Dusk Network Protocol Specification

Toghrul Maharramov*
toghrul@dusk.network
DRAFT v1.1

January 28, 2019

Abstract

Dusk Network is a protocol based on *The Dusk Network And Blockchain Architecture* [Ven18]. Dusk Network is an anonymous cryptocurrency, utilizing the latest breakthroughs in cryptography to guarantee user anonymity. Built on top of a novel consensus protocol - SBA★, Dusk Network offers unrivaled scalability and decentralization for an anonymous platform. This paper outlines the Dusk Network protocol specifications.

*Senior Researcher, Dusk Network

Contents

1 Introduction

This paper is the official reference document of the Dusk Network protocol. While the author aims to highlight the protocol features in detail, the document is not a complete representation of the Dusk Network reference implementation and has to be used in combination with the reference code.

The reader is expected to have a basic understanding of the relevant subjects. The future versions of the document will expand on the principles described in the document and will be adapted to feature more entry-level details on the subjects involved.

NOTE: This is a **DRAFT** version of the document and may contain typos and mistakes. The reader is urged to contact the author through the e-mail provided in the title page if any typos or mistakes are encountered.

2 Notations

2.1 Notations

$r \geq 0$ - the current round number (block height)
$s \geq 1$ - the current step in round r
MAXSTEPS_{BLOCK} - the maximum number of steps s permitted by the protocol
MAXSTEPS_{SIGSET} - the maximum number of steps s permitted by the protocol
sk_i - the secret spend key of the user i
vk_i^s - the secret view key of the user i
pk_i - the public spend key of the user i
vk_i^p - the view key of the user i
$CERT^r$ - the certificate of the block r
S^r - the seed of the round r
$\underline{B^r}$ - the block generated in round r
$\overline{B^r}$ - the verified block B^r with $CERT^r$
l^r - the leader of the round r, l^r produces the block candidate B^r in the round r

2.2 Functions

$\mathcal{H}(x) : \{0,1\}^n \rightarrow \{0,1\}^k$ - the cryptographic digest (hash) of a fixed length k of the message x of an arbitrary length
$\mathcal{H}^{\textbf{SHA-3}}(x)$ - SHA-3 hash function
$\mathcal{H}^{\textbf{MiMC}}(x)$ - MiMC hash function
$\Sigma^{\textbf{BLS}}(x) : (x, sk_i^{\textbf{BLS}}) \rightarrow \{0,1\}^k$ - the signature of a fixed length k of the message x produced with secret key $sk_i^{\textbf{BLS}}$ of the user i
$\Sigma^{\textbf{EdDSA}}(x) : (x, sk_i, r) \rightarrow (S, R)$ - the signature of the message x produced with secret key sk_i of the user i and a random number r
$sig^{\textbf{BLS}}(x) \triangleq \langle \Sigma_{sk}^{\textbf{BLS}}(x), pk_i^{\textbf{BLS}} \rangle$
$sig^{\textbf{EdDSA}}(x) \triangleq \langle \Sigma_{sk}^{\textbf{EdDSA}}(x), pk_i \rangle$
$\mathcal{T}(PAY^r)$ - the Merkle Tree constructed from the payset PAY^r
$\mathcal{T}^{\mathcal{R}}(x)$ - the Merkle Tree root of $\mathcal{T}(PAY^r)$
$Com(x) : (x, a) \rightarrow \{0,1\}^k$ - the Pedersen Commitment of a fixed length k of the value x with a random blinder a

2.3 Sets

BID^r - the set of eligible bids at the beginning of round r
PK^r - the set of public keys belonging to the eligible Provisioners at the beginning of round r
$SV^{r,s}$ - the voter set during round r, step s
PAY^r - the payset contained in B^r
BC^r - the block candidate set in round r

3 Concepts

3.1 Blockchain

Originally proposed in [Nak08], blockchain is a distributed ledger comprised of sets of data (blocks) linked together with the use of cryptographic primitives. Among other essential data, each block contains a cryptographic digest of the previous one, thus ”chaining” the blocks together.

The first block in the blockchain is called a genesis block (B^0). The genesis block is the ancestor to all the blocks in the chain. Dusk Network genesis block seed (S^0) is going to be computed with a MPC (multi-party computation) procedure outlined in the upcoming SBA★ technical paper.

A single block can be valid at one particular height (r), giving blockchain the following structure:

$$B^0 \leftarrow B^1 \leftarrow ..... \leftarrow B^{r-1} \leftarrow B^r$$

3.2 Block

3.2.1 Header

Block header contains block metadata. The header has a constant size of 145 bytes with the cryptographic digest of a header acting as a ”link” between blocks:

$$\texttt{prevBlock} = \mathcal{H}(B^{r-1}_{\mathbf{H}})$$

The header has a following structure:

Field	Size	Description
`version`	1 byte	The version of a block.
`timestamp`	8 bytes	The *Unix* time at which the block has been created.
`height`	8 bytes	The height of the block. Equivalent to r (round).
`prevBlock`	32 bytes	The cryptographic digest of the previous block.
`seed`	32 bytes	CRS derived from the ephemeral key of the block producer.
`txRoot`	32 bytes	Merkle root of the transaction set tree ($\mathcal{T}(PAY^r)$).
`cert`	32 byte	The cryptographic digest of the certificate ($CERT^r$).

Table 1: Block header structure

3.2.2 Body

Block body contains the transactions included in the respective block. The body MUST include a Coinbase transaction (outlined in **Section 3.5.1**) in index 0, which could be followed by other transaction types. There can only be one Coinbase transaction per block.

3.2.3 Certificate

Block certificate contains the witness data required to prove the validity of the block. The certificate consists of 2 batched BLS signatures (outlined in **Section 4.7.2**), 2 step counters indicating the step number at which the consensus has been achieved for Block Reduction and Signature Set Reduction phases respectively, 2 sets containing the lists of public keys that had casted their votes in the successful iterations of the aforementioned phases, as well as 2 sets of sortition proofs (outlined in **Section 5.X.X**) corresponding to the public key sets.

The certificate has the following structure:

Field	Size	Description
`brBatchedSignature`	32 byte	Batched signature of the Block Reduction phase.
`brStep`	1 byte	Block Reduction phase step.
`brPublicKeys`	?	Set of $pk^{\mathbf{BLS}}$ corresponding to `brBatchedSignature`.
`brSortitionProofs`	?	Set of proofs corresponding to `brPublicKeys`.
`srBatchedSignature`	32 byte	Batched signature of the Signature Set Reduction phase.
`srStep`	1 byte	Signature Set Reduction phase step.
`srPublicKeys`	?	Set of $pk^{\mathbf{BLS}}$ corresponding to `srBatchedSignature`.
`srSortitionProofs`	?	Set of proofs corresponding to `srPublicKeys`.

Table 2: Block certificate structure

3.3 Addresses

3.3.1 Spend Address

$$spendaddress_i = sk_i + vk_i^s$$

The secret view key is derived from the secret spend key, by simply hashing the secret spend key (though the user may opt to derive the secret spend key and the secret view key independently):

$$vk_i^s = \mathcal{H}^{\mathbf{SHA\text{-}3}}(sk_i)$$

To be able to spend the input destined for him/her, the receiver has derive a one-time secret key with the following formula:

$$sk = \mathcal{H}^{\mathbf{SHA\text{-}3}}(R \cdot vk^s) \cdot sk_i$$

3.3.2 View Address

$$viewaddress_i = pk_i + vk_i^s$$

The receiver (or a third-party auditor) can utilize this key to scan the network for any incoming transaction without the ability to spend them using the following formula:

$$pk' = (\mathcal{H}^{\mathbf{SHA\text{-}3}}(R \cdot vk_i^p) \cdot G) + pk_i$$

3.3.3 Send Address

$$sendaddress_i = pk_i + vk_i^p$$

The sender can utilize the Key Agreement Scheme (outlined in **Section 4.5**) with the Send Address to compute a one-time public address using the following formula:

$$pk = (\mathcal{H}^{\text{SHA-3}}(r \cdot vk_i^p) \cdot G) + pk_i$$

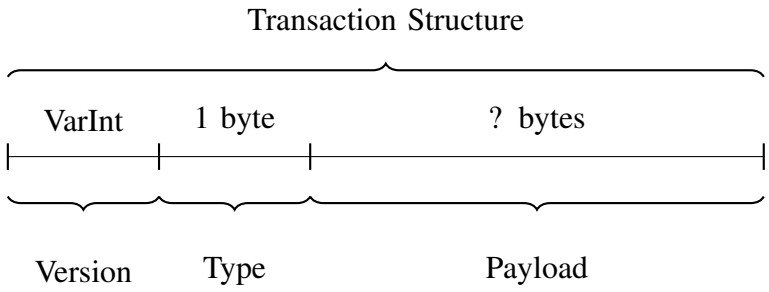
3.4 Transaction

Dusk Network utilizes a UTXO model. A UTXO model has no account balances, instead having inputs associated with an address, with the sum of the inputs equating to the balance of the account.

The input can be only spent in full, meaning that if *Alice* wants to send *Bob* 5 DUSK and her smallest input is equal to 10 DUSK, she has to create 2 outputs:

1. Sending 5 DUSK to *Bob*
2. Sending 5 DUSK to her change account

Dusk Network transactions have the following structure:



3.5 Transaction Types

Transaction Type	Opcode
Coinbase	`0x00`
Bid	`0x01`
Stake	`0x02`
Standard	`0x03`
Time-locked	`0x04`
Contract	`0x05`

Table 3: Transaction Types

3.5.1 Coinbase

Unlike a typical Coinbase transaction structure utilized in other platforms, Dusk Network Coinbase transaction has a more complex structure. The Coinbase transaction plays an essential role in block verification in Dusk Network.

The Coinbase transaction includes a zero-knowledge proof of inclusion in the current bidder set (which enables the bidder to obfuscate his/her identity)[outlined in **Section 4.10.1**], zero-knowledge proof of the score calculation (which enables the bidder to obfuscate his/her bid size) [outlined in **Section 4.10.1**], zero-knowledge proof of ephemeral key generation (which enables the bidder to generate a seed while obfuscating his/her identity)[outlined in **Section 4.10.1**], ephemeral secret key (outlined in **Section 4.1.1**), list of rewards for block B^{r-1} corresponding to the `brPublicKeys` set in $CERT^{r-1}$ and the stealth address of the bidder:

Field	Size	Description
`R`	32 bytes	The transaction ID.
`proof`	?	The range, set inclusion and ephemeral key validity proofs.
`ephemeralKey`	32 bytes	The ephemeral secret key.
`rewards`	?	Provisioner rewards for the previous block.
`generatorAddress`	32 bytes	The one-time public address of the Generator.

Table 4: Coinbase transaction structure

3.5.2 Bid

The Bid transaction is required for a user to become eligible to take a role of a Block Generator. The Bid transaction has to include a cryptographic digest of secret ($M = \mathcal{H}(K)$), with K doubling as a master key for the ephemeral key generation (outlined in **Section 4.1.1**).

The Bid transaction has the following structure:

Field	Size	Description
`R`	32 bytes	The transaction ID.
`count_in`	varInt	The input count.
`inputs`	?	The input set.
`count_out`	varInt	The output count.
`outputs`	?	The output set.
`time_lock`	8 bytes	The time-lock.
`secret`	32 bytes	The cryptographic digest of a secret K.
`fee`	8 bytes	The miner's fee.

Table 5: Bid transaction structure

3.5.3 Stake

The Stake transaction is required for a user to become eligible to take a role of a Provisioner. The Stake transaction has to deanonymize the total input amount (will be oulined in the upcoming SBA★ paper).

The Stake transaction has the following structure:

Field	Size	Description
`R`	32 bytes	The transaction ID.
`count_in`	varInt	The input count.
`inputs`	?	The input set.
`output`	?	The output with a deanonymized output amount.
`time_lock`	8 bytes	The time-lock.
`fee`	8 bytes	The miner's fee.

Table 6: Stake transaction structure

3.5.4 Standard

The Standard transaction is the generic transaction type. The Standard transaction has to contain at least 1 `input` and 1 `output`.

The Standard transaction has the following structure:

Field	Size	Description
`R`	32 bytes	The transaction ID.
`count_in`	varInt	The input count.
`inputs`	?	The input set.
`count_out`	varInt	The output count.
`outputs`	?	The output set.
`fee`	8 bytes	The miner's fee.

Table 7: Standard transaction structure

3.5.5 Time-locked

The Time-locked transaction is a similar type to the Standard transaction with the only difference being the addition of the `time-lock` field. The Time-locked transaction has to contain at least 1 `input` and 1 `output`. If `time-lock` $\geq$ `0x8000000000000000`, then the lock is set in block height (r), else the lock is set in the *Unix* timestamp.

The Time-locked transaction has the following structure:

Field	Size	Description
R	32 bytes	The transaction ID.
count_in	varInt	The input count.
inputs	?	The input set.
count_out	varInt	The output count.
outputs	?	The output set.
time_lock	8 bytes	The time-lock.
fee	8 bytes	The miner's fee.

Table 8: Time-locked transaction structure

3.5.6 Contract

The Contract transaction is the transaction type for smart contract interaction. The type will be defined more thoroughly in the next iterations of this document.

The Contract transaction has the following structure:

Field	Size	Description
R	32 bytes	The transaction ID.
count_in	varInt	The input count.
inputs	?	The input set.
count_out	varInt	The output count.
outputs	?	The output set.
time_lock	8 bytes	The time-lock.
fee	8 bytes	The miner's fee.

Table 9: Contract transaction structure

4 Cryptography

4.1 Pseudo-Random Functions

4.1.1 Common Reference String

The `seed` (mentioned in **Section 3.2.1** and outlined in **Section 5.7**) is a Common Reference String (CRS) is an integral part of the consensus protocol, responsible for unbiased, pseudo-random Block Generator, Signature Set Generator and Provisioner committee extractions (outlined in **5.8**).

The CRS of the block is a BLS signature of the CRS from the previous signature - $\Sigma^{BLS}(S^{r-1})$. The BLS signature is generated by the winning Block Generator using his/her ephemeral secret key and due to the deterministic nature of the BLS signatures, can be treated as a Random Oracle.

4.2 Hash Functions

Hash function have to possess the following three properties to be deemed cryptographically secure:

1. Pre-image Resistant
2. Second Pre-image Resistant
3. Collision Resistant

4.2.1 SHA-3

SHA-3 stands for **Secure Hash Algorithm 3** and is an adapted version of Keccak [Kee05], a sponge construction-based hash function. SHA-3 is parametrized by two values: the rate r and the capacity c. The 256-bit SHA-3 has a 128-bit security level with r set at 1088 and c set at 512. SHA-3 256 has l set at 6.

As always, the input has to be padded before being repeatedly run through a block cipher.

After padding the input, the sponge function absorbs the padded input and repeatedly runs through a block cipher construction featuring 5 transformations:

1. θ : Apply the transform for all x, y, z:

$$A[x, y, z] \leftarrow A[x, y, z] \oplus \sum_{y'=0}^{4} A[x-1, y', z] \oplus \sum_{y'=0}^{4} A[x=1, y', z-1].$$

2. ρ : For a given $(x, y) \in \mathbb{F}_5^2$, define $t \in \{0, ..., 23\}$ by the equation:

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \ (mod 5),$$

with $t = -1$ if $x = y = 0$. Then for all x, y, z apply the transform:

$$A[x, y, z] \leftarrow A[x, y, (z - (t + 1) \cdot (t + 2)/2) \ (mod 2^l)].$$

3. π : For a given $(x, y) \in \mathbb{F}_5^2$, define x', y' by the equation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \ (mod 5),$$

then for all x, y, z apply the transform:

$$A[x', y', z] \leftarrow A[x, y, z].$$

4. χ : For a given (x, y, z) apply the transform:

$$A[x, y, z] \leftarrow A[x, y, z] \oplus (A[x + 1, y, z] + 1) \cdot A[x + 2, y, z].$$

5. ι : For round number $i \in \{0, ..., 12 + 2 \cdot l - 1\}$ we define a round constant $\mathbf{rc}_i$. In round i the round constant $\mathbf{rc}_i$ us added to the (0,0)-lane. The round constants, assuming the standard of 24 rounds, are:

i	$\mathbf{rc}_i$
0	0x0000000000000001
1	0x0000000000008082
2	0x800000000000808A
3	0x8000000080008000
4	0x000000000000808B
5	0x0000000080000001
6	0x8000000080008081
7	0x8000000000008009
8	0x000000000000008A
9	0x0000000000000088
10	0x0000000080008009
11	0x000000008000000A
12	0x000000008000808B
13	0x800000000000008B
14	0x8000000000008089
15	0x8000000000008003
16	0x8000000000008002
17	0x8000000000000080
18	0x000000000000800A
19	0x800000008000000A
20	0x8000000080008081
21	0x8000000000008080
22	0x0000000080000001
23	0x8000000080008008

Table 10: ι round constants $\mathbf{rc}_i$

The block cipher function is executed in the following order:

Algorithm 1 SHA-3 256 function

for $i = 0$ **to** 22 **do**
$A \leftarrow \theta(A)$
$A \leftarrow \rho(A)$
$A \leftarrow \pi(A)$
$A \leftarrow \chi(A)$
$A \leftarrow \iota(A, i)$

4.2.2 Longsight

Longsight is an adaptation of MiMCHash, a circuit-friendly hash function proposed in [Tie16]. Specifically, Dusk Network utilizes a modification of LongsightF322p3 - LongsightF322p5, which utilizes an exponent of 5 instead of 3.

$f(x) \leftarrow x^e$ is a permutation when the exponent e and $p - 1$, where p is the prime order of a finite field ($\mathbb{F}_p$), are co-prime, meaning that:

$$gcd(e, p - 1) = 1$$

$p - 1$ in a Ristretto finite field ($\mathbb{F}_p$, where $p = (2^{255} - 19)/8$) is not co-prime to $e = 3$, so $e = 5$ is used instead.

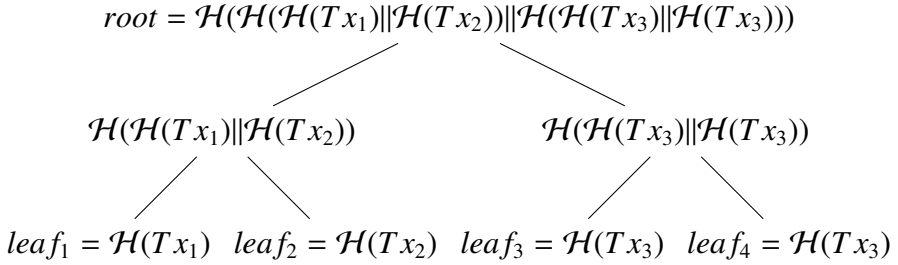
LongsightF322p5 iterates through 322 round of Feistel block cipher, which has the following structure:

$$x_L||x_R \leftarrow x_R \oplus (x_L \oplus c_i)^5||x_L$$

4.3 Merkle Tree

Merkle tree is a tree-like cryptographic construction where the leafs represent the cryptographic digests of the data entries and the parent nodes are the cryptographic digests of their children nodes.

A simple Merkle tree constructed from a set of three transactions (Tx_1, Tx_2 and Tx_3) has the following structure:



4.4 Elliptic Curves

4.4.1 Ristretto

Ristretto is an extension of [Ham15] for cofactor-8 elliptic curve groups. Based on Curve25519, Ristretto forms a thin abstraction layer constructing a prime-order group from the non-prime order curve (Curve25519).

4.4.2 BN-256

BN-256 is a Barreto-Naehrig curve [Sch10] where $E : y^2 = x^3 + b$ is a BN curve over a prime field $\mathbb{F}_p$ with an optimal ate pairing algorithm. There is a $u \in \mathbb{Z}$ such that both p and n, given by:

$$p = p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$$
$$n = n(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$$

are prime ($u = v^3, v = 1868033$).

E has an embedding degree $k = 12$ and the number of $\mathbb{F}_p$-rational points on E is $\#E(\mathbb{F}_p) = n$. O is denoted as a point of infinity.

An optimal ate pairing on E is:

$$a_{opt} : G_2 \times G_1 \to G_3, (Q, P) \longmapsto (f_{6u+2,Q}(P) \cdot g_{6u+2,Q}(P))^{(p^12-1)/n}$$

where $g_{6u+2,Q}(P) = l_{Q_3,-Q_2}(P) \cdot l_{-Q_2+Q_3,Q^1}(P) \cdot l_{Q_1-Q_2+Q3,[6u+2]Q}(P)$ with $Q_1 = \phi_p(Q)$, $Q_2 = \phi_p^2(Q)$ and $Q_3 = \phi_p^3(Q)$. The value $l_{S,R}$

4.5 Key Agreement Scheme

Borrowing the idea from [Hel76], [Sab13] presents a Diffie-Hellman Key Exchange adaptation for a non-interactive one-time public key generation scheme. The scheme functions in the following way:

1. Generate a random scalar $r \in \mathbb{F}_p^*$, where p is the order of the Ristretto curve.
2. Compute the corresponding public key $R = r \cdot G$.
3. Compute the shared secret $r \cdot vk_B^p$. The receiver can calculate the shared secret with the corresponding secret key vk_B^s by computing $R \cdot vk_B^s$ which is equivalent to $r \cdot (vk_B^s \cdot G) = r \cdot vk_B^p$.
4. Compute the one-time public key $pk = (\mathcal{H}^{\textbf{SHA-3}}(r \cdot vk_B^p) \cdot G) + pk_B$.
5. Include R and pk in the transaction.

The receiver is able to determine a transaction destined for him/her using his/her view address (outlined in **Section 3.X.X**) to scan the network for the incoming transactions.

The receiver can verify that the one-time public key has been generated for him/her by computing:

$$pk' = (\mathcal{H}^{\textbf{SHA-3}}(R \cdot vk_B^s)) \cdot G) + pk_B.$$

If $pk = pk'$, then the transaction has been destined for him/her.

4.6 Symmetric Encryption Scheme

For the receiver to open the Pedersen's Commitment (outlined in **Section 4.9.1**), the sender has to include the value x and the blinder a in the transaction. However, including the transparent x and a would render the commitment ineffective, so the values have to be encrypted before being included in the transaction.

The encryption scheme utilized produces a ciphertext from exclusive OR (XOR; $\oplus$) of the plaintext and the key:

$$plaintext \oplus key \rightarrow ciphertext$$

To encrypt x and a, the sender has to XOR the value with the one-time public key pk (outlined in **Section 4.6**) - $x \oplus pk \rightarrow x_{pk}^{\mathbf{enc}}$ and $a \oplus pk \rightarrow a_{pk}^{\mathbf{enc}}$.

4.7 Signature Schemes

4.7.1 EdDSA

Presented in [Yan11], EdDSA is generalized version of Ed25519 digital signature scheme. Originally based on a twisted Edwards birational equivalent of Curve25519 (ed25519), Dusk Network utilizes a generalized version on a Ristretto curve (outlined in **Section 4.4.1**).

The scheme includes two functions:

1. A signing function - EdDSA.$\mathcal{S}$()
2. A verifying function - EdDSA.$\mathcal{V}$()

The signing function $\mathcal{S}$ takes the message x, random number $r \in \mathbb{F}_p$ and the secret spend key sk_i as an input and outputs a 64-byte tuple - (R, S). The output is generated via the following procedure:

$$\mathcal{S}(x, r, sk_i) : R \leftarrow r \cdot G;\ S \leftarrow (r + \mathcal{H}(R, pk_i, x) \cdot sk_i) \mod p$$

The verifying function $\mathcal{V}$ takes the message x and a tuple (R, S) as an input and outputs a boolean value b. The output is generated via the following procedure:

$$\mathcal{V}(x, (R, S)) : b \leftarrow S \cdot G = r \cdot G + \mathcal{H}(R, pk_i, x) \cdot pk_i$$

The signature is deemed to be valid if the verifying function $\mathcal{V}$ outputs $b = 1$.

4.7.2 BLS

Proposed in [Sha01], Boneh-Lynn-Shacham signatures utilize elliptic curve pairings to produce short, deterministic signatures.

4.8 Ring Signature Scheme

Ring singature schemes have to possess the following three properties to be deemed cryptographically secure:

1. Signer Ambiguity.
2. Linkability.
3. Unforgeability.

4.8.1 MLSAG

MLSAG [Noe15] stands for Multilayered Linkable Spontaneous Anonymous Group and is a generalization of the LSAG signature scheme, originally presented in [Won04], applicable when we have a set of $n \cdot m$ keys:

$$\mathcal{R} = \{pk_{i,j}\} \text{ for } i \in \{1, 2, ..., n\} \text{ and } j \in \{1, 2, ..., m\}$$

The scheme includes two functions:

1. A signing function - MLSAG.$\mathcal{S}$()
2. A verifying function - MLSAG.$\mathcal{V}$()

The signing function $\mathcal{S}$ takes set $\mathcal{R}$, message x, a set of random numbers $\mathcal{A} = \{\alpha_1, ..., \alpha_m\}$, where $\alpha_j \in \mathbb{F}_p$, and a set of random numbers $P = \{r_{1,1}, ..., r_{n,m}\}$ (except for $i = \pi$), where $r_{i,j} \in \mathbb{F}_p$ as an input and outputs $\sigma(x)$ alongside a set of key images $\mathcal{K} = \{\tilde{K}_1, ..., \tilde{K}_m\}$. The output is generated via the following procedure:

1. Compute key images $\tilde{K}_j = k_{\pi,j} \cdot \mathcal{H}(pk_{\pi,j})$ for all $j \in \{1, 2, ..., m\}$.
2. Compute $c_{\pi+1} = \mathcal{H}(x, [\alpha_1 \cdot G], [\alpha_1 \cdot \mathcal{H}(pk_{\pi,1})], ..., [\alpha_m \cdot G], [\alpha_m \cdot \mathcal{H}(pk_{\pi,m})])$.
3. For $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ compute, replacing $n + 1 \to 1$,
   $c_{i+1} = \mathcal{H}(x, [r_{i,1} \cdot G + c_i \cdot pk_{i,1}], [r_{i,1} \cdot \mathcal{H}(pk_{i,1}) + c_i \cdot \tilde{K}_1], ..., [r_{i,m} \cdot G + c_i \cdot pk_{i,m}], [r_{i,m} \cdot \mathcal{H}(pk_{i,m}) + c_i \cdot \tilde{K}_m])$.
4. Define $r_{\pi,j} = \alpha_j - c_\pi \cdot k_{\pi,j} \mod p$.
5. Output the signature $\sigma(x) = (c_1, r_{1,1}, ..., r1, m, ..., rn, 1, ..., rn, m)$ and the key image set $\mathcal{K}$.

The veryfing function $\mathcal{V}$ takes the message x, the signature $\sigma(x)$ and the key image set $\mathcal{K}$ and outputs a boolean value b. The output is generated via the following procedure:

1. For all $j \in \{1, ..., m\}$ check $p \cdot \tilde{K}_j \stackrel{?}{=} 0$.
2. For $i = 1, ..., n$ compute, replacing $n + 1 \to 1$,
   $c'_{i+1} = \mathcal{H}(x, [r_{i,1} \cdot G + c_i \cdot pk_{i,1}], [r_{i,1} \cdot \mathcal{H}(pk_{i,1}) + c_i \cdot \tilde{K}_1], ..., [r_{i,m} \cdot G + c_i \cdot pk_{i,m}], [r_{i,m} \cdot \mathcal{H}(pk_{i,m}) + c_i \cdot \tilde{K}_m])$.
3. $c'_1 = c_1$, then output $b = 1$.

4.9 Commitment Scheme

A basic definition of a commitment is function that is hiding and binding, meaning that the function hides the original value while making it computationally infeasible for the user to find another value that produces the same output. Taking the outlined logic into account, a cryptographic digest ($\mathcal{H}(x)$) of the value might seem to be an ideal commitment scheme. However, for the required application, cryptographic digests cannot produce a valid commitment for two reasons:

1. Small digit commitments are vulnerable to binary search attacks.
2. Hash functions are not homomorphic, meaning $\mathcal{H}(a) + \mathcal{H}(b) \neq \mathcal{H}(a + b)$

While **(1)** can be solved by appending a random integer $r \in \mathbb{F}_p$, called a **blinder**, to the value x before hashing - $\mathcal{H}(r\|x)$, **(2)** cannot be solved without switching to a different scheme.

4.9.1 Pedersen's Commitment

Pedersen's Commitment scheme is a partially homomorphic commitment scheme, meaning the scheme possesses an additive homomorphic property, but not the multiplicative. However, for the required application the additive homomorphic property will suffice.

The commitment scheme requires two generators - G and H, where the discrete logarithm between the two is unknown - $H \stackrel{?}{=} \iota \cdot G$. To achieve the aforemention property, G is hashed to generate a nothing-up-my-sleeve number, which is then mapped on a curve E:

$$H \leftarrow Hash\_to\_Point(\mathcal{H}(G)).$$

The relationship ι between the points G and H could only be determined by breaking the Discrete Logarithm Problem of the underlying group.

Pedersen Commitment has the following structure:

$$C(x) = r \cdot G + x \cdot H$$

4.9.2 Vector Pedersen's Commitment

The Pedersen's Commitment scheme can be extended to vectors, where vector $\mathbf{v} = \{v_1, ..., v_n\}$ is committed instead of x. The aforementioned procedure is formalized below:

$$C(\mathbf{v}) = r \cdot G + (v_1 \cdot H_1, ..., v_n \cdot H_n) = r \cdot G + \mathbf{v} \cdot \mathbf{H}$$

with $\mathbf{H} = \{H_1, ..., H_n\}$ generated through a procedure outlined in **Section 4.9.1**:

$$H_i \leftarrow Hash\_to\_Point(\mathcal{H}(G\|i)) \text{ for all } i \in \{1, ..., n\}.$$

4.10 Zero-Knowledge Proof Protocol

4.10.1 Bulletproofs

Bulletproofs belong to a family of general zero-knowledge proof of knowledge protocols capable of generating a proof for any statement belonging to a $\mathcal{NP}$ language $\mathcal{L}$. $\mathcal{NP}$ is a group of non-deterministic polynomial time decision problems which have verifiable proof in polynomial time.

Proposed by [Max17]

4.10.2 Range Proof

Range proof is a statement π proving that value $v \in [0, 2^n)$, without revealing any additional information about the value.

To be prove the correctness of the state, the value has to satisfy the following 3 conditions:

1. $\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$
2. $(\mathbf{a}_L - \mathbf{1}) - \mathbf{a}_R = \mathbf{0}$
3. $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{0}$

The statements above have to be combined in one statement. The two vector-statements need to be combined into one statement. For this, challenge y is generated using the Fiat-Shamir heuristic:

$$\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$$

$$\langle (\mathbf{a}_L - 1) - \mathbf{a}_R, \mathbf{y}^n \rangle = 0$$

$$\langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{y}^n \rangle = 0$$

The three statements can be combined together, with z generated using the Fiat-Shamir heuristic:

$$z^2 \cdot v = z^2 \cdot \langle \mathbf{a}_L, \mathbf{2}^n \rangle + z \cdot \langle (\mathbf{a}_L - 1) - \mathbf{a}_R, \mathbf{y}^n \rangle + \langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{y}^n \rangle$$

The statement can be rearranged, to combine the terms into a single inner product:

$$z^2 \cdot v + z \cdot \langle \mathbf{1}, \mathbf{y}^n \rangle = \langle \mathbf{a}_L, z^2 \cdot \mathbf{2}^n + z \cdot \mathbf{y}^n + \mathbf{a}_R \circ \mathbf{y}^n \rangle + \langle -z \cdot \mathbf{1}, \mathbf{a}_R \circ \mathbf{y}^n \rangle$$

An addition of $\langle -z \cdot \mathbf{1}, z^2 \cdot \mathbf{2}^n + z \cdot \mathbf{y}^n \rangle$ and further simplification renders the following statement:

$$z^2 \cdot v + (z - z^2) \cdot \langle \mathbf{1}, \mathbf{y}^n \rangle - z^3 \cdot \langle \mathbf{1}, \mathbf{2}^n \rangle = \langle \mathbf{a}_L - z \cdot \mathbf{1}, z^2 \cdot \mathbf{2}^n + z \cdot \mathbf{y}^n + \mathbf{a}_R \circ \mathbf{y}^n \rangle$$

Combining and factoring out all the non-secret terms into a new term δ:

$$\delta \cdot (y, z) = (z - z^2) \cdot \langle \mathbf{1}, \mathbf{y}^n \rangle - z^3 \cdot \langle \mathbf{1}, \mathbf{2}^n \rangle$$

The following statement is obtained:

$$z^2 \cdot v + \delta \cdot (y, z) = \langle \mathbf{a}_L - z \cdot \mathbf{1}, \mathbf{y}^n \circ (\mathbf{a}_R + z \cdot \mathbf{1}) + z^2 \cdot \mathbf{2}^n \rangle$$

Splitting the inner product into unblinded $\mathbf{l}(x)$ and unblinded $\mathbf{r}(x)$, the following 3 statements are obtained:

1. unblinded $\mathbf{l}(x) = \mathbf{a}_L - z \cdot \mathbf{1}$
2. unblinded $\mathbf{r}(x) = \mathbf{y}^n \circ (\mathbf{a}_R + z \cdot \mathbf{1}) + z^2 \cdot \mathbf{2}^n$
3. $z^2 \cdot v + \delta \cdot (y, z) = \langle$unblinded $\mathbf{l}(x)$,unblinded $\mathbf{r}(x) \rangle$

The prover cannot send the two unblinded vectors to the verifier without leaking the information about the value v.

Instead, the prover selects two blinding vectors:

$$\mathbf{s}_L, \mathbf{s}_R \xleftarrow{R} \mathbb{Z}_p^n,$$

using them to contstruct vector polynomials:

1. $\mathbf{l}(x) = \mathbf{l}_0 + \mathbf{l}_1 \cdot x = (\mathbf{a}_L + \mathbf{s}_L \cdot x) - z \cdot \mathbf{1}$
2. $\mathbf{r}(x) = \mathbf{r}_0 + \mathbf{r}_1 \cdot x = \mathbf{y}^n \circ (\mathbf{a}_R + \mathbf{s}_R \cdot x) + z \cdot \mathbf{1}) + z^2 \cdot \mathbf{2}^n$

Deducing the following statement:

$$\langle \mathbf{l}_0, \mathbf{r}_0 \rangle = z^2 \cdot v + \delta \cdot (y, z)$$

and setting the following:

$$t(x) = \langle \mathbf{l}(x), \mathbf{r}(x) \rangle = t_0 + t_1 \cdot x + t_2 \cdot x^2,$$

the coefficients of $t(x)$ can be expressed using Karatsuba's method:

$$t_0 = \langle \mathbf{l}_0, \mathbf{r}_0 \rangle$$

$$t_2 = \langle \mathbf{l}_1, \mathbf{r}_1 \rangle$$

$$t_1 = \langle \mathbf{l}_0 + \mathbf{l}_1, \mathbf{r}_0 + \mathbf{r}_1 \rangle - t_0 - t_2$$

4.10.3 Set Inclusion Proof

Set inclusion proof is a statement ρ proving that value $w \in S$, where S is a set of values.

The algorithm for the set inclusion proof is currently in the works and will be included in the upcoming iterations of the document.

5 Consensus

5.1 General Overview

SBA★, which stands for Segregated Byzantine Agreement, is a hybrid Proof-of-Stake protocol with statistical finality guarantees. The protocol is based on an Honest Majority of Money (an Adversary can corrupt nodes controlling up to f percent of the total stake value [$\geq 3f + 1$]) and weak network synchrony assumptions.

The roles in the protocol are split between two different node types: Block Generators (outlined in **Section 5.2**) and Provisioners (outlined in **Section 5.3**). Block Generator retain their privacy, with the proofs computed in zero-knowledge to preserve the anonymity of the Block Generator. On the other hand, Provisioners are required to deanonymize their stakes and remain transparent about their activities in the consensus while their stake remains valid.

Both bids and stakes have a registration period of t, which begins when a Bid or Stake transaction is included in a final block and is required to elapse before the full-node is eligible to participate in the consensus.

5.2 Block Generator

Block Generator is the first of the two full-node types eligible to participate in the consensus. To become a Block Generator, a full-node has to submit a Bid transaction (outlined in **Section 3.5.2**).

The Block Generator is eligible to participate in one phase - Block Generation phase. In the aforementioned phase, Block Generators participate in a non-interactive lottery (outlined in **Section 5.8**) to be able to forge a block candidate, with the incentive being the block reward.

5.3 Provisioner

Provisioner is the second of the two full-node types eligible to participate in the consensus. To become a Block Generator, a full-node has to submit a Stake transaction (outlined in **Section 3.5.3**). Unlike a Block Generator, a Provisioner node is required to deanonymize the value of the stake to be able to participate in the consensus. While it is technically possible to obfuscate the stake value, the team has decided against the latter as the addition of stake value obfuscation would have slowed down the consensus and simultaneously increased the block size.

The Provisioner is eligible to participate in five phases - Block Reduction, Block Set Agreement, Signature Set Generation, Signature Set Reduction and Signature Set Agreement phases.

5.4 Binary Reducation

The Binary Reduction algorithm is an adaptation of [Coa84], with two variations of the algorithm forming the core of SBA★. Unlike Algorand [Mic17], which utilizes the original algorithm as an extenstion of BBA★ [Mic18].

Binary Reduction is a two-step algorithm, with the input of the second step depending on the output of the first one. If no consensus have been reached on a uniform value, the algorithm return value and waits for the next instantiation.

Algorithm 2 Binary Reduction

```
procedure BinaryReduction(ctx, user, value)
    CommitteeVote(ctx, user,REDUCTION_ONE, value)

    value_1 ← CountVotes(ctx, round,REDUCTION_ONE)
    fallback_value ← init_value

    if value_1 = TIMEOUT then
        CommitteeVote(ctx, user,REDUCTION_TWO, value)

    else
        CommitteeVote(ctx, user,REDUCTION_TWOvalue_1)

    value_2 ← CountVotes(ctx, round,REDUCTION_TWO)

    if value_2 = TIMEOUT then
        return fallback_value

    else
        return value_2
```

5.5 Set Agreement

During the conduction of a technical analysis on [Mic17], the team has discovered a vulnerability, which increases the probability of a the consensus forking, called a ”timeout fork”. As a result, the team has concluded that SBA★ requires a modification to guarantee the statistical finality under the basic assumptions of the protocol.

Set Agreement is an asynchronous algorithm running in parallel with the main loop. The protocol includes two variation of the Set Agreement algorithm - Block Set Agreement and Signature Set Agreement. Successful termination of one of the variations of the algorithm indicates that the relevant section of the main loop has been successfully executed. The algorithm provides a statistical guarantee that at least one honest node has received a set of votes exceeding the minimum threshold required to successfully terminate the respective phase of the protocol.

Algorithm 3 Set Agreement

```
procedure SetAgreement(ctx, user)
    sets ← {}

    msgs ← incomingMsgs[round].iterator()

    while len(sets[i]) < ctx.T_STEP · ctx.τ_STEP do
        m ← msgs.next()

        if m = ⊥ then continue

        else
            ⟨set, step⟩ ← ProcessSetMsg(ctx, τ, m)

            if set ∉ sets[step] then
                sets[step].include(set)

    return sets[i]
```

5.6 Blind Bid

The Blind Bid scheme enables the Block Generator to provably preserve his/her anonymity while participating in the consensus.

The proof of the bid alongside the proof of the score correctness is included in every Coinbase transaction (outlined in **Section 3.5.1**) of a candidate block. The proof does not leak information about the bidder or the size of the bid. The Blind Bid proves the validity of the bid (through the Merkle opening proof) and the correctness of the score (through the bid size proof).

5.6.1 Notation

Let E be the Bulletproof curve with prime order r:

Variables:

- Seed S – 32-byte string
- Secret K – 32-byte string
- Transaction hash X – 32-byte string
- Secret hash M – 32-byte string
- Merkle tree $\mathcal{T}$ of bids with root $\mathcal{T}^{\mathcal{R}}$
- Coin amount d – integer between 0 and 2^{64}
- Counter N – 32-byte string

Functions:

- $\mathcal{H}^{\mathbf{MiMC}}$ – Longsight hash, a Bulletproof-friendly hash function
- $\mathcal{H}^{\mathbf{MiMC}}(X, O)$ Merkle root construction function. It assumes that O is a Merkle opening for X in a tree built using $\mathcal{H}$, and outputs the tree root corresponding to the opening
- $F_B(d, Y)$ – score function. Takes 64-bit input d and 256-bit input Y and operates as follows:

  - Truncate Y to left 128 bits and interpret the result as 128-bit integer Y'.
  - Output $f = d/Y' \mod r$, where r is the prime order of the Bulletproof curve.

5.6.2 Proof

Let C be the following computation:

- **Public Inputs**: S
- **Auxiliary Inputs**: K, d, N
- **Flow**:
  1. $M = \mathcal{H}^{\mathbf{MiMC}}(K)$
  2. $Z = \mathcal{H}^{\mathbf{MiMC}}(S, M)$
  3. $X = \mathcal{H}^{\mathbf{MiMC}}(d, M, N)$
  4. $\mathcal{H}^{\mathbf{MiMC}}(X, \mathcal{O}) = \mathcal{T}^{\mathcal{R}}$
  5. $Y = \mathcal{H}^{\mathbf{MiMC}}(S, X)$
  6. $Q = F_B(d, Y)$

  Public Output: Z, R, Q

Then Π is the Bulletproof proof of computational integrity of C.

5.6.3 Protocol

Procedure:

1. A universal counter N is maintained for all bidding transactions in the lifetime.
2. Seed S is computed and broadcasted.
3. Bidder selects secret K.
4. Bidder sends a Bid transaction with data $M = \mathcal{H}^{\mathbf{MiMC}}(K)$.
5. For every bidding transaction with d coins and data M an entry $X = \mathcal{H}^{\mathbf{MiMC}}(d, M, N)$ is added to $\mathcal{T}$. Then N is increased.
6. Potential bidder computes $Y = \mathcal{H}^{\mathbf{MiMC}}(S, X)$, score $Q = F_B(d, Y)$, and identier $Z = \mathcal{H}^{\mathbf{MiMC}}(S, M)$.
7. Bidder selects a bid root $\mathcal{T}^{\mathcal{R}}$ and broadcasts proof:

   $\pi = \Pi(Z, \mathcal{T}^{\mathcal{R}}, Q, S\,; K, d, N)$.

5.7 Seed

5.8 Ephemeral Secret Key

The ephemeral secret key generation scheme is designed to generate a zero-knowledge proof of an ephemeral secret key derivation procedure, proving that the secret was derived correctly from the pre-image of M, included in the relevant Bid transaction.

Ephemeral secret keys are utilized in seed generation (outlined in **Section 5.7**) to guarantee the pseudo-randomness of the seed, which is essential to the security assumptions of the protocol.

5.8.1 Notation

Variables:

- Round r – 8-byte integer
- Seed S - 32-byte string
- Master secret key K – 32-byte string
- Transaction hash X – 32-byte string
- Child secret key K^r – 32-byte string
- Secret hash M – 32-byte string

Functions:

- $\mathcal{H}^{\mathbf{MiMC}}$ – Longsight hash, a Bulletproof-friendly hash function
- Signature $\mathcal{S}^{\mathbf{BLS}}(a)$ - A signature of message a.

5.8.2 Protocol

Let C be the following computation:

- **Public Inputs**: S, M
- **Auxiliary Inputs**: K
- **Flow**:
  1. $M = \mathcal{H}^{\mathbf{MiMC}}(K)$
  2. $Z = \mathcal{H}^{\mathbf{MiMC}}(S, M)$
  3. $K^r = \mathcal{H}^{\mathbf{MiMC}}(K||S^{r-1}||r)$

  Public Output: Z, K^r

Then Π is the Bulletproof proof of computational integrity of C.

5.8.3 Proof

The proof section of the ephemeral secret key generation scheme will be included in the upcoming iterations of the document.

5.9 Sortition

Sortition is a pseudo-random lottery, which produces an output dependant solely on the seed S, private key $pk^{\mathbf{BLS}}$ and the weight of the stake τ. [Mic17] proposed using a Verifiable Random Function (VRF) to compute the participant's score, however, the solution was not space-efficient, requiring 96 bytes of storage per score-proof pair.

After a lengthy consideration, the Dusk Network team has concluded that the inefficiency of the former can be solved with the use of BLS (outlined in **Section 4.7**) signatures. Being deterministic, BLS signatures require no random inputs and no proofs as a consequence, resulting in a compact score consuming only 32 bytes of storage.

SBA★ utilizes two variations of the Sortition algorithm - Generator Sortition and Committee Sortition algorithms. The former outputs the largest score produced by the distribution fucntion:

Algorithm 4 Generator Sortition

procedure GENERATORSORTITION($seed, sk, \tau, role, \omega, W$)

$p \leftarrow \frac{\tau}{W}$
$j \leftarrow 0$

$score \leftarrow Sig_{BLS}^{user.BLS.sk}(seed||role)$

while $\frac{hash}{2^{len(hash)}} \notin [\sum_{k=0}^{j} F_P(k, \omega, p), \sum_{k=0}^{j+1} F_P(k, \omega, p)]$ **do**
j++

if $j > 0$ **then**
return $score$

while the latter outputs j, the counter of the distribution function outputs exceeding the threshold for the respective phase:

Algorithm 5 Committee Sortition

procedure COMMITTEESORTITION($seed, sk, \tau, role, \omega, W$)

$p \leftarrow \frac{\tau}{W}$
$j \leftarrow 0$

$score \leftarrow Sig_{BLS}^{user.BLS.sk}(seed||role)$

while $\frac{hash}{2^{len(hash)}} \notin [\sum_{k=0}^{j} F_P(k, \omega, p), \sum_{k=0}^{j+1} F_P(k, \omega, p)]$ **do**
j++

return $\langle score, j \rangle$

5.10 Guru

Proposed in [Ale17], Guru is a reputation module for permissioned protocols. The module enables the permissioned protocols to function in a network of untrusted validator and adapt to the permission-less settings.

While the original proposal does not fit the purpose of SBA★, as the protocol is designed to securely function in permission-less settings. However, with a few minor modifications, Guru is well-suited to improve the security of the protocol. As SBA★ is already dependent on the lotteries to extract the Provisioner committees, Guru can be utilized in combination with the stakes to produce a more granular extraction scheme and effectively suppress malicious behaviour.

5.11 Adversary Model

Adversary $\mathcal{A}$ can simultaneously corrupt a maximum of $\mathbf{f}$ consensus participants. $\mathcal{A}$ is a mildly adaptive adversary, meaning that $\mathcal{A}$ chooses the nodes to corrupt k rounds before the actual corruption happens.

$\mathcal{A}$ is a computationally-bounded adversary, implying that $\mathcal{A}$ cannot find hash collisions or forge signatures beyond a negligible probability.

6 Future Additions

6.1 Deterministic Sortition

An improved sortition algorithm for Provisioner committee extractions enables the nodes to extract the committee from the Provisioner set with public values, meaning that no interaction and score propagation is no longer required from the Provisioners to prove their inclusion in a committee.

The implementation of deterministic sortition nearly halves the size Block Certificate (outlined in **Section 3.2.3**) while simultaneously decreasing the computational load on the full-nodes (pairing functions can be considered computationally-heavy tasks).

6.2 Cryptoeconomics

The team is currently researching the distribution functions applicable to the inverse proportions of the ”extraction probability vs reward” setting. Simultaneously, the investigation into reward emission distributions and novel reward models is ongoing.

6.3 Ring signatures with Bulletproofs

Our team is testing a novel ring signature approach with the use of Bulletproofs. The procedure requires a generation of a zero-knowledge proof of set inclusion in a set of inputs, without a need to generate the signatures.

If no attack vectors are discovered, this scheme could dramatically decrease the transaction size and there improve the throughput of the platform.

References

[Hel76] Whitfield Diffie Martin E. Hellman. *New Directions in Cryptography*. 1976. URL: https://ee.stanford.edu/~hellman/publications/24.pdf.

[Coa84] Russell Turpin Brian A. Coan. *Extending Byzantine Binary Agreement To Multivalued Byzantine Agreement*. 1984. URL: https://groups.csail.mit.edu/tds/papers/Coan/TurpinCoan-ipl84.pdf.

[Sha01] Dan Boneh Ben Lynn Hovav Shacham. *Short signatures from the Weil pairing*. 2001. URL: https://www.iacr.org/archive/asiacrypt2001/22480516.pdf.

[Won04] Joseph K. Liu Victor K. Wei Duncan S. Wong. *Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups*. 2004. URL: https://eprint.iacr.org/2004/027.pdf.

[Kee05] Guido Bertoni Joan Daemen Michaël Peeters Gilles Van Assche Ronny Van Keer. *Keccak implementation overview*. 2005. URL: https://keccak.team/files/Keccak-implementation-3.2.pdf.

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[Sch10] Michael Naehrig Ruben Niederhagen Peter Schwabe. *New software speed records for cryptographic pairings*. 2010. URL: https://cryptojedi.org/papers/dclxvi-20100714.pdf.

[Yan11] Daniel J. Bernstein Niels Duif Tanja Lange Peter Schwabe Bo-Yin Yang. *High-speed high-security signatures*. 2011. URL: https://ed25519.cr.yp.to/ed25519-20110926.pdf.

[Sab13] Nicolas van Saberhagen. *CryptoNote v 2.0*. 2013. URL: https://cryptonote.org/whitepaper.pdf.

[Ham15] Mike Hamburg. *Decaf: Eliminating cofactors through point compression*. 2015. URL: https://eprint.iacr.org/2015/673.pdf.

[Noe15] Shen Noether. *Ring Confidential Transactions*. 2015. URL: https://eprint.iacr.org/2015/1098.pdf.

[Tie16] Martin Albrecht Lorenzo Grassi Christian Rechberger Arnab Roy Tyge Tiessen. *MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity*. 2016. URL: https://eprint.iacr.org/2016/492.

[Ale17] Dmitry Khovratovich Alex Biryukov Daniel Feher. *Guru: Universal Reputation Module for Distributed Consensus Protocols*. 2017. URL: https://eprint.iacr.org/2017/671.pdf.

[Max17] Benedikt Bunz Jonathan Bootle Dan Boneh Andrew Poelstra Pieter Wuille Greg Maxwell. *Bulletproofs: Short Proofs for Confidential Transactions and More*. 2017. URL: https://eprint.iacr.org/2017/1066.pdf.

[Mic17] Jing Chen Silvio Micali. *Algorand*. 2017. URL: http://48d6hp28tnn92pdfye2fog4w-wpengine.netdna-ssl.com/wp-content/uploads/2018/10/Theoretical.pdf.

[Mic18] Silvio Micali. *Byzantine Agreement, Made Trivial*. 2018.

[Ven18] Emanuele Francioni Fulvio Venturelli. *The Dusk Network And Blockchain Architecture*. 2018. URL: https://github.com/dusk-network/whitepaper/releases/download/v0.3/dusk-whitepaper.pdf.