scheme for greater network protection.

Additional benefits of DigitalNote are different network specifications (like block time) and focusing on free market economy with an original supply curve.

We believe DigitalNote will become one of the bases for the future crypto economy.

References

[1] https://cryptonote.org

[2] http://en.wikipedia.org/wiki/Ring_signature

[3] http://en.wikipedia.org/wiki/Economic_liberalism

[4] https://github.com/bitcoin/bitcoin/pull/2738

[5] https://bitcointalk.org/index.php?topic=236742.0

[6] https://bitcointalk.org/index.php?topic=245938.0

[7] http://cr.yp.to/chacha.html

[8] http://bitcoin.stackexchange.com/questions/3718/what-are-multi-signature-transactions

[9] https://en.wikipedia.org/wiki/Perfect_number

[10] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld, "Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake", 2014, https://eprint.iacr.org/2014/452

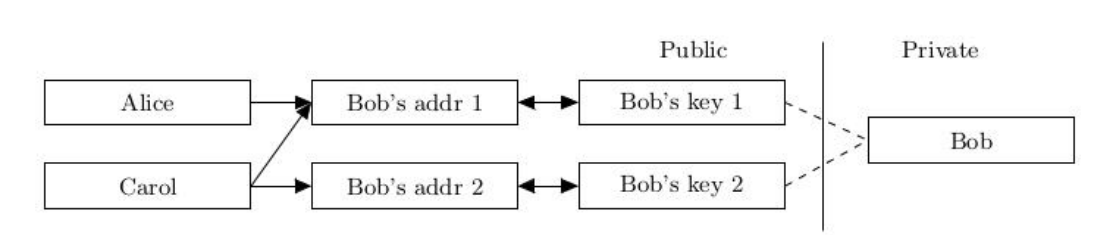
[11] https://cryptonote.org/cns/cns008.txt

A Unlinkable transactions

In this section we cite CryptoNote whitepaper[1]

One-time keys

Classic Bitcoin address, once being published, becomes unambiguous identifier for every incoming payment, linking them together and tying to the recepient's pseudonym. If someone wants to receive "untied" transaction — he should convey his address to the sender by a private channel. If he wants to receive different transactions which can not be proven to belong to the same owner — he should generate all the different addresses and never publish them in his own pseudonym.



Pic. 1. Traditional Bitcoin keys/transactions model.

We propose a solution allowing user to publish **a single address** and receive unconditional unlinkable payments. Destination of each CryptoNote output (by default) is an public key, derived from recipient's address and sender's random data. The main advantage against Bitcoin is that every destination key is unique by default (unless the sender use the same data for each his transaction to the same recipient), even for a change which the sender returns to himself. So that there is no such issue as "address reuse" by design and no observer can determine if any transaction was sent to a specific address or link two addresses together.

Pic. 3. CryptoNote keys/transactions model.

Firstly the sender performs Deffie-Hellman exchange protocol to get a shared secret from his data and a half of the address. Then he computes one-time destination key, using these secret and the second half. Two different recipient's ec-keys are needed for these two steps, so standard CryptoNote address is nearly twice as large as Bitcoin one. The receiver also performs Diffi-Hellman protocol and then recovers corresponding secret key.

Standard transaction goes as follows:

1. Alice wants to sent a payment to Bob, who has published his standard address. She unpacks the address and gets Bob's public user key (A, B).
2. Then she generates a random $r \in [1, l-1]$ and computes public one-time key $P = \mathcal{H}_s(rA)G + B$.
3. Alice uses P as a destination key for the output and also puts value $R = rG$ (as a part of Diffie-Hellman protocol) somewhere into the transaction. Note that she can create other outputs with unique public keys: different recipients' keys (A_i, B_i) imply different P_i even with the same r.

Pic. 4. Standart transaction construction.

4. Now Alice can send the transaction.

5. Bob checks every passing transaction with his private user key (a, b), computing $P' = \mathcal{H}_s(aR)G + B$. If there was Alice's transaction, then $aR = arG = rA$ and $P' = P$.

6. Now Bob can recover corresponding one-time private key: $x = \mathcal{H}_s(aR)+b$, so as $P = xG$. He can spend this output at any time by signing transaction with x.

Pic. 5. Incoming transaction check.

As a result Bob gets incoming payments, associated with one-time public keys which are **unlinkable** for a wingside spectator. Some additional notes:

- When Bob "recognizes" his transactions (step 5) he practically uses only half of his private information: (a, B). This pair — **tracking key** — he can pass to a third party (Carol) and delegate to her processing new transactions. Bob doesn't need to trust Carol his money, because she can't recover secret one-time key p without full private user key (a, b). This approach is useful when Bob lacks bandwidth or computation power (smartphones, hardware wallets etc.).

- In case Alice wants to prove she sent a transaction to Bob's address she can either disclose r or use any kind of zero-knowledge protocol to prove she knows r (for example, just sign the transaction with r).

- If Bob wants to have an auditable address with all incoming transaction to be linkable, he can either publish his tracking key or use **truncated address**. That address represent only one public ec-key B, and the rest part (which the standart protocol needs) is derived from it: $a = \mathcal{H}_s(B)$ and $A = \mathcal{H}_s(B)G$. In both cases every person is able to "recognize" all Bob's incoming transaction, but, of course, can not spend it without secret key b. This scenario is applicable for accounting public donations.

B Untraceable payments

In this section we cite CryptoNote whitepaper[1]

One-time ring signature

The protocol above allows users to achieve unconditional unlinkability. But ordinary types of signatures still preserve traceability of all transfers: each transaction input points to a certain output. Our solution lies in using more complicated signature type.

At first we will provide a description of our algorithm without reference to e-cash and then show how to incorporate it in the standard transaction protocol.

One-time ring signature consists of four algorithms (**GEN**, **SIG**, **VER**, **LNK**):

GEN takes public parameters and outputs ec-pair (P, x) and public key I.

SIG takes message m, set $\mathcal{S}'$ of public keys $\{P_i\}_{i \neq s}$, pair (P_s, x_s) and outputs signature σ and set $\mathcal{S} = S' \cup \{P_s\}$.

VER takes message m, set $\mathcal{S}$, signature σ and outputs "true" or "false".

LNK takes set $\mathcal{I} = \{I_i\}$, signature σ and outputs "linked" or "indep".

General purpose of the protocol is simple: a user produces a signature which can be checked not by a single public key, but a set of them. The real signer is indistinguishable from the other key owners until he produces the second signature under the same keypair.

Pic. 6. Ring signature anonymity.

GEN: Signer picks up a random secret key $x \in [1, l-1]$ and computes the corresponding public key $P = xG$. Additionally he computes another public key $I = x\mathcal{H}_p(P)$ which we will call "key image".

SIG: Signer generates one-time ring signature with non-interactive zero-knowledge proof using the technics from [**?**]. He selects a random subset $\mathcal{S}'$ of n other users' public keys P_i, his own keypair (x, P) and key image I. Let $0 \le s \le n$ be Signer's secret index in $\mathcal{S}$ (so that his key is P_s).

He picks up random $\{q_i \mid i = 0 \dots n\}$ and $\{w_i \mid i = 0 \dots n, i \neq s\}$ from $(1 \dots l)$ and makes *commitments*:

$$L_i = \begin{cases} q_i G, & \text{if } i = s \\ q_i G + w_i P_i, & \text{if } i \neq s \end{cases}$$

$$R_i = \begin{cases} q_i \mathcal{H}_p(P_i), & \text{if } i = s \\ q_i \mathcal{H}_p(P_i) + w_i I, & \text{if } i \neq s \end{cases}$$

Next step is getting non-interactive *challenge*:

$$c = \mathcal{H}_s(m, L_1, \dots, L_n, R_1, \dots, R_n)$$

And finally Signer computes *response*:

$$c_i = \begin{cases} w_i, & \text{if } i \neq s \\ c - \sum\limits_{i=0}^{n} c_i \mod l, & \text{if } i = s \end{cases}$$

$$r_i = \begin{cases} q_i, & \text{if } i \neq s \\ q_s - c_s x \mod l, & \text{if } i = s \end{cases}$$

The resulting signature is $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$.

VER: Verifier checks the signature, reconstructing the commitments:

$$\begin{cases} L'_i = r_i G + c_i P_i \\ R'_i = r_i \mathcal{H}_p(P_i) + c_i I \end{cases}$$

Then Verifier checks if $\sum\limits_{i=0}^{n} c_i \stackrel{?}{=} \mathcal{H}_s(m, L'_0, \dots, L'_n, R'_0, \dots, R'_n) \mod l$

If this equality is correct, Verifier runs algorithm **LNK**, otherwise rejects the signature.

LNK: Verifier checks if I has been used in past signatures (these values are stored in set $\mathcal{I}$). Double use means that two signatures were produced under the same secret key.

The meaning of the protocol: using L-commitments Signer proves that he knows such x that at least one $P_i = xG$. To make this proof non-repeatable we introduce key image as $I = x\mathcal{H}_p(P)$. Signer uses the same coefficients (r_i, c_i) to prove almost the same: that he knows such x that at least one $\mathcal{H}_p(P_i) = I \cdot x^{-1}$.

If mapping $x \to I$ is a one-way injection:

1. Nobody can recover public key from the key image and identify the Signer;
2. Signer cannot make two signatures with different I's and the same x.

C Blockchain analysis resistance

In this section we cite CryptoNote whitepaper[1].

Standard CryptoNote transaction

Combining both methods (unlinkable public keys and untraceable ring signature) Bob achieves new level of privacy in comparison with original Bitcoin scheme. It takes him only to store one private key (a, b) and publish (A, B) to start receiving and sending anonymous transactions.

While validating each transaction Bob additionally performs only two elliptic curve multiplication and one addition per output to check if it belongs to him. For his every output Bob recovers one-time keypair (p_i, P_i) and stores it in the wallet. Any inputs can be *circumstantially proved* to have the same owner only if they appear in one transaction. But in fact this relationship is much harder to be established because of one-time ring signature.

With ring signature Bob can effectively hide his every input among somebody else's ones; all possible spenders will be equiprobable, even the previous owner (Alice) has no more information than any observer.

Pic. 7. Ring sinature generation in a standart transaction.

When signing his transaction Bob specifies n foreign outputs with the same amount as his output has, mixing all of them without the participation of other users. Bob himself (as well as anybody else) does not know if any of these payments has been spent: an output can be used in thousands signatures as ambiguity factor and never as target of hiding. The double spend check occurs in **LNK** phase when looking up in the used key images set.

Bob can choose the ambiguity degree on his own: $n = 1$ means that will have spent the output with 50% probability, $n = 99$ gives 1%. The size of resulting signature is linear $O(n+1)$, so the improved anonymity costs to Bob extra transaction fees. He also can set $n = 0$ and make his ring signature to consist of only one element: it will instantly reveal him as a spender.

D CPU-efficient ASIC resistant Proof-of-work

In this section we cite CryptoNote whitepaper[1] and CryptoNote Standart 008[11].

CryptoNight

We propose new memory-bound algorithm for proof-of-work price function. It relies on random access to a slow memory and emphases latency dependence. As opposed to scrypt every new block (64 bytes length) depends on the *all* previous blocks, not only one. As a result a hypothetical "memory-saver" should increase his calculation speed exponentially.

Scratchpad initialization

`First, the input is hashed using Keccak [KECCAK] with parameters b =`

1600 and c = 512. The bytes 0..31 of the Keccak final state are interpreted as an AES-256 key [AES] and expanded to 10 round keys. A scratchpad of 2097152 bytes (2 MiB) is allocated. The bytes 64..191 are extracted from the Keccak final state and split into 8 blocks of 16 bytes each. Each block is encrypted using the following procedure:

```
for i = 0..9 do:
    block = aes_round(block, round_keys[i])
```

Where aes_round function performs a round of AES encryption, which means that SubBytes, ShiftRows and MixColumns steps are performed on the block, and the result is XORed with the round key. Note that unlike in the AES encryption algorithm, the first and the last rounds are not special. The resulting blocks are written into the first 128 bytes of the scratchpad. Then, these blocks are encrypted again in the same way, and the result is written into the second 128 bytes of the scratchpad. Each time 128 bytes are written, they represent the result of the encryption of the previously written 128 bytes. The process is repeated until the scratchpad is fully initialized.

Memory-hard loop

Prior to the main loop, bytes 0..31 and 32..63 of the Keccak state are XORed, and the resulting 32 bytes are used to initialize variables a and b, 16 bytes each. These variables are used in the main loop. The main loop is iterated 524,288 times. When a 16-byte value needs to be converted into an address in the scratchpad, it is interpreted as a little-endian integer, and the 21 low-order bits are used as a byte index. However, the 4 low-order bits of the index are cleared to ensure the 16-byte alignment. The data is read from and written to the scratchpad in 16-byte blocks. Each iteration can be expressed with the following pseudo-code:

```
scratchpad_address = to_scratchpad_address(a)
scratchpad[scratchpad_address] = aes_round(scratchpad
  [scratchpad_address], a)
b, scratchpad[scratchpad_address] = scratchpad[scratchpad_address],
  b xor scratchpad[scratchpad_address]
scratchpad_address = to_scratchpad_address(b)
a = 8byte_add(a, 8byte_mul(b, scratchpad[scratchpad_address]))
a, scratchpad[scratchpad_address] = a xor
  scratchpad[scratchpad_address], a
```

Where, the 8byte_add function represents each of the arguments as a pair of 64-bit little-endian values and adds them together, component-wise, modulo 2^64. The result is converted back into 16 bytes.

The 8byte_mul function, however, uses only the first 8 bytes of each

```
argument, which are interpreted as unsigned 64-bit little-endian
integers and multiplied together. The result is converted into 16
bytes, and finally the two 8-byte halves of the result are swapped.
```

Result calculation

```
After the memory-hard part, bytes 32..63 from the Keccak state are
expanded into 10 AES round keys in the same manner as in the first
part.

Bytes 64..191 are extracted from the Keccak state and XORed with the
first 128 bytes of the scratchpad. Then the result is encrypted in
the same manner as in the first part, but using the new keys. The
result is XORed with the second 128 bytes from the scratchpad,
encrypted again, and so on.

After XORing with the last 128 bytes of the scratchpad, the result is
encrypted the last time, and then the bytes 64..191 in the Keccak
state are replaced with the result. Then, the Keccak state is passed
through Keccak-f (the Keccak permutation) with b = 1600.

Then, the 2 low-order bits of the first byte of the state are used to
select a hash function: 0=BLAKE-256 [BLAKE], 1=Groestl-256 [GROESTL],
2=JH-256 [JH], and 3=Skein-256 [SKEIN]. The chosen hash function is
then applied to the Keccak state, and the resulting hash is the
output of CryptoNight.
```

Our algorithm requires about 2 Mb per instance for the following reasons:

1. It fits in L3 cache (per core) of modern processors, which will come into mainstream in a few years;
2. A megabyte of internal memory is almost unacceptable size for modern ASIC pipeline;
3. GPUs may run hundreds concurrent instances, but they are limited in other ways: GDDR5 memory is slower than CPU L3 cache and remarkable for its bandwidth, not random access speed.
4. Significant expansion of the scratchpad would require increase in iterations, which in turn implies overall time raise. “Heavy” calls in a trustless p2p network may lead to serious vulnerabilities, because nodes are obliged to check every new block’s proof-of-work. If a node spends a considerable amount of time on each hash evaluation, he can be easily DDoSed by flooding with fake objects with arbitrary work data (nonce values).

Second level of special-purpose devices protection is using modern CPU built-in instructions like 64-bit multiplication and AES-NI. This method together with memory-bound approach makes current universal calculator (common PC) near optimal as regards ratio “price / efficiency” and equalizes all the participants.

E Adaptive parameters

Difficulty

CryptoNote has retargeting algorithm which changes the difficulty every block. This decreases system reaction time when network hashrate is intensely growing or lessening, preserving constant blocks rate. Original Bitcoin method calculates the relation of actual and target timespan between the last 2016 blocks and uses it as the multiplier for the current difficulty. Obviously it does not suit to rapid recalculations (because of large inertia) and results in oscillations.

General idea our algorithm is to sum all work nodes have done and divide it by the time they have spent. The measure of work is the corresponding difficulty values in each block. But due to unaccurate and untrusted timestamps we cannot determine exact time interval between blocks. A user can shift his timestamp forward into the future and the next time intervals might be improbable small or even negative. Presumably there will be not many such incidents, so we can just sort the timestamps and cut-off the outliers (i.e. 20%) . The range of the rest values is the time which was spent for 80% of corresponding blocks.

Size limits

Users pay for storing the blockchain and shall be entitled to vote for its size. Every miner deals with trade-off between maintaining costs and profit from the fees and sets his own "soft-limit" for creating blocks. Also the core rule for maximum block size is necessary for preventing flooding the blockchain with bogus transaction, but this value should not be hardcoded.

Let M_N to be the median value of the last N blocks sizes. Then the "hard-limit" for the size of accepting blocks is $2 \cdot M_N$. It averts the blockchain bloating but still allows the limit to slowly grow with time if necessary.

Transaction size does not need to be limited explicitly. It is bounded by the size of a block; and if somebody wants to create a huge transaction with hundreds of inputs/outputs (or with the high ambiguity degree in ring signatures) — he can do it with paying sufficient fee.

Excess size penalty

A miner still has the ability to stuff a block full of his own zero-fee transactions up to it's maximum size $2 \cdot M_b$. Even though only the majority of miners can shift the median value, there is still a possibility to bloat the blockchain and produce additional load on the nodes. To discourage harmful participants from creating large blocks we introduce a penalty function:

$$NewReward = BaseReward \cdot \left(1 - \left(\frac{BlkSize}{M_N} - 1\right)^2\right)$$

This rule is applied only when *BlkSize* is greater than minimal free block size which should be something like $\texttt{max}(10\text{kb}, M_N \cdot 110\%)$. Miners are permitted to create blocks of "usual size" and even exceed it with profit when overall fees suprass the penalty. But fees are unlikely to grow quadratically as penalty value so there will be an equilibrium.