UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
LABORATÓRIO DE SEGURANÇA EM COMPUTAÇÃO

ZENO Layer

Douglas Martins[1] **Douglas Simões Silva** [2] **Martín Vigil** [3]

[1] marcelino.douglas@posgrad.ufsc.br

[2]douglas.simoes@posgrad.ufsc.br

[3]martin.vigil@ufsc.br

FLORIANÓPOLIS
23RD NOVEMBER 2018

1. Introduction

In this report, we propose four solutions to be used in Rhizom Plataform ZENO Layer. We start by dealing with privacy. Privacy concerns individuals controlling what information related to them may be stored and by whom and to whom that information may be disclosed [Stallings 2016]. Moreover, privacy has become an important requirement due to law obligations (e.g., the European Union General Data Protection Regulation) and to the raise of public awareness. In this context, we devise solutions that work as zero knowledge proof protocols, i.e., they allow one to check data from participants in a blockchain without learning that data. These solutions use the help of delegates. They are blockchain distinguished participants which are trusted to carry certain operations. Our privacy-based solutions are found in sections 2, 3, and 4.

Finally, we address the issue of a blockchain participant transferring coins to an invalid wallet by mistyping its address. The solution uses Error Correcting Code and is found in Section 5.

1.1. Notation

In this document we adopt the notation found in Table 1.

Symbol	Meaning
Kr_U	Private key of user U
Ku_U	Public key of user U
$\{X\}_K$	Piece of data X (de)encrypted with key K

Table 1. Notation

2. Query Data from Past Transactions Without Learning Them

A blockchain usually preserves and publicizes all transaction its users have ever issued. On the one hand, this public database of transactions allows to learn useful information. For example, one can check whether online shop *Doe-Sports* that accepts blockchain payments has been active. On the other hand, publicizing transactions can raise privacy concerns. For instance, publicizing transactions may disclose that customer *Jane* has transacted with *Doe-Sports*.

Further privacy issues can be listed. Nonetheless, in this section we limit ourselves to addressing the following privacy problem: can a blockchain allow us to query data from past transactions, involving a particular user, such that we cannot learn these transactions?

We propose a zero knowledge proof protocol consisting of three participants. A participant U who wants to query data from past transactions involving participant S. A delegate D which is chosen by U to execute his or her query.

We now outline our protocol. Assume U wants to ask D the number transactions and the amount of coins that S received from time t_0 to t_1 in the past. Then, U submits this query to D. Next, delegate D asks S to prove that S is the recipient of some transactions from t_0 to t_1. To prove this, S needs to show that he or she controls the wallet to which the transactions were addressed. More precisely, S chooses some transactions T_i addressed to public key Ku_{T_i} and demonstrates to D that S knows the corresponding private key

Kr_{T_i}. Then, D replies U the number of transactions and the amount of coins received by S without identifying any transactions.

The protocol is detailed in Protocol 1. To verify that U is the recipient of some transactions T_i addressed to public key Ku_{T_i} and confirmed from t_0 to t_1, D asks U to encrypt a fresh nonce N with the corresponding private key Kr_{T_i}.

Protocol 1 for querying transaction data.

---

1. $U \rightarrow D : \{t_0, t_1, Ku_S\}_{Ku_D}$ where $t_0 < t_1$ are two moments in the past.
2. $D \rightarrow S : \{N, t_0, t_1\}_{Ku_S}$ where N is a fresh nonce.
3. $S \rightarrow D : \{T_1, T_2, \ldots, T_n, \{N\}_{Kr_{T_1}}, \{N\}_{Kr_{T_2}}, \ldots, \{N\}_{Kr_{T_n}}\}_{Ku_D}$ where T is a transaction addressed to U's public key in $[t_0, t_1]$ and Kr_T is the private key corresponding to U' public key.
4. $D \rightarrow U : \{m, \sum C_i\}_{Ku_U}$, where m is the number of distinct transactions T_i addressed to U's public key Ku_{T_i} and C_i is the number of coins T_i transferred to U's public key Ku_{T_i}, provided that $\{\{N\}_{Kr_{T_i}}\}_{Ku_{T_i}}$ equals N and that T_i was confirmed in $[t_0, t_1]$, $0 \leq i, m \leq n$.
5. U is convinced that S received $m = \{\{m\}_{Ku_U}\}_{Kr_U}$ transactions from t_0 to t_1, adding up to $\sum C_i = \{\{\sum C_i\}_{Ku_U}\}_{Kr_U}$ coins received.

All communication is kept private by sending data encrypted with the recipient's public key. Communication can be off-chain using private channels. Alternatively, participants could communicate addressing transactions containing a encrypted message and no coins to be spent to each other.

Although we assumed participant U asks delegate D data from transactions whose recipient is participant S, the protocol can be easily changed to reply questions as to the transactions whose originator is S. Moreover, notice that participant S chooses which transactions can be learned by delegate D, considering that high deterministic wallets are used. In addition to that, S may not want U to learn any number coins S received in the period $[t_0, t_1]$. In this case, Protocol 1 can be changed as follows. Assume U wants to know whether S has received at least L coins between times t_0 and t_1. Then, in Step 1 U sends t_0, t_1, Ku_S and L to delegate D. In Step 4, instead of sending $\sum C_i$ encrypted, D sends a boolean asserting whether $\sum C_i \geq L$.

3. Checking Credentials without Learning Them

A service may be required to check customers' credentials before transacting with them. For example, seller *Bebidas 24x7* asks user *Fernando* to demonstrate he is over 18 before selling him alcohol. However, customers may not want to disclose their credentials.

To meet the above requirements, we propose a zero knowledge proof protocol which uses the help of a delegate. More precisely, upon request a delegate confirms the customer's credentials to the service. The customer has to disclose his or her credentials to the delegate but not to the service.

We start by outlining the protocol based on the use case of checking customers age. Assume user U wants to transact with service S. Then, service S sends a fresh

nonce N_D encrypted with the public key Ku_D of delegate D to user U. Next, U presents encrypted N_D together with his or her credentials to D. Delegate D checks the credentials of U. If and only if user U is over 18 years old, delegate D decrypts N_D using his or her private key Kr_D and sends N_D to service S. Service S is convinced that user U is over 18 without learning U's credentials.

The protocol is detailed in Protocol 2 and explained in the following. Each nonce N_D is uniquely related to a user U and should not be learned by other users. For this reason, N_D is also encrypted with U's public key Ku_U by service S in Step 2 and with U's private key Kr_U by U in Step 3 (this last encryption is equivalent to a digital signature of U). In Step 4, we assume delegate D can map user U's public key Ku_U into U's credentials and verify that U is over 18 years old.

In Step 5, delegate D decrypts nonce N_D using keys Ku_U and Kr_D in this order. In addition, D sends N_D encrypted with service S's public key Ku_S and encrypted with Kr_D private key. The last encryption is equivalent to a signature of delegate D.

In Step 6, service S decrypts $\{\{N_D\}_{Ku_S}\}_{Kr_D}$ using public key Ku_D and private key Kr_S in this order.

Protocol 2 for checking credentials.

---

1. $U \rightarrow S : Ku_u$
2. $S \rightarrow U : \{\{N_D\}_{Ku_D}\}_{Ku_U}$
3. $U \rightarrow D : \{\{N_D\}_{Ku_D}\}_{Kr_U}, Ku_U$
4. D checks whether U is over 18. Abort if false.
5. $D \rightarrow S : \{\{N_D\}_{Ku_S}\}_{Kr_D}$
6. S is convinced of U's credentials if $\{\{\{\{N_D\}_{Ku_S}\}_{Kr_D}\}_{Ku_D}\}_{Kr_S}\}$ equals N_D.

It is worth mentioning that the protocol does not prevent user U from sharing his or her granted access to service S with an unauthorized user (e.g., a user who is under 18). This limitation is also true outside the blockchain. However, steps 2–3 allow to bind nonce N_D to user U, thereby holding him or her accountable for sharing access with an unauthorized user.

4. Verifying Properties of a Digital File without Learning It

Assets in the form of digital files are nowadays traded online. For example, digital art can be bought from artists' online galleries. These assets may have to fulfill requirements from buyers. For instance, a piece of digital art should be at high resolution.

Verifying these requirements can raise the following problem. On the one hand, a buyer wants to check the properties of a digital file before paying for it. This usually needs read access to the file. On the other hand, the seller wants to hand over the file only after being paid.

The solution we propose is similar to that in Section 3. The difference is that, instead of verifying the credentials of a user (see Step 4 in Protocol 2), now a delegate checks whether a seller's digital file has the properties a buyer desires. The seller may need to disclose the file to the delegate, but not to the buyer.

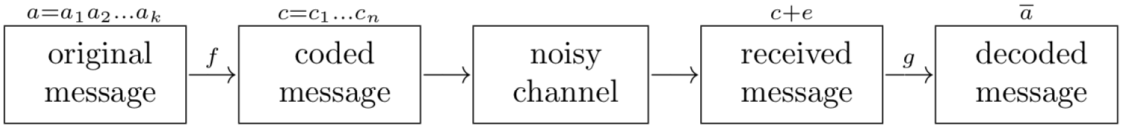

Figure 1. Transmission of a codeword through a noisy channel.

5. Preventing Transfers to Invalid Wallets

Misdirected transactions in a blockchain is one of the causes for the loss of several million bitcoins [Roberts 2017]. Ensuring the correct wallet address in a transaction can be a difficult task for humans. The reason is that the wallet appears to be a random sequence of characters to provide security. Thus, it is necessary to prevent this behavior, and implement a strategy to guarantee that all transactions are made correctly. That is, the aim of this section is to answer the question below:

> *Alice* wants to send tokens to *Bob* through the cryptocurrency based on blockchain *O*. How can she ensure that his wallet address is correct and actually belongs to him?

5.1. Strategy

To prevent one from addressing a transaction to an invalid wallet, we add a last step to the wallet generation process. It will consist of the inclusion of redundant information in the original wallet address. This extra information gives the capability of detecting and correcting some errors that occurred on the transmission or human input of the wallet address. There exist many strategies to add this redundancy in a thoughtful manner, according to coding theory. The most popular error correcting code (ECC) described in the literature is based on linear codes [McEliece 2002]. We will introduce the theory behind these concepts below in a brief way.

The aim of ECC is to prevent the information loss on the receiving end when a message is sent through a noisy channel. In order to detect or even correct some errors, linear codes add the redundancy through the transformation of the original message with respect to a given matrix, that defines an instance of a linear code. It is called the generator matrix, usually defined as G. Indeed, after the transmission of the message, we need to verify if it has errors. This is done through operations with another matrix, called the parity check matrix, defined as H. If errors are present, the redundancy added enables the recipient to solve them, thus recovering the original message. This process is demonstrated on Figure 1.

Let a be a k-bit message, $f : \{0,1\}^k \to \{0,1\}^n$ a function defined by $f(a) = aG$, and $c = f(a)$ an n-bit codeword of a, where $0 < k < n$. This is possible because a can be represented as a vector of k bits, and G has, by definition, k lines and n columns. A function g is divided in two steps. First, the application of H to the received codeword $c+e$ is computed, where e is a n-bit error vector. The result of this multiplication is called "syndrome". If it is equal to $\vec{0}$, then no errors occurred. The second part of g happens when the syndrome is not a null vector. In this case, a decoding algorithm is needed to

correct up to t errors of the received codeword[1].

Our strategy for solving the problem of misdirected transactions can be divided into two algorithms. The first is to encode the wallet address into a codeword. This process is described in Procedure 1. This step is done after the original wallet generation, and before publication of address. The second procedure checks if a wallet address is indeed correct, since addresses are now codewords, and any variations can be detected by the syndrome computation. This is described in Procedure 2.

The proposed strategy requires that all network users have access to the same linear code and its generator and parity check matrices. This can be achieved by means of choosing a specific code and publishing it on blockchain protocol. For instance, a specific smart contract may be created by delegates, or even in a specific application outside the blockchain, where users can input their wallet addresses.

Procedure 1 Generation of the ECC-embedded wallet address.

Network requirements. A generator matrix G and a parity check matrix H previously agreed by the participants.
Inputs. Original wallet address a.
Goal. Generate an ECC-embedded address, that allows the detection of any errors, and correction of up to t errors.
Description.

1. **Transform the wallet address into a codeword.**
   (a) User computes the corresponding codeword of the original address a, multiplying it by the generator matrix. This will output the $new\ address = a \times G$.
   (b) User outputs the $new\ address$.

Procedure 2 Validation of the ECC-embedded wallet address.

Network requirements. A generator matrix G and a parity check matrix H previously agreed by the participants.
Inputs. Candidate wallet address in the form $new\ address$.
Goal. Output if the candidate wallet address is a valid address or not. In the negative, output the potential corrected address.
Description.

1. **Validate the ECC-embedded address.**
   (a) Compute the $syndrome$ of the $new\ address$, by the following equation: $syndrome = new\ address \times H$.
   (b) If $syndrome = \vec{0}$
      i. Output that $new\ address$ is a valid wallet address.
   (c) If $syndrome \neq \vec{0}$
      i. Recover a potential wallet by a decoding algorithm of code and present this result.

---

[1]The amount of errors to be corrected depends directly on the construction of the code. Different codes have different values of t, since this value can act as a parameter of the code.

References

McEliece, R. (2002). *The theory of information and coding*, volume 3. Cambridge University Press.

Roberts, Jeff John Rapp, N. (2017). Exclusive: Nearly 4 million bitcoins lost forever, new study says. http://www.fortune.com/2017/11/25/lost-bitcoins. Accessed: 2018-10-01.

Stallings, W. (2016). *Cryptography and Network Security: Principles and Practice, Global Edition*. Pearson Education Limited, 7 edition.