Yobicash: a cryptocurrency for secure sharing and storage of data

Christian Nyumbayire

Revised December 2, 2017

Abstract

Yobicash is a cryptocurrency built to secure the storage and sharing of data by ensuring the main properties of information security: confidentiality, integrity and availability. To enforce these properties, Yobicash employs cryptographic protocols, data replication, economic incentives and a dagchain of transactions.

Contents

1 Introduction

The World Wide Web is built on top of technologies for sharing, storing and retrieving data. A few decades after its inception, the web has become the backbone of the information economy, and thanks to innovations as the Internet of Things, Virtual Reality and Augmented Reality, it is going to become ever more pervasive.

Even if it is so critical for our civilization, the web is all but safe. Data breaches, communication disruptions, and data losses are becoming more common, less expensive to accomplish and more dangerous. Meanwhile, we are increasing our dependence to the web even for the most common operations.

To ensure safe data transmission and outsourcing, we need to solve the web's lack of information security, which means to enforce data confidentiality, integrity, and availability.

Yobicash is a peer-to-peer network for sharing and storing data safely, incentivized by a cryptocurrency. It is based on a dagchain, an authenticated directed acyclic graph of transactions, a more scalable alternative to the blockchain.

Yobicash takes inspiration from previous works such as Bitcoin [**?**] and Iota [1], improving on the scalability of blockchain-based cryptocurrencies, the security and decentralization of IOTA-like dagchains, the censorship-resistance of both, and adding incentives for on-chain sharing and storage of data.

2 Information Security

Information security [2] is given by three properties, called the CIA triad: confidentiality, integrity, and availability. It is important to understand what they are, why they matter and what are the disincentives to enforce them.

2.1 Confidentiality

Information is considered confidential when its owner can reveal its content only to those she or he desires. This property is pertinent for all sensitive data. Even data which may not seem informative at first glance can indirectly reveal some personal information about its owner when aggregated and analyzed.

Confidentiality ensures censorship-resistance, which is important in order to reduce the incentives of any malevolent agent that tries to breach, destroy or illegitimately alter the data or users it has chosen to target. In the lack of clues on the content of the data and the identity of its owner, attackers would have to limit themselves to disrupting random communication channels or to blocking

the operations of network nodes. But it would not be able to breach sensitive data.

The tools for data confidentiality are encryption, mixers and anonymous credential systems. Encryption ensures content privacy, mixers provide untraceable clients locations, and anonymous credential systems anonymize identities.

When there are proper measures to enforce confidentiality, an attacker has to access users credentials to be able to unveil data, identities or location.

By targeting single clients or node machines, attackers can gain access to credentials, or exploit communication channels and learn access patterns when the measures are not well designed or well implemented.

2.2 Integrity

The integrity of information is certified when no agent has altered it without the consent of its owner. This property is the more important, as it lacks put in danger data usability.

The error conditions of information integrity are data corruption and data loss. Data corruption can be identified through cryptographic authentication, prevented by data replication and recovered by forward error correction. Data loss is only prevented through data replication or recovered when it is possible to regenerate it.

An attacker can corrupt or drop stored and shared data by compromising single machines or through man-in-the-middle attacks against communication channels.

2.3 Availability

Information is considered available when its requester can always retrieve it, and it is more or less available according to the retrieval latency. The measure of latency depends on the needs of the requester, but under equal conditions, a network with lower latency is considered more available.

Most operations nowadays require data to be available at low latency. The lack would result in discoordination, financial loss, and sometimes worse.

Data unavailability is a temporary condition caused by computer or network failures. When permanent, unavailability become data loss and a failure in information integrity.

An attacker can make data unavailable in two ways: by dropping or rerouting packets, or through denial of service attacks directed toward network nodes.

2.4 Disincentives

The Internet, as every open network, is subject to network externalities. This means that products, service, technology on the Internet are interdependent. As shown by Ross Anderson [3], Information security is a common good on the Internet: nodes that invest in their safety increase the overall security of all the network. Conversely, nodes that undermine their security make the entire network more vulnerable.

As for other common goods, information security is affected by a tragedy of the commons: the marginal cost of increasing a node security is higher than its marginal benefits, and so there are no incentives to invest to make the necessary investment to maintain or improve the network security.

Also, the market for secure software works as a "market for lemons", a market in which customers, everything else equal, cannot tell ex-ante the difference between a secure product and an insecure one, and they will naturally choose the less expensive, which will likely be the worse. As security incidents do not occur regularly, this is a cost-effective strategy in the short-term, but the net result is that there are no widespread incentives in providing and acquiring secure software and safeguard the network for long-term gains.

To worsen the situation, as an Internet service invests in market positioning, it has no incentives to diverge its investments to its security, putting the network further in danger. As the service reaches a profitable position, it has incentives to lock-in its users and differentiate its software and protocols to enforce it. In this case, the benefits of the service security are not shared by all the network, the service is not profiting from the positive network externalities of battled and safe open software and standards, and if successful, the service signals to the market that it would be profitable to do the same. So the results are the disruption of positive network externalities, more fragile services, and a worse network security.

Differently, malevolent nodes and clients have an incentive to invest in building and sharing tools and techniques to attack the system, because the expected benefits of attacks trump their costs and they are favored by the disincentives of nodes to secure themselves and the network adequately. This situation has caused an increase in attackers' capacity and a spike of attacks targeting important segments of entire countries.

3 Cryptographic primitives

Yobicash makes use of standard cryptographic primitives to reach its goals. The advantages are that they are well understood and safer and simpler than new cryptographic constructions, while the tradeoffs are that they are less powerful, and in particular cases less efficient and that most of them could not resist a quantum attack.

In the future, the primitives used will have to be improved to defend the network from malicious use of quantum computing [4, 5], but even there the choice of the primitives will have to give priority to simplicity and security.

3.1 Hashing Function

A hashing function H is a cryptographic function $H(m) = d$ where m, the message, is a binary string of varied length and d, the digest, a binary string of fixed length. A hashing function is preimage-resistant when the probability to extract m from digest d is negligible, and collision resistant when the likelihood of finding two strings m and m' where $H(m) = H(m')$ is negligible. For this reason, hashing functions are one-way cryptographical functions, and are often used for cryptographic authentication of binary strings.

The hashing algorithms used SHA3-512 [6] and Balloon Hashing [21]. Balloon makes use of SHA3-512, which is preimage and collision resistant and is thought to be resistant to quantum attacks [5].

3.2 Message Authentication Code

A message authentication code protocol MAC is a cryptographic construction that maps a binary string of fixed length k, the key, and a varied length binary string m, the message, to a fixed length binary string t, the tag. The function has to be preimage and collision resistant. The tag t is used to authenticate the message m.

The protocol proceeds in the following way:

- generates an authentication tag t for the message m with the key k as $t = MAC(k, m)$;
- given a tag t, a key k, and a message m, for $t' = MAC(k, m)$ the authentication accepts if $t' = t$.

Yobicash uses MAC used by Yobicash is HMAC [7], which employs a preimage and collision-resistant hashing algorithm. The hashing algorithm used is SHA3-512.

3.3 Key Derivation Function

A key derivation function is a collision-resistant cryptographic function $KDF(p) = k$, where p is a binary string of varied length and k is a binary string of fixed length, used as a cryptographic key.

Yobicash uses HKDF [8], a key derivation function that uses a preimage and collision-resistant hashing algorithm, in this case, SHA3-512 [CIT].

3.4 Discrete Logarithm Problem

The Discrete Logarithm Problem states that given a group $\boldsymbol{G}$ of order q a generator g of $\boldsymbol{G}$, an element x of $\mathbb{Z}$ and $y = g^x$, it is computationally hard to find x given only y and g. x is called the discrete logarithm of y to the base g.

The problem is a base assumption of most public key cryptography available today, but it can be solved using the the Shor algorithm, a quantum algorithm.

3.5 Symmetric Encryption

A symmetric encryption algorithm is a cryptographic protocol E = *(encrypt, decrypt)*, where encrypt maps a binary string p, the plaintext, and a binary string of fixed length k, the key, to a binary string c, the cyphertext, decrypt maps c and k back to the plaintext p. The protocol has to ensure that it is computationally hard to extract the plaintext p or the key k from the cyphertext c alone. The key k is generated with a gen function that given a security parameter creates a random binary string of fixed length.

Given a function $gen(\cdot)$ that outputs a random byte string from a security parameter λ, and a plaintext p:

- $k = gen(\lambda)$;
- $c = encrypt(p, k)$;
- $p = decrypt(c, k)$.

In some constructions, the encryption and decryption operations may require other data in addition to the key, the plaintext, and the ciphertext. The algorithm used in Yobicash is AES-256-GCM [9], which uses 256-bit length keys and is considered resistant to quantum attacks [5].

3.6 Elliptic Curve Integrated Encryption

An elliptic curve integrated encryption algorithm *ECIES* [10] is a cryptographic protocol for authenticated encryption based on an elliptic curve of a group $\mathbf{G}$ of order q and a generator g of $\mathbf{G}$. The encrypter and the decrypter each generate a pair of keys *(pk, sk)*, where *pk* is the public key, and *sk* is the secret key.

The two parties exchange their public keys pk_e and pk_d, which are used to build a shared key S that is then used to produce a symmetric encryption key k through a key derivation function *KDF* to encrypt the plaintext p. Then a message authentication code *MAC* authenticates the cyphertext c.

Setup:

- The encrypter e and the decrypter d generate their keypair (pk_e, sk_e) and (pk_d, sk_d), where $sk \xleftarrow{\$} [0, q-1]$ and $pk = g^{sk}$;
- the parties exchanges their public keys pk_e and pk_d.

Authenticated encryption:

- the encrypter e derives the elliptic curve point $P = (x, y) = {pk_d}^{sk_e} = g^{sk_d sk_e}$ and accepts if P is not the point at infinity, which means if $P + q \neq q$;
- e derives the shared key S as $S = x$, where x is the x field member of the point P;
- e derives the symmetric encryption key k as $k = KDF(S)$;
- e encrypts the plaintext p in a ciphertext c where $c = encrypt(p, k)$;
- e authenticate c with the tag $t = MAC(k, c)$.

Authentication and decryption:

- the decrypter d derives the elliptic curve point $P = (x, y) = {pk_e}^{sk_d} = g^{sk_e sk_d}$ and accepts if P is not the point at infinity, which means if $P + q \neq q$;
- d derives the shared key S as $S = x$, where x is the x field member of the point P;
- d derives the symmetric encryption key k as $k = KDF(S)$;
- d authenticates the ciphertext c with tag $t' = MAC(k, c)$ and accepts if $t' = t$;
- d decrypts the plaintext p from cyphertext c with $p = decrypt(c, k)$.

The *ECIES* algorithm uses elliptic curve *curve25519* [11], the key derivation function HKDF with SHA3-512, the symmetric encryption algorithm AES-256-GCM and the message authentication algorithm HMAC with SHA3-512.

3.7 Zero-Knowledge Proof

Zero-knowledge proofs allow a prover P to prove to a verifier V that it, she or he has a solution to a problem, the instance x, without revealing it, but by using a proxy w, said witness of the instance. More formally, a zero-knowledge proof *ZKP* is an interactive (non-interactive) proof system *(Prove, Verify)* for a relation $R = (x, w)$ of a language $L = (X, W)$ where x is an element of the

set X and w of the set W, and where it is computationally hard to extract the instance x only knowing the witness w or running *Verify*.

In a zero-knowledge proof, the verifier sends to the prover one or more challenges which may depend on random values. A zero-knowledge proof is said public-coin when the random values chosen by the verifier are made public, private-coin otherwise.

3.8 Sigma Protocol

A Σ protocol [12] is a 3-steps public-coin interactive zero-knowledge proof $ZKP = (Prove, Verify)$ of an NP language. In the first step of a sigma protocol, *Prove* outputs an instance x. Next, *Verify* outputs a random challenge c, the public coin. Finally, *Prove* replies with a response r that *Verify* may accept or reject by outputting 1 or 0.

1st step:

- given $R = (x, w)$, the prover derives a binary string $a = Prove(x, w)$;
- the verifier receives *(w, a)*.

2nd step:

- the verifier generates a challenge $c = Verify(x)$;
- the prover receives the challenge c.

3rd step:

- the prover derives the response $r = Prove(c, w)$;
- the verifier receives r, and the protocol accepts if $Verify(a, c, r) = 0$ or it rejects if $Verify(a, c, r) = 1$.

More Σ protocols can be composed, obtaining an Or Σ protocol, that accepts only if at least one of the sub-protocols accepts, or an And Σ protocol, that rejects if one of the sub-protocols rejects.

3.9 Fiat-Shamir Transform

The Fiat-Shamir Transform or Heuristic [13] is a method to transform an interactive public-coin zero-knowledge proof to a non-interactive one. It relies on the use of a preimage- and collision-resistant hashing function to generate the challenge, which is the hash of the public coins of the protocol.

- given $R = (x, w)$ and some public coins θ, the prover derives a binary string $a = Prove(x, w)$;
- the prover generates a challenge $c = H(\theta, a)$;
- the prover derives the response $r = Prove(c, w)$;
- any verifier can verify that $c = H(\theta, a)$ and if the protocol accepts with $Verify(a, c, r) = 0$ or it rejects with $Verify(a, c, r) = 1$.

The Fiat-Shamir Transform is secure under the random-oracle model.

3.10 Schnorr Protocol

The Schnorr Protocolc [14] is a public-coin Σ protocol where, given a group $\mathbf{G}$ of order q, said generator of $\mathbf{G}$, a natural number x where $x \in [0, q-1]$ and an element w of the group $\mathbf{G}$ where $w = g^x$. The public coin is the generator g. The protocol is zero-knowledge assuming the hardness of the Discrete Logarithm Problem in the random-oracle model.

1st step:

- the prover generates u as $u \xleftarrow{\$} [0, q-1]$ and the commit $t = g^u$;
- the verifier receives *(w, t)*.

2nd step:

- the verifier generates the challenge $c \xleftarrow{\$} [0, q-1]$;
- the prover receives c.

3rd step:

- the prover generates the response $r = u + xc \ (mod\ q)$;
- the verifier receives r, and the protocol accepts if $g^r = tw^c$.

If the prover was truthful, then $g^r = g^{u+xc} = g^u g^{xc} = tw^c$, and assuming the hardness of the Discrete Logarithm Problem in the random-oracle model, the instance x cannot be leaked from the relation.

Through the Fiat-Shamir Transform, the protocol becomes non-interactive:

- the prover generates u as $u \xleftarrow{\$} [0, q-1]$ and the commit $t = g^u$;
- the prover generates the challenge $c = H(g, t)$;
- the prover derives the response $r = u - cx(modq)$;

The protocol accepts if:

- given the public coins g, w and t, $c = H(g, w, t)$;
- given g, w, t and c, $g^r w^c = g^{u-cx} g^{xc} = g^u = t$.

The protocol is the first historical example of a Σ Protocol, and of a zero-knowledge system for anonymous credentials.

Yobicash uses the Fiat-Shamir transformed Schnorr protocol on *curve25519* [11].

4 Design

Yobicash is aimed to be a simple system purposed for financial and data exchanges. Complex software is more difficult to verify, easier to attack and harder to improve. As software complexity tends to increase during its lifetime, it is important to have a as simple a structure as possible from the start. This principle has guided the general design of the Yobicash system.

The transaction is the principal data type in Yobicash. A transaction is a transfer of coins and data from a source wallet to one or more destination wallets.

Transactions have outputs and inputs. Outputs register the data and coins sent, while inputs reference past transaction outputs toward the transaction owner. An output can store b bytes encrypted data, where b is its amount of coins. As the output is spent, its data is dropped, relieving the network.

Nodes are paid in coins to store transactions and compete in a market for fee meant to reduce the costs of the system. This measure helps network liveness in two ways: First, it incentivizes nodes to be active and honest, and second, it disincentivizes potential attackers.

Consensus is reached on a dagchain, a directed acyclic graph of transactions, where every transaction verifies its inputs while the payment of nodes disincentivizes clients to attempt double-spending attacks.

4.1 Anonymous Credentials

Yobicash credentials are anonymous and untraceable, so the involved parties know just what is needed for a one-time transfer of coins and data between them. Even if credentials are anonymous, anyone can verify cryptographically that the transfers made are legitimate.

The method used to prove one's identity uses the Schnorr protocol, made non-interactive through the Fiat-Shamir Transform. Users share one-time credentials associated with a secret attribute. In the proof, the one-time credential is the witness w of the protocol, while the instance x is the secret attribute, which, proven, identify the legitimate owner of a transaction output.

4.2 Encryption

Data in Yobicash is encrypted with *ECIES*, to ensure that only the parties involved can access the shared data. Public key reuse is forbidden and enforced by using the anonymous credential w as *ECIES* public key. The use of public key cryptography obviates the need to rely on insecure channels of communication to build shared keys, reducing the attack surface of the system.

4.3 Dagchain

A dagchain is an authenticated directed acyclic graph (DAG) D with one or more root nodes, where every node cryptographically verifies its parent nodes. Nodes in Yobicash are transactions, edges are $(output, input)$ pairs, and authentication is given by the use of the checksum of the parent transaction built through a preimage- and collision-resistant hashing function.

The Yobicash dagchain enforces the following properties:

- every transaction is a progenitor of one or more genesis transactions, root nodes of the DAG;
- every transaction inputs from valid existing transactions using their checksums, so authenticating them.

Compared to a blockchain, a dagchain allows for faster confirmation times because transactions are in confirmation as they are known to the nodes and are confirmed as they are known to n nodes. In a blockchain, transactions have to be included in a block that has to be confirmed by n children blocks. Blockchains impose an hard-coded time t between a block and its children so that the confirmation time of a transaction tx is at least nt from its inclusion in a block. The time to include a transaction tx into a block varies according to the number of incoming transactions n_T, and the maximum size of a block MAX_SIZE, so

that if $MAX\_SIZE > n_T$, the probability that tx is included increases, while it decreases as n_T grows.

Dagchains are the product of years of research on blockchain scalability [15, 16, 17], culminated in projects as Byteball [18], Iota [1], and Dagcoin [19].

4.4 Wallet

A wallet is a tuple $wallet = (C, T_{in}, T_{out})$, where:

- C is a set of tuples $c = (g, x, w, id_{tx}, idx_o)$, where g is a generator of a group $\boldsymbol{G}$ of order q, x is an instance of the Schnorr protocol $x \xleftarrow{\$} [0, q-1]$, w is a witness of the Schnorr protocol $w = g^x$, id_{tx} the id of the transaction output of index idx_o which sent coins and data to w;
- $T_{in}$is a set of transactions where a subset of their outputs are directed to one of the identities in$C$;
- O_{in} is the set of the outputs in T_{in} directed to confidential identities in C;
- $T_{out}$is a set of transactions which inputs injectively map the outputs in the transactions of T_{in};
- O_{out} is the set of the outputs in T_{out}.

A wallet is valid if:

- $\bigcup_{c \in C}(id_{tx_c}, idx_{o_c}) = \emptyset$and $\bigcup_{c \in C} w_c = \emptyset$, which means that witnesses can be used only once;
- given $T = T_{in} \cup T_{out}$, every transaction in T is valid according to the dagchain and transaction validation rules.

Given a function $\sigma(\cdot)$ that extract the total amount of coins spendable from an output, the balance of a wallet $wallet$ is $b = \sum_{o \in O_{in}} \sigma(o) - \sum_{o \in O_{out}} \sigma(o)$.

4.5 Input

Input is a reference to a transaction output already registered in the dagchain. More formally, given an output o in the set O_{in} in the wallet W, an input i of the output o is a tuple $input = (id_{tx}, idx_o, h, g, t, r)$, where:

- $id_{tx}$is the id of the transaction containing the output o;
- $idx_o$is the index of the output o in the transaction identified by id_{tx};

- the height h, where $h \in \mathbb{Z}^+$;
- a generator g of the group $\boldsymbol{G}$ of order q;
- a Schnorr protocol commit t;
- a Schnorr protocol response r.

The input is valid if:

- the transaction with id id_{tx} exists in the dagchain D;
- the output with index idx_o exists in the transaction id_{tx} and has height $h' = h - 1$;
- given the transaction tx_{sender} containing the output o to the witness $w_{receiver}$ of the wallet $W_{receiver}$ from the witness w_{sender}, the partial transaction *tx'* of the transaction tx which will contain the input i spending the output o, for $c = H(g, H(T), H(o), H(T'), w_{sender}, w_{receiver}, t)$, it occurs that $g^r w^c = g^{u-cx} g^{xc} = g^u = t$.

The input does not leak any information about the secret instances of the parties, and it is bound to the transaction tx through the commitment c, making impossible to any attacker to steal the coins and data in o without having access to the wallet W before tx is broadcasted to the network.

4.6 Output

Output is a transfer of coins and data to a credential $w_{receiver}$ of $wallet_{receiver}$. More formally, an output is a tuple $output = (w_{sender}, w_{receiver}, m, d, t, t')$, where:

- w_{sender} is a witness of a Schnorr protocol of a wallet W_{sender};
- $w_{receiver}$ is a witness of a Schnorr protocol of a wallet $W_{receiver}$;
- m is a Yobicash amount, and $m \in \mathbb{Z}^+$;
- d is the AES-256-GCM encrypted data;
- t is the authentication tag of d;
- *t'* is a custom 256-bit cleartext bytestring.

The output is valid if:

- $w_{sender} \neq w_{receiver}$ and they are both unique in the dagchain D;

- given a function $size(\cdot)$ mapping a binary string to its size, $size(d) = m$ or $size(d) = 0$.
- Only the receiver can authenticate and decrypt the data d, by deriving the shared secret key from w_{sender} and $w_{receiver}$, by using as initial vector $iv = H(w_{sender}, w_{receiver}, m, t')$ and by authenticating through the MAC authentication tag t.

Outputs containing data that get spent by other transactions, have their data dropped after the spending transactions are confirmed.

4.7 Transaction

A transaction is a transfer of coins and data from a wallet to one or more wallets. More formally, a transaction is a tuple $tx = (id, t, I, O)$ of a wallet W_{sender}, where:

- id is the checksum of the transaction without id, or formally, $id = H(tx')$ where $tx' = (t, I, O)$;
- t is the transaction timestamp and $t \in \mathbb{Z}^+$;
- I is a set of inputs referencing wallet outputs in the set O_{in} of the wallet W_{sender};
- O is a set of outputs from the wallet W_{sender} to a set of wallets $\boldsymbol{W}_{receivers}$.

A transaction tx is valid if:

- $id = H(tx')$, where $tx' = (t, I, O)$;
- given the current time t_{now}, the set of timestamps from the referenced transactions K_{old}, a duration constant ε, $\{t \mid t \leq t_{now} + \varepsilon \wedge \forall t_{old} \in K_{old}, t \geq t_{old} - \varepsilon\}$;
- inputs in I are valid;
- outputs in O are valid;
- there is no quantity inflation nor deflation or given a function $\sigma(\cdot)$ that extracts the amount of an output and a function $\zeta(\cdot)$ that maps an input to its referenced output, $\sum_{i \in I} \sigma(\zeta(i)) = \sum_{o \in O} \sigma(o)$;
- there is no doublespending, which means that given the dagchain D and T_D the set of transactions in the dagchain, at time $t' \leq t + \varepsilon$, no input in I is present in T_D.

4.8 Coinbase

A coinbase is a mint of coins $cb = (id, t, PoSt, PoW, O)$ of a wallet W_{minter}, where:

- $id = H(cb')$, where $cb' = (t, PoSt, PoW, O)$;
- t is the coinbase timestamp and $t \in \mathbb{Z}^+$;
- $PoSt$ is a tuple $PoSt = (id_{tx}, difficulty, nonce, A_{id_{tx}}digest)$, where:
  - $id_{tx}$is the id of a transaction tx;
  - $difficulty \in \mathbb{Z}^+$;
  - $nonce$ is a random integer and $nonce \in \mathbb{Z}^+$;
  - $A_{id_{tx}}$ is the set of the ancestor transactions of the transaction with $id = id_{tx}$, ordered by id;
  - $digest$ is $H(c_0|c_1|...|c_{difficulty-1})$ on the set $I_{tx} = \{tx\}$, where:
    - c_i is the byte of index $idx_i = to\_int(H(idx_{i-1}))$ mod $size(tx_i)$, with $idx_0 = to\_int(H(nonce))$ mod $size(tx_0)$ and $tx_i$the ith transaction in I_{tx};
    - the set $I_{id_{tx}} = \{tx_{idx_i}\}$ for $i \in \mathbb{Z}^+ \wedge 0 \leq i < difficulty$, is the set of transactions which id has index $idx_i = to\_int(H(idx_{i-1}))$ mod $|A_{id_{tx}}|$ in $A_{id_{tx}}$ and $idx_0 = to\_int(H(nonce))$ mod $|A_{id_{tx}}|$;
- PoW is a tuple $PoW = (digest_{PoSt}, nonce, s_{cost}, t_{cost}, \delta, memory, digest)$, where:
  - $digest_{PoSt}$ is the digest of the coinbase $PoSt$;
  - $nonce$ is a random integer and $nonce \in \mathbb{Z}^+$;
  - s_{cost}, t_{cost} and δ are the Balloon Hashing parameters;
  - $memory$ is $Balloon_{memory}(digest, cost, t_{cost}, \delta) - Balloon_{memory}(digest, difficulty_{PoSt}, difficulty$
  - $digest$ is $Balloon_{hash}(digest_{PoSt}, nonce, s_{cost}, t_{cost}, \delta)$.
- O is a set of outputs from the wallet W_{minter} to a set of wallets $\boldsymbol{W}_{receivers}$.

A coinbase cb is valid if:

- $id = H(cb')$, where $cb' = (t, PoSt, PoW, O)$;
- given the current time t_{now}, the set of timestamps from the referenced transactions K_{old}, a duration constant ε, $\{t \mid t \leq t_{now} + \varepsilon \wedge \forall t_{old} \in K_{old}, t \geq t_{old} - \varepsilon\}$;
- PoSt is valid;
- PoW is valid;

- outputs in O are valid;
- the total output amount is $memory = Balloon_{memory}(digest, cost, t_{cost}, \delta) - Balloon_{memory}(digest, difficulty_{PoSt}, difficulty_{Post}, difficulty_{PoSt})$.

PoSt can be easily verified by every node storing the ancestor transactions of the transaction with *id* id_{tx}. *PoW*, as *PoSt* is verifiable by all the nodes with at least $Balloon_{memory}(digest, cost, t_{cost}, \delta)$ available and it allows to link coin creation to the amount of available memoryin the network.

4.9 Node

A node is a tuple $node = (W, I, S, c)$, where:

- W is the node wallet;
- I is the set of network addresses of the node;
- S is the store of the wallet;
- c is a tuple containing the node configurations.

A node requests, receives, validates, stores, and sends transactions and coinbases. To ensure the liveness of the network, Yobicash incentivizes node activity. Every node can ask for a fee transaction for storing one or more new transactions and produce coins through mining.

A fee transaction has an output to a witness w of the node wallet W and has to be sent before the new transactions it is paid for, including itself, and has to have only inputs to already confirmed transactions, to hedge nodes against double spenders and discourage attackers.

The amount of the fee depends directly on the size of the transactions to store and the maximum latency set by the node to avoid denial of service attacks. Fees are denominated in bytes per transaction, but they are set to MAX, a maximum amount set to be unlikely to reach, when the latency wall is reached, to make any new request economically unfeasible. As the wall would be put at a latency lower than the one that would make the node incapable of serving new requests, the minimum price for a denial of service attack would become $P_{DoS} = MAX + c$, where c would be the size of the data required to attack the node successfully. As $P_{DoS} > MAX$ and transaction fees have to be sent before the associated transactions, a denial of service attack to one or more nodes would be unfeasible.

Fee transactions have many benefits:

- they incentivize nodes to keep their nodes running, for mining and fee-collecting may be profitable in the long run;

- they disincentivize double-spending;
- they increase the edge degree (the connectedness) of transactions, and so they reinforce the security of the dagchain;
- they protect nodes from DoS and DDoS attacks.

While requests for storing transactions are paid, requests for stored transactions and stored set of node peers are free, but rate limited.

As the value of the currency increases, so does the number of nodes joining the network. Market forces tend to deflate fees and inflate the value of the currency, leaving in the network only the nodes capable to compete. In order to have any significance, an attacker joining the network as a node would have to make an upfront investment in economic resources and make profits in the long-run.

4.10 Consensus

Yobicash nodes reach consensus on transactions by querying other nodes. A node asks its peers about the state of a transaction: if it is known or not, and if it is a doublespending transaction or not.

Given an incoming new transaction tx, a duration constant τ, an error duration constant ε, the set of active peers U, a function $\rho(\cdot)$ for extracting the first time the node has been seen, and the set $U_{old} = \{u \in U \mid \rho(u) < t_{now} - \tau + \varepsilon\}$ of the oldest known active peers, the algorithm proceeds in the following way:

- check if tx is valid, querying a sample of $n_{query} < |U_{old}|$ from U_{old} for missing transactions, and prevent it if it is not;
- query a sample of $\frac{2}{3}|U_{old}|$ to know if any of the outputs linked to its inputs are already spent by some transaction, or are known doublespends, tagging them as doublespends if the first case if true.

Queried nodes provides a coinbase containing a proof-of-storage of the transaction ancestors. In the case the coinbase is valid, including the proof-of-storage, it is added to the dagchain.

The $\frac{2}{3}$ ratio is a standard introduced by the PBFT paper [22]. Given N nodes which f are faulty, $2f + 1$ is the number of to ensure the functioning of the network. With $f + 1$ liveness of the network is saved, but not consensus, with $2f + 1$ both liveness and consensus of the network are preserved as the majority of the network is not faulty. As $N = f + 2f + 1 = 3f + 1$, $\frac{2}{3}N$ nodes agreeing on the state of a transaction tx are enough to state that the network has reached consensus.

Supposing a fully connected topology of 4,000 nodes, to reach consensus on the state of a valid transaction would require at least 2667 requests. In the

worst case scenario, every node would receive 4,000 peer queries per second, which is not a big number considering that modern servers are built to handle 50,000 requests per second. In this scenario, the confirmation time would be a few seconds and would depend mostly on network and storage latencies. As running a node requires investment in resources, it is likely that many nodes will be stable in the network, and that it will likely have a fully connected topology.

In the case of a newly added unspent transaction, the propagation time and consequently the confirmation time would depend on just one factor: the number of spendable coins in transaction fees by the transaction owner. The more fee transactions the owner can pay, the more nodes it will be able to reach. The same applies for nodes, which act as wallet clients when they want to push new transactions to the network.

When a node knows of an unknown transaction spent by one or more transactions it is validating, it requests it and starts its validation. By consequence, the consensus algorithm is the fastest way a regular transaction gets shared in the network because it does not require to economize on node requests. So the consensus algorithm prioritizes consensus on past transactions against new ones, to have a consistent state and be able to decide quickly on the status of new incoming transactions.

This strategy ensures that nodes have a consistent view of past states of the dagchain and allows for the reduction of the validation and confirmation time of new transactions. An attacker would have to join the network for a long term and make upfront investments in economic resources to try to enter in the oldest set of peers of different nodes. Even in this case, as the size of the network increases, it would be impossible to have a weight on the consensus mechanism with one node, and the attacker would have to invest in multiple nodes, and that number will grow in tandem with the network size.

5 Security

Yobicash ensures confidentiality, integrity, and availability of data against most attacks, and achieve it thanks to the use of cryptographical tools and incentive design in the set of computer networks. Cryptography is enough to guarantee the security of one node, but not enough to secure an open network of nodes, hence the use of a consensus mechanism incentivized by an internal currency.

Although all those tools assure a good information security to the network, more can be done to increase Yobicash confidentiality, integrity and availability.

5.1 Confidentiality

Yobicash identities and data are confidential assuming the hardness of the Discrete Logarithm Problem in the random-oracle model. Although the problem

is at the base of most cryptographical tools used nowadays, it is vulnerable to a quantum attack using the Shor algorithm [4, 5].

Quantum computers capable of directing a similar attack will be a reality in 10-20 years. Post-quantum cryptography is a young but very active field, so it is possible to expect that before ten years from now, there will be safe cryptographical primitives to substitute those currently used in Yobicash.

Clients and nodes access patterns are a source of data leaks and vulnerability. It is possible to gain sensitive information by analyzing the frequency, the direction and the content of requests, even when most of the content is encrypted. Requests should be executed in encrypted full-duplex connections to avoid leaking their content or number.

What is harder not to leak is IPs and locations, which are transparent to the Wide World Web. For attackers with substantial resources, the ability to leak the locations of clients and nodes is paramount to attack the network, and so this is an issue to address.

Making location leaks harder would further reduce the ability to collect clients, and nodes access patterns, increasing the overall information security of the network.

An attacker cannot break data and identity confidentiality without the use of quantum computers. However, if location and connection data are transparent, attackers can use this information to target single clients and nodes. It is possible to use tools such as meshnets and mixers to reduce the ability of attackers to obtain location and access patterns.

5.2 Integrity

Yobicash applies different measures to increase data integrity.

Checksums allow to verify whether transactions have been illegitimately modified after their creation. Authenticated encryption of data enables the same for ciphertexts. Nodes and clients can always retrieve integer versions of the altered transactions from other nodes and clients, as they will eventually be requested in the execution of the consensus algorithm when any of their outputs will be spent by other transactions.

A further option could be to use forward error correction. The problem with this solution is that an attacker could always alter the section containing the bits required to recover the transaction, making the measure useless.

An attacker could undermine the integrity of Yobicash data only by dropping or altering packets in the communication channel or by accessing nodes or clients. For the first option, given an average number C of concurrent connections between a node and its peers and clients, and given the number N of network

nodes, an attacker should be able to attack at least $\frac{2}{3}$ of the nodes, hence launch a man-in-the-middle attack against $\frac{2}{3}NC$ connections, or insulate one or more particular clients. Only this last option would be feasible, but it can be managed by using meshnets and similar solutions. The second option is harder for nodes, who are incentivized to invest in their security to secure their fee transactions, while empirically clients tend to have lower security. This latter case can be handled by using light clients connected to network nodes, which would act as wallets for end users.

5.3 Availability

Data availability requires the liveness of the network, the full replication of data, and low latencies.

As saw in section 4.9, spent transactions which are not known by all the nodes get acquired for free as any transaction that inputs them makes their existence known. Differently, new unspent transactions are pushed to the network in exchange for a fee. In this case, old transactions are more available than new ones, which are still known by few nodes. As they get spent, their spending transactions trigger nodes that do not have them in their store to look for them, validate them, and save them, increasing their availability.

In the worst case, the consensus algorithm requires any node that at least $\frac{2}{3}$ of its peers agrees on the state of a transaction. The subset may not be the $\frac{2}{3}$ of the entire network, as this scenario would require a fully connected topology, or that a particular node know of every active node of the network. So as the set of peers known by single nodes may intersect with the set of peers of other nodes, more connected nodes will also be those more capable to ensure data availability of a transaction at any time t. So said, the consensus algorithm affects an increasing replication of transactions and an eventual full replication of data.

While a distributed system with full replication can propagate and confirm data faster, a decentralized system trades the velocity of replication for more resilience against attacks, network failures and other failures that may affect nodes. So under the same attack and with the same data sets, a distributed system may permanently lose some data, making it unavailable, while a decentralized system would be better prepared to handle the attack and have some nodes still capable of retrieving it.

Yobicash is a decentralized system with full replication of data, and it incentivizes nodes to store new information thanks to transaction fees and coinbases and the fact that data is strictly linked to the transaction amounts (a 1:1 or 1:0 relation) and is dropped after the container outputs are spent by confirmed transactions. As shown in section 4.8, the cost to launch a denial of service attack with store requests to a node is $P_{DoS} = MAX + c$, so given a network of N nodes, a successful attack against the system would require a cost of $P_{DoS_{net}} = \frac{2}{3}N(MAX + c)$.

Both the attacks are economically unfeasible. An attacker may opt for free requests, but those requests are rate-limited, and even if a node were not able to serve non-attackers, other nodes would be still available, and at the increase of the size of the network, the failure rate of similar attack would increase.

5.4 Incentives

As in Yobicash data has economic value, nodes have an incentive to secure it, while clients have an incentive to not misuse it, because storing data has a cost that increases linearly with its size. Fees remove clients' free-runs and nodes' perverse incentives to look for gains through the misuse of user data.

Differently from common nodes, Yobicash nodes are incentivized to invest in their security to save their earnings, and they have to focus on very restricted zones of improvement because the Yobicash protocol already guarantees most of the information security they need. Nodes have an incentive to secure their past and future fees by playing along with their peers and run the protocol used by the expected majority. So they have no incentive on custom solutions to capture more capital, for they would cost more to keep aligned with the protocol consensus and they would have a greater development and management cost in general.

Additionally, in an open network, barriers to entry are too low to ensure returns on substantial investments for positioning as oligopolies or monopolies. Nodes which made it to, would put in danger the information security of the system, lower the value of the network and the value of the currency and, by consequence, the value of the fees they collected and of their future returns.

At the same time, the costs incurred in pushing node fees to the network and mint coins require nodes to invest upfront to be sustainable. Attackers joining the network to disrupt it would have to be willing to burn economic resources. The barrier to entry would increase during time with the increase of the network size, for market forces would deflate fees and inflate the value of the currency. This occurs also in other proof-of-work blockchain-based systems, in which the economic resources required to attack the system grow in parallel to the blockchain length and network size [**?**].

6 Future Work

Yobicash is meant to be the most information-secure [2] cryptocurrency available at the current technological level, but it will require more work to make it optimal. We review the issues that are already noted in other sections of the paper.

For more confidentiality, it is important to be able to hide the locations of clients and nodes from potential attackers. At the time there are different tools available to address this issue, but they have to be carefully evaluated.

To ensure security, incentive-compatibility, and consensus, Yobicash uses a storage market where nodes bid their space to clients. This market process may bring a latency penalty that is counter-balanced by the utilization of a dagchain, which by design has lower latencies than a blockchain. Nonetheless, both storage price and latencies have to be lowered to increase the utility of the network and assure that more type of applications can be built on top of it. It will be important to continue research on measures to increase available storage, and decrease fees and latencies sustainably.

The other issue is its vulnerability of its pre-quantum cryptography to quantum attacks [5]. Post-quantum cryptography is still a young field of research, but cryptographers are already finding a consensus around certain primitives [4]. As the security of post-quantum cryptography will be satisfiable, it will be important to start evaluating possible post-quantum substitutes to the elliptic curve cryptography used in Yobicash.

7 Conclusions

Yobicash is a cryptocurrency meant to increase the information security on the Internet by ensuring confidentiality, integrity, and availability of shared and stored data. It is anonymous thanks to the Schnorr protocol, a zero-knowledge proof based on elliptic curve cryptography, and valid under the assumption of the hardness of the Discrete LogarithmProblem in the random-oracle model. It encrypts data using the ECIES protocol, always based on elliptic curve cryptography, the Discrete Logarithm Problem and the random-oracle model. It ensures data availability through full replication, and scalability through a dagchain, a directed acyclic graph of authenticated transactions. It incentivizes network nodes with an internal currency, the value of which is based on the fact that it enables users to store and securely share their data on the network. It incentivizes nodes to invest in security to secure their gains, and discourages clients from free-running on the network and putting its economic sustainability and safety in danger.

Nonetheless, although a scalable, well-incentivized and safe solution, Yobicash is not perfect. Future work will be needed to increase the security of its cryptographical primitives, which are based on pre-quantum cryptography, to increase node storage and lower latencies to make it more scalable, to remove the lack of confidentiality of its clients and nodes locations.

References

[1] *S. Nakamoto*, Bitcoin: a Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf, 200

[2] *S. Popov*, The Tangle, https://iota.org/IOTA_Whitepaper.pdf, 201

[3] *Wikipedia*, Information Security, https://en.wikipedia.org/wiki/Information_securit

[4] *R. Anderson*, *T. Moore*, The Economics of Information Security, https://www.cl.cam.ac.uk/~rja14/Papers/sciecon2.pdf, 200

[5] *D. Augot, L. Batina, D. Bernstein et al.*, Initial Recommendations of Long-Term Secure Systems, https://pqcrypto.eu.org/docs/initial-recommendations.pdf, 201

[6] *T. Lange*, Standardization of Post-Quantum Cryptography, https://pqcrypto.eu.org/slides/awacs.pdf, 201

[7] *G. Bertoni*, *J. Daemen*, *M. Peeters*, *G. Van Assche*, The Keccak Reference, https://keccak.team/files/Keccak-reference-3.0.pdf, 201

[8] *H. Krawczyk*, *M. Bellare*, *R. Canetti*, HMAC: Keyed-Hashing for Message Authentication, https://www.ietf.org/rfc/rfc2104.txt, 199

[9] *H. Krawczyk*, *P. Eronen*, HMAC-based Extract-and-Expand Key Derivation Function (HKDF), https://tools.ietf.org/html/rfc5869, 201

[10] *NIST*, Advanced Ecryption Standard (AES), http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 200

[11] *V. Shoup*, A Proposal for an ISO Standard for Public Key Encryption, http://www.shoup.net/papers/iso-2_1.pdf, 200

[12] *D. J. Bernstein*, Curve25519: New Diffie-Hellman Speed Records, https://cr.yp.to/ecdh/curve25519-20060209.pdf, 200

[13] *I. Damgård*, On Σ-protocols, http://www.cs.au.dk/~ivan/Sigma.pdf, 201

[14] *A. Fiat and A. Shamir*, How to Prove Yourself: Practical Solutions to the Identication and Signature Problem, Proc. of Crypto 198

[15] *C. Schnorr*, Efficient Signature Generation by Smart Cards, Journal of Cryptology vol. 4 (3), 199

[16] *Y. Lewenberg*, *Y. Sompolinsky*, *A. Zohar*, Inclusive Blockchain Protocols, 201

[17] *B. McElrath*, Branding the Blockchain, Scaling Bitcoin 201

[18] *X. Boyen*, *C. Carr*, *T. Haines*, Blockchain-Free Cryptocurrencies, 201

[19] *A. Churyumov*, Byteball: A Decentralized System for Storage and Transfer of Value, 201

[20] *Y. Ribero*, *D. Raissar*, https://dagcoin.org/public/Dagcoin_whitepaper.pdf, 201

[21] *M. Castros*, *B. Liskov*, Practical Byzantine Fault Tolerance, http://pmg.csail.mit.edu/papers/osdi99.pdf, 199

[22] *D. Boneh, H. Corrigan-Gibbs, S. Schechter*, Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks, 2017, https://crypto.stanford.edu/balloon