Zano

Andrey N Sabelnikov

December 7th, 2018

Zano leverages the proven and time-tested cryptographic primitives that were first introduced with CryptoNote. Transactions are made both untraceable, and unlinkable by using stealth addresses and ring-signatures. As first implemented in Boolberry, downstream sender privacy is guaranteed by using output flags.

1 Hybrid PoS — PoW Consensus Mechanism

Zano uses a hybrid PoS — PoW consensus mechanism. This makes double-spend attacks both unfeasible and improbable. PoS was implemented to complement and enhance the security provided by traditional PoW blockchains. With the ability to "rent" hashing power, many PoW coins have been vulnerable to 51% attacks, as an attacker must only have a sufficient amount of hashing power to rewrite the longest chain. This requires a certain investment and the objective is to double spend the coins without regard to the impact of reputational damage, and consequent price decline. However, in a 51% attack on a PoW coin, the attacker has no interest or concern regarding the coin's value beyond the initial attack.
Similarly, pure PoS coins have their weaknesses. While there is an inherent disincentive to attack a network that you have a financial interest in, pure PoS protocols face certain pitfalls such as "nothing at stake", "stake grinding", among others. Proposed solutions include slashing, improving validator randomization, staking specific tokens, and forgers, yet there remain a variety of challenges with each of them.
Any attack on Zano's hybrid parity protocol, would require an attacker to obtain not only a majority of hashing power, but also a majority of the coins involved in staking. As a result, the risk/reward profile shifts, such that a 51 % attack on a hybrid network has a negligible probability of economic benefit.

1.1 Classic Proof-of-Stake

PoS mining is typically implemented such that a random coin owner obtains the right to sign a new block with their secret key. The process of verification compromises full anonymity, because anyone can verify the signature using this owner's public key. Nevertheless, it is possible to preserve *untraceability* and *unlinkability*.

1.2 Zano's PoS Implementation

Ring signatures allow the transaction creator to provide a set of possible public keys for signature verification, thus keeping their identity indistinguishable from other users. The concept of an anonymous, secure PoS mechanism seemed to be unachievable.
The basis of PoS is a so-called *kernel*, which is a data structure that depends on the transaction output and includes:

- Current timestamp with 15-second granularity.
- Key Image, which corresponds to each transaction output. A *keyimage* is comprised of 32 pseudo-random bytes derived from the key in such a way that it is impossible to reconstruct the key, given only its image.
- Stake Modifier. An additional 64 pseudo-random bytes derived from the last PoS and PoW blocks, which disallows any predictability of the *stakemodifier* in the blocks ahead.

During mining, a user is allowed to sign a block, if $Hash(kernel) < CoinAmount * PosTarget$, where $CoinAmount$ is the amount of coins in a particular output, and $PosTarget$ is an adaptive parameter that works to keep the block creation rate constant.

The PoS miner iterates the timestamp (within the allowed boundaries) for each of their unspent outputs (UTXO) and checks to see if they possess a UTXO that's *keyimage* satisfies the *PoSTarget* formula

above. In the event of a "*winning*" result, they spend this particular output, anonymously, with a ring-signature.

Note: The signature includes the *keyimage* (used in the kernel), but not the key itself, which is why it does not compromise anonymity. The miner signs both the transaction and the block and broadcasts the new block to the network.

1.3 Design Considerations

Here are some of the factors that were incorporated in our design decisions:

a. **Kernel Structure:** Unlike the classic PoS where the kernel includes a direct link to the winning output, an anonymous transaction system (Zano) cannot use this approach, since it compromises privacy. This data was replaced with a *keyimage*, which literally is a hash of an output public key. Thus, the key image contains 32 bytes of pseudo-random data related to the *winning* output.

b. **Stake Modifier:** The purpose of *stakemodifier* is to make the kernel unpredictable, thereby protecting it from machinations. The *stakemodifier* is recalculated for each new block. The value is the concatenation of the last known PoS block's *kernel hash* and the last known PoW *block ID*. Both structures are essentially hashes, i.e. unpredictable bytes. It should be noted that a pure PoS (without PoW) network would not be secure under such rules, (typical attacks are described here:
https://download.wpsoftware.net/bitcoin/old-pos.pdf), but our hybrid model is safe.

c. **Age of Coins:** The age of coins has no impact on the probability of mining a PoS block. Restricting coins eligibility to participate in staking has two direct impacts. While reducing the coins in circulation can increase scarcity and possibly price, project's that employ these requirements tend to be *illiquid*, which tends to increase volatility, and makes them less attractive than a free-flowing market with price discovery and liquidity. Additionally, this creates a more equitable environment for users; we consider egalitarianism a condition precedent to acceptance and wide scale adoption, both on PoS mining and PoW mining through our ASIC-resistant hash function, Wild Keccak [11].

1.4 Cumulative Difficulty Adjustment

As described above, any attack on Zano's hybrid parity protocol, would require an attacker to obtain a majority of both hashing power and a majority of coins.
To enforce this hybrid parity protocol, the core will always prefer a chain with alternating work types and we have developed a method of adjusting the weighting of successive PoS or PoW blocks such that the core will always prefer the chain with alternating work.

The following rules were applied:

- Cumulative difficulty (CD) is used in determining which chain is "better."
- The cumulative difficulty for each chain is the weighted sum of each block's difficulties:

$$CD = \sum_{i=0}^{h} w_i \cdot d_i$$

- Weights are chosen to favor a chain in which blocks alternate in type: $PoS - PoW - PoS - PoW$ etc. In this ideal case each $w_i = 1$
- Weights decrease at rate $r = 0.75$ in cases of successive blocks of the same type.

For example, a chain PoS - PoW - PoW - PoW - PoS - PoS has the following cumulative difficulty:

$$CD = (d_1 + d_2 + r \cdot d_3 + r^2 \cdot d_4 + d_5 + r \cdot d_6)$$

These rules may lead to a situation where a 10 block-long chain will have less cumulative difficulty than an 8 block-long one. This may be surprising, but it derives from the assumption that an attacker with only one overwhelming type of mining power (even 75% of PoS or PoW) will not be able to overpower the rest of the network.

2 Aliases

Each Zano user can register with an alias, for example: @easytouse, a human-readable name associated with a payment address and text comment, which is stored in the blockchain. This alias provides a short, easy-to-remember name rather than a long and confusing string of random characters. Blockchain storage ensures that aliases are protected from being altered or commandeered.

An alias registration is a special kind of transaction that includes a record of an alias assignment. The system core validates the record and checks the availability of the name. Once validation is successful, the transaction reaches the blockchain and a new record is created in the alias database.

Zano users can easily send transactions to an alias: their wallet automatically checks whether the name is registered in the blockchain, and then obtains the associated public keys to which the transaction will be sent. In the event of a missing alias, the wallet will generate an error report and no funds will be sent or lost.

Aliases can be used for more than just Zano transactions. Think of them as a decentralized address book with universal IDs that can be used for various services based on the Zano platform.

To reduce possibility of phishing we set limitations on alias registrations. Users can use any combination of the lower-case Latin letters a-z and the Arabic numerals 0-9. Additionally, there is a length minimum of 6 characters, and a maximum length of 12.

3 Encrypted Attachments

3.1 Additional Data Fields

Zano allows for including arbitrary data in transactions; this data can be stored in the blockchain permanently or removed after passing checkpoints, similar to the pruning mechanism that was first implemented in Boolberry. The purpose of pruning was initially to manage blockchain bloat, but we have extended this to include arbitrary data that similar to ring signatures, have a diminished relevance over time This gives developers the power to build a variety of applications based on the Zano blockchain platform.

For this purpose, we implemented two types of transaction data fields:

- **Transaction Extra:** This field contains data that is included in the transaction prefix. The prefix is the first part of any transaction; it stores essential information about the financial transaction. Data in the prefix is completely hashed during the generation/validation of transaction signatures, which is why it cannot be later removed from the blockchain.
- **Transaction Attachments:** This field contains additional data that does not need to be kept in the blockchain permanently. A couple of examples are transactional data (including escrow proposals), comments on transactions, or random data used by third-party services. Old

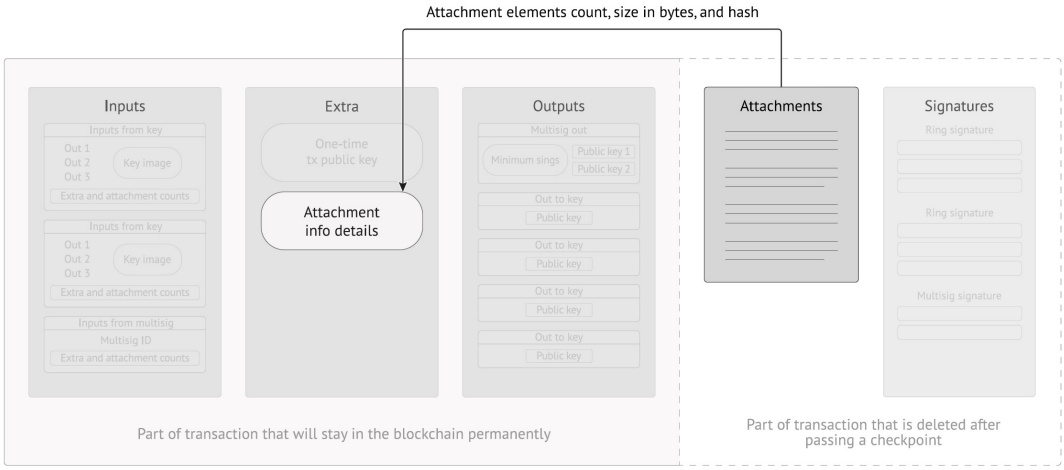

Figure 1

transaction attachments are deleted (voluntarily removed from the local hard drive) after every checkpoint has been passed, meaning the core will not check them again. To protect data from being altered, we have put the transaction attachment hash into the transaction extra field, so that the deleted data can always be verified, even after passing checkpoints.

3.2 Encryption

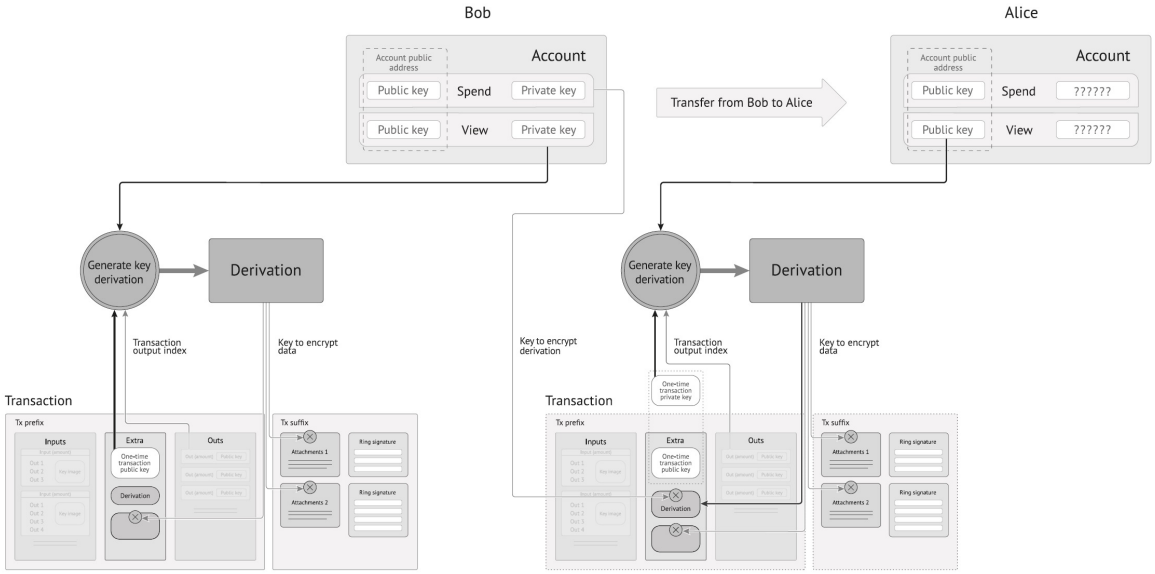

Figure 2

For many tasks, such as text comments in transactions, special fields in escrow proposals, etc., we need to encrypt data included in transactions (*attachments* / *extra*). However, all keys should be retrievable, for both the sender and recipient, in the event of hard drive failure or recovery of a wallet

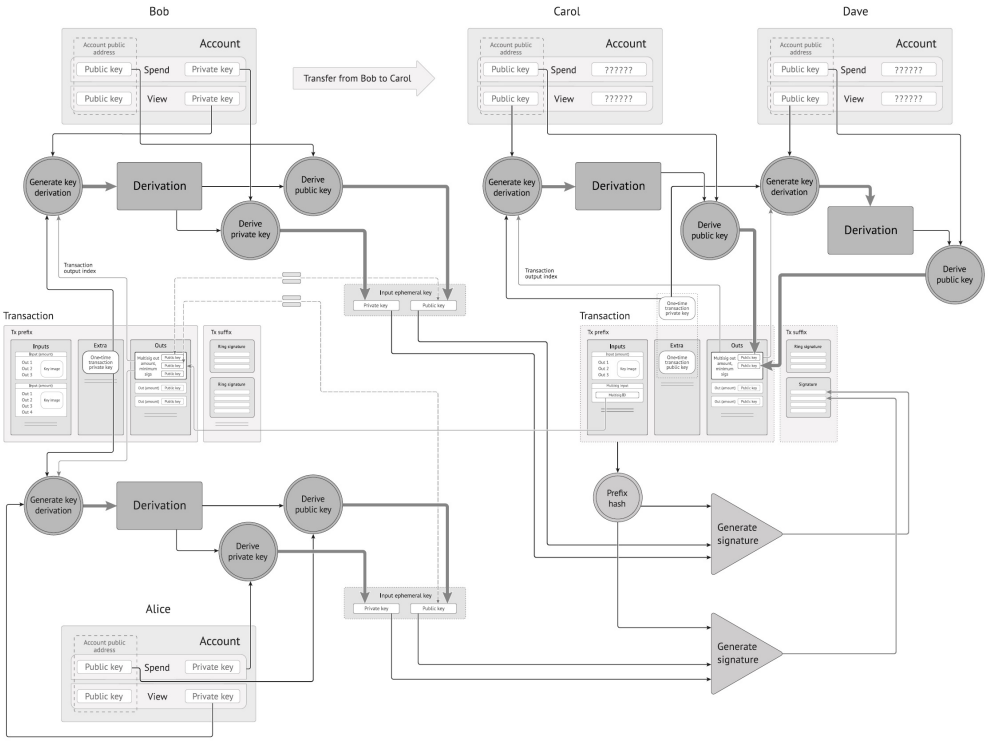

Figure 3

from a seed phrase (brain wallet). Essentially, they should be able to be derived from the main wallet key and data in the blockchain.*

In each transaction we already have a common secret element for both participants, known as a derivation. This is part of an ephemeral key, which is produced using the Diffie-Hellman protocol for unlinkable payments (see section 1) [4][7]. All transferred data is encrypted via a block cipher ChaCha using the key obtained from the derivation. (Figure 3)

The Challenge is that while the recipient can always calculate a derivation, given only an incoming transaction, the sender cannot. Usually, the sender's first step when preparing a transaction is to generate a random number (without saving it) to produce the derivation. The next time the sender sees this transaction in the blockchain, he cannot re-calculate the same derivation without that randomly generated number.

To overcome this, we allow senders to store the derivation in the transaction and encrypt it using their own secret key. As a result, both sender and recipient can reconstruct the derivation and decrypt their shared data from the transaction, and no private information is ever published.

4 Multi-Sig

Multi-signature, or simply *multisig*, is a commonly used term for special types of transactions in which money can only be redeemed jointly by several recipients. $MofN$ *multisig* means that a transaction (namely, the identifier of its specific output) contains N keys, which belong to the *multisig* users, and at least M of these users (key owners) must cooperate by providing their signatures in order for the cryptocurrency to be released.

Zanos *multisig* feature is extremely useful for user security. For example, this feature allows a user to hold their coins in a 2-of-2 *multisig*, where both keys belong to this user, but are stored on different devices In this case, the user obtains a two-factor authentication mechanism, which protects their wallet from a single key compromise.

Another popular feature of Zano is the escrow service, which will be described later in section 6.

Technically speaking, a *multisig* output differs only slightly from a standard output: it contains N keys (instead of one key) and an M number. Developers' work with *multisig* usually relates to 1) the consolidated creation of transactions, or how to make the process of joint creation and signing more user friendly; and 2) the development of a key and output indexing structure.

4.1 Indexing

In the interest of simplicity, we have introduced a new method of multi-signature output identification. Currently, existing cryptocurrencies employ one of the two most commonly used identification methods:

a. Identification by the transaction hash plus the order number of an output inside the transaction: Here we would need to maintain a global index of all transactions. The downside to this approach is that if a transaction is changed (new inputs or outputs are added), its hash also changes. Therefore, we cannot refer to an output (using its identifier) until the transaction has been finalized, which disallows any subsequent changes.

b. The global index of outputs method (CryptoNote, [2]): The amount of the output plus its order number among other outputs with the same amount. This method optimizes the support of the ring signature feature, which ensures the *untraceability* of payments. The challenge here is that we do not know the output order number until the transaction reaches the blockchain (another transaction can be confirmed prior to it, thus altering the global index of outputs).

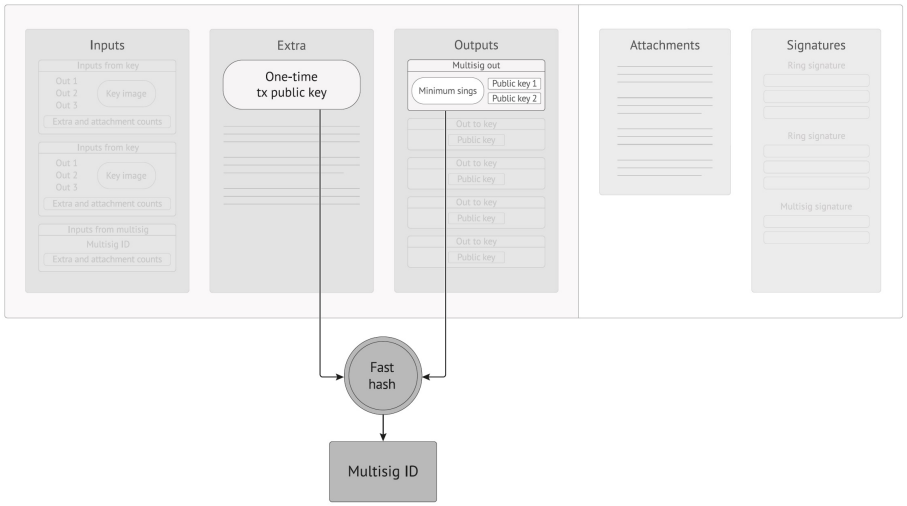

Figure 4. Multi-Sig Identification

To ensure the system supports the escrow service, (see next section) we must be able to make a reference to each *multisig* output before its transaction is fused into the blockchain and even before its transaction is in its final form. Specifically, we want to include a template for the future transaction B

as an attachment to transaction A, so that the embedded transaction B refers to transaction A "from within."

As a unique multi-sig output *ID* we use:

$ID = H(txPubKey||Out)$

where:

- *ID* is a multisig output identifier;
- H is a cryptographic hash function;
- *txPubKey* is a one-time transaction public key;
- *Out* is the output body;
- $||$ is the concatenation operation.

Due to the random nature of *txPubKey* and hash mapping properties, we get non-repeating identifiers for *multisig* outputs (Figure 5) In order to prevent fraudulent operations, where attackers generate identical public keys and outputs to create identical *IDs*, the program's core tracks the uniqueness of each *multisig ID*.

4.2 Consolidating Transaction Terms

One of the most important conditions for enabling the escrow service on the platform is to provide the ability for two or more users to jointly create a transaction. This way, payments are atomic (all or nothing) and each participant's funds are protected from theft, being frozen, or intentional money burn.

Figure 6 illustrates the basic principles in making a joint transaction. For example, Alice and Bob agree to make a consolidated transaction, which sends money from each of them to a 2-of-2 *multisig* (meaning that only together, can Alice and Bob release the funds).

a. Alice prepares the main *multisig* output, adds all relevant data (her inputs, outputs for balance of payments, attachments, etc.) and signs the transaction. She also uses a special flag TX FLAG SIGNATURE MODE SEPARATE, which will activate the core's special advanced mode of transaction signature protocol verification.

b. At this point the transaction is incomplete: the total output sum is greater than the total input sum. Bob needs to fund his portion of the transaction, so Alice sends a "*transaction template*" (transaction information) to Bob as a special attachment.

c. If Bob accepts this proposal (*transaction template*), then Bob funds his portion of the transaction and signs it. Once the *transaction template* is fully funded, and signed, the transaction is considered valid and can be broadcast to the network and subsequently included in the blockchain.

Security notes:

- To protect Alice's attachment's from being altered, the data hash is stored in the corresponding transaction input, which is sealed by and with Alice's signature.
- Until the transaction has been confirmed by the network, none of the participant's funds are considered spent. Once the transaction proposal is created, the funds used in this transaction are temporarily locked in the wallet, this mitigates potential conflicts regarding other coins being spent from the same wallet. If the proposal (*transaction template*) is rejected by the counterparty,

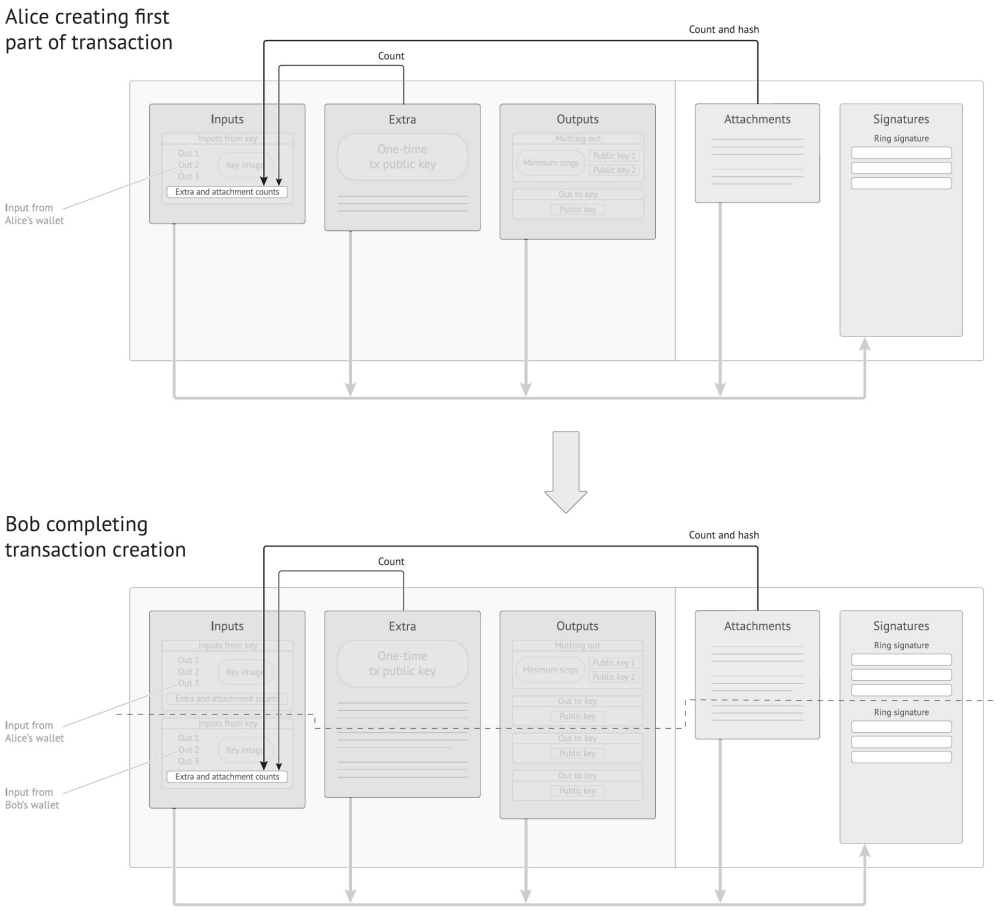

Figure 5. Creation of Consolidating-Transaction

or expires, the coins will be unlocked in the wallet from which they were sent, without any further action.

- If Bob (or anyone else) were to attempt to interfere with or obstruct Alice (for example, locking Alice's money by publishing the transaction with incorrect data), they would need to make the transaction semantically correct, which would require adding their own coins to the inputs. This requirement creates a disincentive that detours malicious behavior.

5 Escrow

Escrow, like its name, is a mechanism that was designed to facilitate secure, anonymous, payments between counter-parties. Traditionally, the term "escrow" refers to an independent "trusted" third party that is tasked with acting as an intermediary to oversee and provide mutual assurances to each party that their agreement is executed in a manner that is satisfactory to all parties. Their fiduciary duty includes carrying out the agreement as it was intended, but also allows for modifications with mutual consent. Finally, the "escrow" is responsible for disposition, transfer, or distribution as a final step in the process.

Trust and reputation create value for participants, and justify fees commensurate with such escrows, along with warranties, both expressed and implied. Certain aspects of an escrow are difficult to aug-

ment or mechanize. However, we have developed a system that is intended to let participants chose, from a variety of variables, the structure that best suits them and the nature of their respective transaction.

Zano provides the framework for a secure and private transaction without the need for a trusted third party. We anticipate market participants are best suited to choose how the framework is applied to facilitate their objectives relative to consummating a transaction. Our Escrow system, as proposed, will require participants to make additional deposits, which they will forfeit if there is any attempt to act maliciously, or in a way that is contemptuous toward their counterparty. In the absence of a rating or feedback system, similar to eBay or other escrow types, we must rely on financial incentives to augment credibility and integrity in the absence of a trusted third party.
The main idea is as follows:

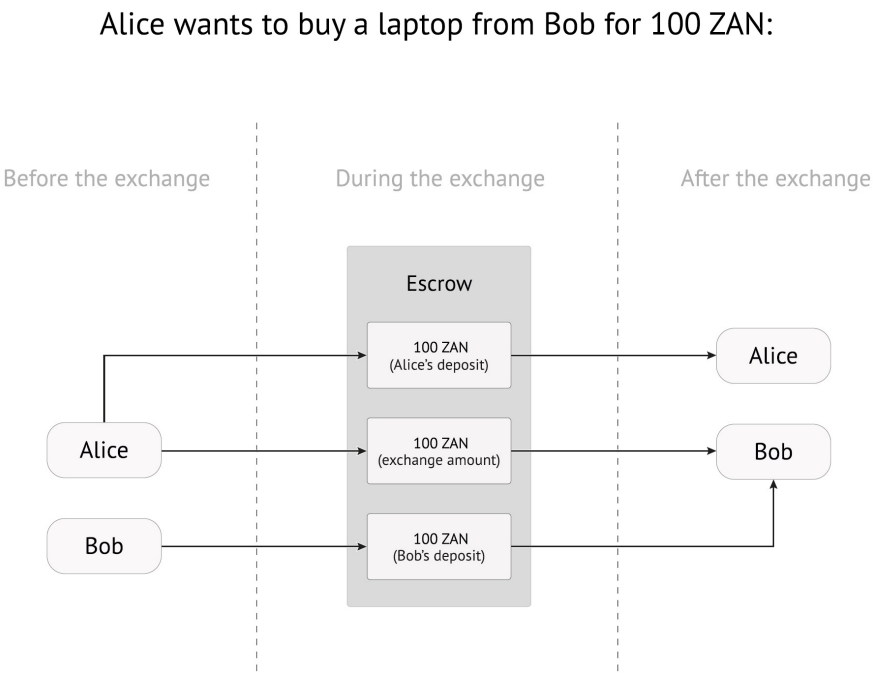

Figure 6

a. Alice wants to buy an item from Bob for 100 ZAN.

b. Alice prepares an escrow *transaction template* that will transfer a total of 300 ZAN from both participants on a 2-of-2 *multisig*. The first key belongs to Alice and the second key to Bob.

c. Alice can include details, including delivery instructions, messages etc., all of which will be encrypted and only retrievable by Bob. This data type was described above as *transaction attachments* and while the hash of the attachment will be verifiable indefinitely on the blockchain, the actual *attachment data* will fall off after passing checkpoints to manage blockchain bloat as this information at some point in the future loses its relevance. For example, the shipping details including name, address etc. on a package you received a year ago is no longer relevant.

d. Alice adds her inputs with 200 ZAN and sends the proposal transaction template (which is not yet semantically correct) to Bob for his consideration.

e. Upon acceptance, Bob adds his input for 100 ZAN and broadcasts the transaction (which now becomes semantically correct) to the network. At this point the funds involved in the transaction are locked.

f. Bob, pursuant to their agreement, sends the item(s) to Alice and waits for a response.

g. If Alice is satisfied with the order, both users sign a new transaction, which unlocks the money and sends 100 ZAN back to Alice and 200 ZAN to Bob.

h. Other scenarios may include: Alice sends the goods back to Bob, both get their money back, or Alice keeps defective goods, asks for compensation and Alice and Bob share the money 50/50, and so on.

i. If anyone breaks the escrow (Bob does not send any goods or Alice receives the shipment, but refuses to sign the money unlocking transaction), both will lose more than they would receive (Bob loses 100 ZAN or Alice pays twice for the goods). Therefore, it is in everyone's best interest to play fair.

Note: The size of locked deposits is determined by mutual agreement between the merchant and the customer.

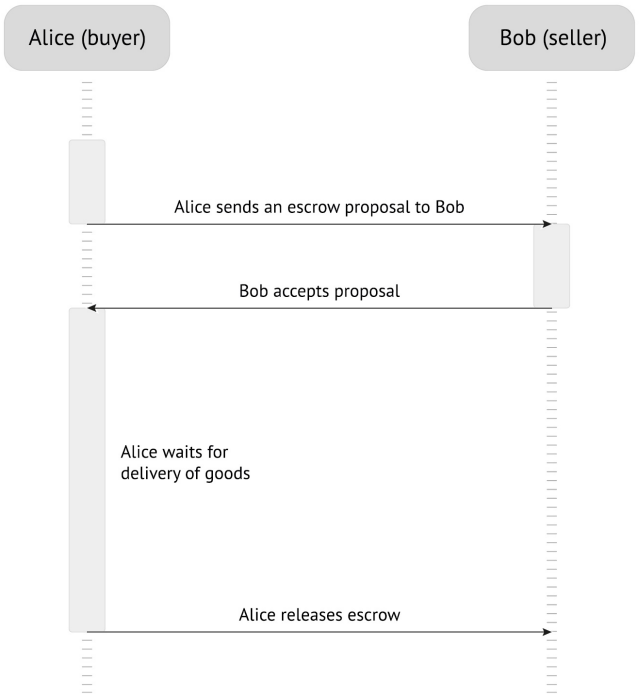

Figure 7

5.1 Escrow Proposal

The main steps here are an escrow transaction. The buyer sends an incomplete "*transaction template*" to the merchant / counterparty (we refer to this as the "proposal"). They cannot do so directly, since establishing direct connections tends to be inconvenient and, more importantly, destroys anonymity. Additionally, these transactions cannot be sent to the network using broadcast packets, as this would

flood the network with excessive traffic. This is where "*transaction attachments*" come in (see section 4).

The buyer prepares a "proposal" also referred to as a "transaction template" (Figure 9), packs it, and stores it in the attachment field of a special "transport/carrier" transaction (the transferred amount of money in such a transaction can be as low as the buyer wants, or even without any outputs). Additionally, the buyer encrypts the attachment using the mechanism described in section 4.2 in this paper, to preserve privacy. Once this is complete, the transaction enters the blockchain and the merchant can then decrypt the attachment and proceed with the escrow transaction which remains in the first transaction's input.

IMPORTANT: Attachment data is not stored in the blockchain forever but is removed after passing each checkpoint. The only "excess data" is the 32-byte hash of the attachments. At the same time, the escrow attachments can be arbitrarily large (the only limit is the systems current maximum transaction size) and can contain any necessary information: order ID, delivery address, deposit sums, payment IDs, etc.

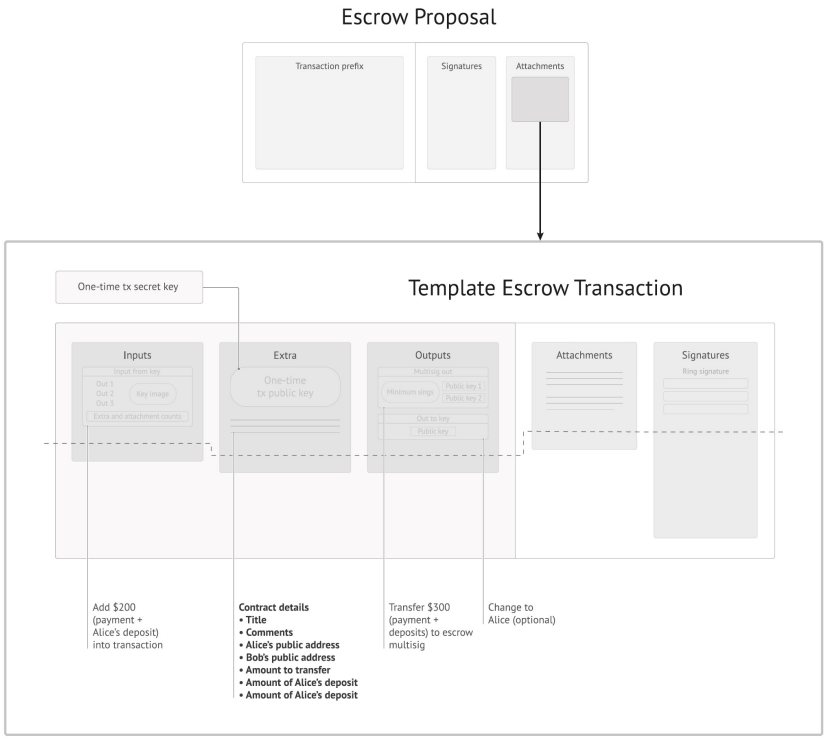

Figure 8

5.2 Escrow Response

The last steps are simple. Once the merchant receives the escrow template, they can decline it (no money will be locked, no additional actions are needed) or accept it (by completing the escrow transaction and shipping the order to the buyer). Prior to sending purchased goods to the buyer, the seller must prepare a "release transaction template" which will send 100 ZAN back to Alice and 200 ZAN to Bob, then encrypt it with Alice's key and store it in the attachment field of the escrow transaction (see figure 10).

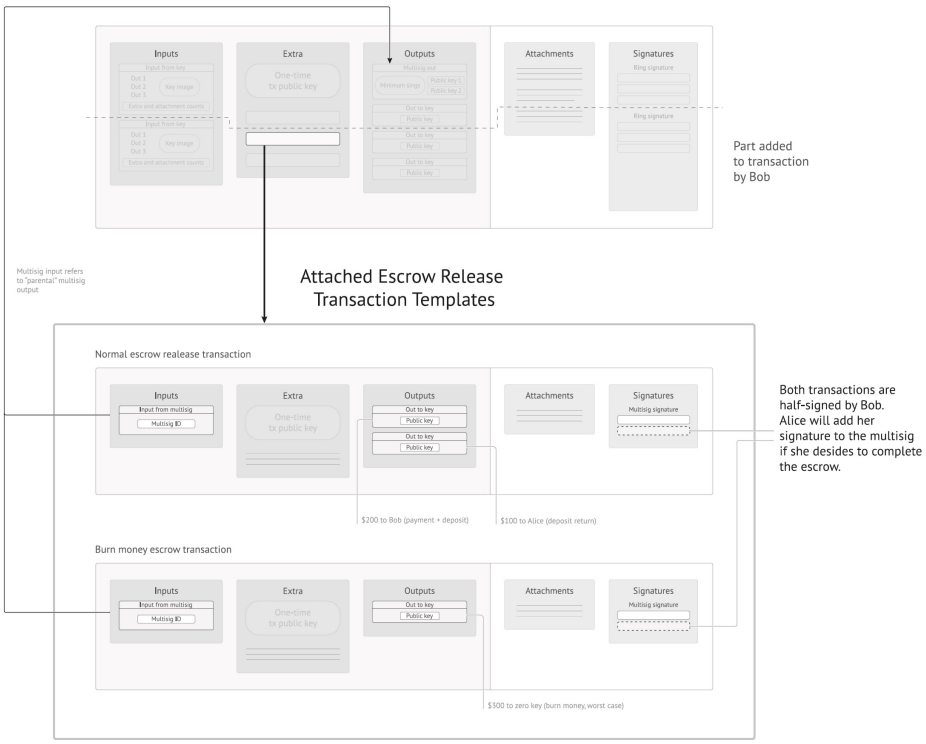

Figure 9

The buyer receives the template of the second transaction and then gives their signature upon delivery of the goods purchased from the merchant to release the money. Altogether, we employ a three-round protocol with no direct connection between participants.

6 Code Design

Zano's methodologies are aimed at maximizing reliability, while maintaining a flexible and secure architecture.

- **Component-based Modular Structure:** There are three main software components: the daemon, miner, and the wallet. The daemon connects to the network, validates and exchanges new blocks and transactions, and maintains the local blockchain database. The daemon is the main component, sometimes referred to as the node. The miner is a subprogram, that performs mining tasks and produces new blocks. The wallet stores users' private keys and provides an interface for creating transactions. The wallet connects to the daemon independently, which provides the following advantages:

  a. The daemon is the only component connected to the network, but it does not store private keys. This means it is easier to ensure security since keys are separated from the network.

  b. A single daemon can serve two or more wallets. This is particularly important for enterprise solutions, such as web-wallets or payment processors that work with multiple keys at the same time.

  c. Any modifications affecting the mining algorithm or user wallet features (GUI, keys encryptions) do not affect the daemon's operation.

- **Forward and Backward Compatibility:** When updating the software and enabling new features, it is of great importance to maintain compatibility between old client software versions and new versions Usually when a "new client" sends a modified (new) command or message to an "old client," the old client software does not recognize it. Zano's protocol allows older versions of client software to identify portions of data in the new versions of messages that correspond to previous versions of messages (older protocol). Obviously, the client software of older versions ignores the new data in new versions of messages, because it cannot handle such data. Nevertheless, this feature helps to avoid the typical problems of evolving the protocol and adding new product features.

- **Asynchronous Core Architecture:** Most Cryptonote projects use an interlocked multithreading model: each incoming network or RPC call, is handled by the core, and passes through a mutex lock, which only allows one thread to work with the core simultaneously. In the event of multiple requests from the network or RPC calls (high transaction flow or high-load service requests) all requests will be handled consecutively, one-by-one, which significantly limits network throughput. Our new core is a fully asynchronous architecture. This allow multiple network / RPC requests to be handled in parallel. For example: Even if the atomic operation of "adding a block" to the core is processing, the core still can handle parallel RPC/network requests simultaneously, and without limitations. This was accomplished by a deep refactoring and improvements to the multithreading model of the core.
  Another strength of the new asynchronous core is that the transaction pool is no longer blocked during the operation of adding a new block to the core. In classic CryptoNote implementation, transaction pool is locked for the entire "Add New Block" operation (which can take several seconds, in case of big blocks). During times of high transaction flow, while the core is processing larger size blocks, validation of these blocks can cause the transaction pool to be blocked for up to several seconds. During this seemingly short period, connections from other nodes while relaying new transactions would be blocked, which can cause connection time-outs, which are damaging to network connectivity, throughput, and impact network propagation. While in this sub-optimal state, transfers can be rejected, and chain-splits / orphan blocks become more likely. We were faced with this exact situation while conducting stress tests and we able to solve this issue by making transaction pool lock independently, and for a particularly fast, individual, operation, thereby eliminating the bottle neck associated with the "Add New Block" operation.

  To illustrate this how significant of an improvement this represents, we will show screenshots from our diagnostics tool, which monitors connectivity of the network:

  Colored dots are network nodes and lines between them represent connections. The color and width of the lines reflect connection age:

  - Newer connections are illustrated with a thin red line.
  - Older connections are shown with thin blue line.
  - Stable long-term connections are highlighted by bold blue lines.

  On the left side you can see the result of stress-testing one of our old test networks with, literally, hundreds of thousands of transactions. Most connections are timing out and reconnecting, and the network is suffering from these time outs and the bad relaying of transactions / blocks. On the right side is an example that illustrates a network in perfect condition all connections are stable, ongoing transactions and blocks are being relayed without any glitches or gaps.

- **Note regarding Core tests:** The risk of a "*bug*" in the currency core has the potential to be devastating, and with ongoing development, even the most cautious approach introduces the possibility of human error.

  With this fact in mind, we expended an extraordinary amount of effort on rigorous "*core tests*" –large set of project specific unit-tests, which cover most of the currency core and wallet codebase. Almost every currency rule that is validated by the core has a corresponding core test

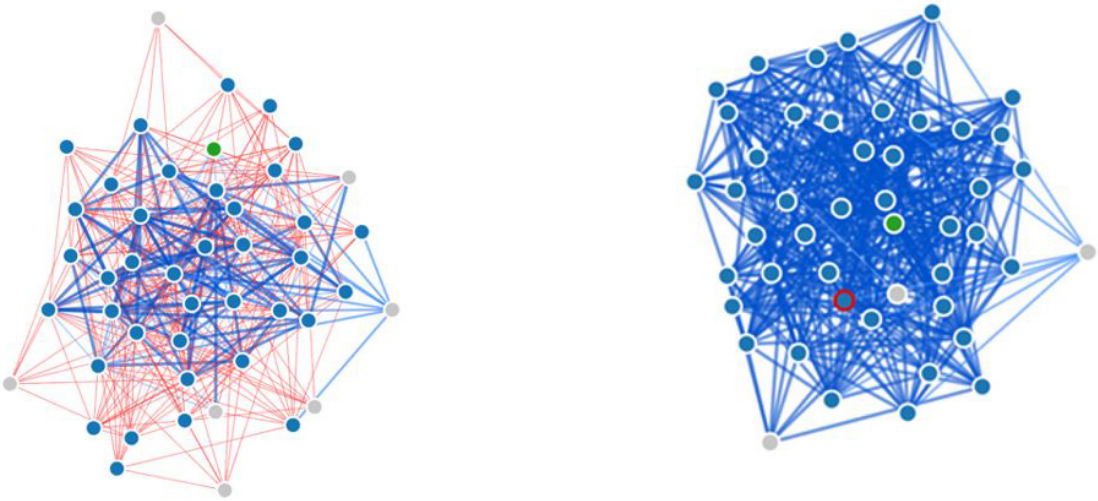

Figure 10. Network Connectivity Comparison

which is designed to stress or attempt to break the system, and in the event of any unexpected core behavior, the core tests will report any problems. Each time we found a problem or bug in the core, we would implement a core test which simulates that same circumstance and add it to our testing suite. We use this suite routinely in our automatic regression tests, so after each commit, all the tests are run on the server to validate full functionality, which mitigates the chance of human error.

REFERENCES

a. Satoshi Nakamoto, "Bitcoin: A peer-to-peer electronic cash system,"
URL: https://bitcoin.org/bitcoin.pdf

b. Nicolas van Saberhagen, "CryptoNote v 2.0," URL: https:// cryptonote.org/whitepaper.pdf, October 17, 2013

c. Nicolas van Saberhagen, Johannes Meier, Antonio M Juarez, "CryptoNote Signatures," URL: https://cryptonote.org/cns/cns001.txt, December 2011

d. Nicolas van Saberhagen, Seigen, Johannes Meier, Richard Lem, "CryptoNote One-Time Keys,"
URL: https://cryptonote org/cns/cns006.txt, November 2012

e. Whitfield Diffie and Martin E. Hellman, "New directions in cryptography,"
URL: https://ee.stanford.edu/˜hellman/publications/24.pdf, November 1976

f. Daniel J Bernstein, "ChaCha, a variant of Salsa20,"
URL: https://cr.yp.to/chacha/chacha-20080128.pdf

g. Andrew Poelstra, "Distributed Consensus from Proof of Stake is Impossible,"
URL: https://download.wpsoftware.net/bitcoin/old-pos.pdf, May 28, 2014

h. URL: https://boolberry.com/files/Boolberry_Solves_CryptoNote_Flaws.pdf

i. URL: https://boolberry.com/files/Block_Chain_Based_Proof_of_Work.pdf