EtherYou: Ethereum-based privacy-preserving social media platform

Pedro I. C. Oyama[1], **Jó Ueyama**[2], **Paulo Matias**[3]

[1]Secretaria Geral de Informática – Universidade Federal de São Carlos (UFSCar)
Rod. Washington Luís km 235 – São Carlos – SP – Brazil

[2]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Avenida Trabalhador São-carlense, 400 – São Carlos – SP – Brazil

[3]Departmento de Computação – Universidade Federal de São Carlos
Rod. Washington Luís km 235 – São Carlos – SP – Brazil

pedro.oyama@ufscar.br, joueyama@usp.br, matias@ufscar.br

Abstract. *Social media has become part of our daily lives. It brought significant developments in the way we communicate, but it also raised some concerns, including privacy and censorship. In this context, this work presents a social media platform – EtherYou – that makes use of cryptographic primitives and an Ethereum smart contract to overcome these issues. Experiments were conducted to evaluate the operating costs involved. The results showed considerable values for senders, zero for receivers, and zero maintenance costs, indicating its potential in scenarios with a reduced number of content producers and a large number of consumers. The proposal offers users privacy over their data, transparency on the system behaviour and censorship resistance.*

1. Introduction

Social media use has continuously increased for years, and with this trend, not only have technological challenges emerged but also some concerns involving privacy, social control and censorship. Although users are the ones creating content, they are not its actual owners, since they do not have control over how the produced data is used by the platforms that store and manage it. The matter is aggravated by the fact that the social media market is controlled by very few groups and that they have free access to all exchanged data.

In 2008, amidst the ascension of social media, a white paper [Nakamoto 2008] was published proposing a peer-to-peer electronic cash system named Bitcoin. The paper proposed a solution to securely remove trust amongst the participants of transactions involving an electronic asset – Bitcoin – and solved the double-spend problem. Up to then, to prevent an electronic asset to be copied and spent more than once, a central trusted third party was needed. The concept that makes these new features possible is the blockchain, a distributed data structure based on cryptographic techniques and ruled by a consensus protocol. The decentralised feature of Bitcoin soon began to be seen as a solution where the requirement of trust amongst participants is undesirable, and many projects based on its code were born. Ethereum [Buterin et al. 2014] is one of them, and was conceived to be a general-purpose decentralised platform capable of executing arbitrary scripts.

In this context, it seems natural to use blockchain technology to build a social media platform, removing the trusted third party responsible for storing data and for provid-

ing its processing and distribution. Some works have been published proposing solutions in this regard, as the following ones. An instant messaging system that uses blockchain in the context of Chinese online shopping was proposed in [Yi 2019]. It aims to provide a reliable, traceable and secure messaging mechanism for both clients and shops. [Chakravorty and Rong 2017] proposes a social network based on blockchain that allows users to grant and revoke access and limit sharing while ensuring privacy, security and anonymity. Steemit[Steem 2017] uses its own blockchain to base a social media, in a system that rewards content producers and curators with tokens. Even though these systems offer solutions to the aforementioned problems in electronic communication, further analyses are needed to support their feasibility. Steemit is an exception, as it has been operating for some time, successfully overcoming costs and scalability challenges, but it was done by the expense of allowing a higher level of centralisation.[Li and Palanisamy 2019].

In this paper, we propose a decentralised social network – EtherYou – that provides a censorship-resistant environment and has cryptographic features that prevent non-authorised agents from accessing private content. Our proposition uses a smart contract deployed into the Ethereum blockchain to transparently manage the storage and distribution of data so any user can know how the processing is done. We further contribute with experiments to analyse the expenses involved in the proposed system. The expenses were assessed in terms of maintenance and charges for exchanging messages; also, how costs rise as the length of a submitted message increases, how they rise as the number of private recipients increments, and how celerity in processing a message impacts the price.

The remainder of this paper is organised as follows: section 2 shows our proposition and implementation details; section 3 describes the experiments conducted to assess the system viability; section 4 brings the conclusions.

2. Proposed System

EtherYou[1] is divided into two parts: the client software and the blockchain smart contract. The former is responsible for encapsulating the user message in a structure that provides authenticity, confidentiality (optional, as the message can be public) and integrity. The latter is responsible for storing the messages in a decentralised environment and building a timeline of messages that cannot be modified. The system uses two key pairs to provide its security features: ECDSA to use the Ethereum wallet to interact with the smart contract and Curve25519 as part of the encryption and signature mechanism. The Curve25519 public key is also used as user identification.

An overview of the system, showing the interaction between the two parts and the actors, is portrayed in Figure 1, with an example of Alice sending a private message to Bob. First, Alice uses the client software executed in her computer to write a message (1). The client software encapsulates her message in a data structure that can only be reverted to the original message by Bob. This sequence of bytes is sent to the smart contract (2) where it is processed and then stored in the blockchain. At any time later, Bob uses the client software executed in his computer to retrieve any public message or private message addressed to him (3). The client software prompts the smart contract to return the messages (4). The encapsulated message from Alice is returned (5), and the client software processes it to give Bob the original message (6).

[1] An implementation of EtherYou is available at https://github.com/pedrooyama/etheryou

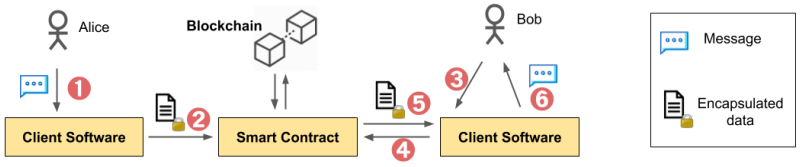

Figure 1. System Overview

2.1. The client software

As a sender, the client software receives from the user the following parameters: the message in plaintext m, which is encoded by an encoding system e, its type t, a set of n recipients' Curve25519 public keys $K^+ = \{k_1^+, k_2^+, ..., k_n^+\}$ (an empty set if it is a public message), a compression algorithm c and the user's Curve25519 private key k_u^- and Curve25519 public key k_u^+. This input is used to build the data structure that is sent to the smart contract: the message block. In the first step of this procedure some flags containing metadata needed to read the message (if the message is encrypted or not, the compression algorithm c and the encoding system e) are computed and encoded in a byte. The next step, presented in Equation 1, is to compress the message, its type, the user public key and a nonce (Unix timestamp). Notice that $\|$ is the concatenation operation.

$$m_c = Compress(nonce \parallel k_u^+ \parallel t \parallel m, c) \tag{1}$$

Then comes the encryption step. In order to send the message to multiple recipients, a trivial multi-key encryption schema is adopted, as follows. A symmetric key k_s (AES in CBC mode) is randomly generated to encrypt the compressed content and its length (Equation 2). Then, a set of ciphered versions of this key is calculated by Equation 3. If the message is not to be encrypted, this step is skipped and $m_e = m_c$ and $K^s = \{\}$. To yield authenticity, all generated data are concatenated, and the resulting bytes are hashed and then signed with the sender's private key. The signature is given by Equation 4. At this point, all elements of the message block have been created, and it can be assembled using Equation 5.

$$m_e = Encrypt(length(m_c) \parallel m_c, k_s) \tag{2}$$

$$K^s = \{k_1^s, ..., k_n^s\}, \textit{ where } k_i^s = Encrypt(k_s, k_i^+), \textit{ for } k_i^+ \in K^+ \tag{3}$$

$$Signature = Sign(Hash(flags \parallel m_e \parallel K^s), k_u^-) \tag{4}$$

$$messageBlock = flags \parallel signature \parallel m_e \parallel K^s \tag{5}$$

As a receiver, the client software fetches message blocks from the smart contract and delivers the pertinent ones to the user. If the messages are public, their content can be directly obtained by decompression. For a private message, the software tries to decrypt the symmetric key with the user's private key. If the attempt is successful, the user is a

rightful recipient, and the original message can be recovered. After decoding the content the message signature can be extracted, and authenticity and integrity can be checked.

2.2. The smart contract tailored for privacy

The smart contract[2] is responsible for storing into and retrieving messages from the blockchain. The client software can interact with it playing two distinct roles – sender and receiver – and each one of these roles has an interfacing function. The client software accesses the contract as a sender by calling the sendMessage function with the message block as a parameter. The message block is appended to an array along with calculated metadata: the block timestamp and its hash. The block hash serves as an identifier for the message so a user can reference it to reply, share or register a reaction. With the client software as a receiver, the smart contract can be accessed by the view function getMessagesFromTimestamp with a timestamp as parameter. The function returns all message blocks registered in the blockchain since the given timestamp, along with their respective metadata. Since a view function does the retrieval of message blocks, no transaction needs to be submitted, and therefore, there are no costs involved.

3. Experiments

3.1. Overview

One of the aspects to take in consideration to assess the viability of the proposed system is its cost. Since all expenses reside in sending messages, two experiments[3] were conducted in that regard: the first one investigating how the costs rise as the message length increases, and the second one analysing the expenses as the number of recipients of a private message increases. In both experiments, messages were created and sent to the smart contract to be processed, so the amount of gas used in each transaction could be registered. The smart contract was deployed in a test network using Ganache Truffle[4]. In the message block building procedure three compression algorithms – LZMA, ZLIB and BZ2 – were tested and the best one (or none) was selected. In the first experiment, different sizes of messages were used, ranging from 1 to 2000 characters. The characters were randomly chosen amongst the printable ones and the white space. In the second experiment, a fixed-length encrypted message (composed of all 96 printable characters and the white space) was built with a varying number of private recipients (from 1 to 200).

3.2. Results and Discussion

The data collected with the experiments are measured in gas, therefore they must be combined with the gas price in a given transaction to calculate the real cost. Higher gas prices tend to result in faster transaction processing. So, two arbitrary priorities (expected processing time) were defined: low (1 minute) and high (15 seconds). The gas market price for these values was estimated by perusing the Ethereum blockchain[5]. The closest values obtained were: 2 gwei for 63 seconds, and 5 gwei for 13 seconds ($1gwei = 10^{-9}Ether$). The Ether market price was valued[6] at 170.39 American dollars (USD) at the time.

[2] Available at https://github.com/pedrooyama/etheryou/blob/master/smart_contract/EtherYou.sol

[3] Available at https://github.com/pedrooyama/etheryou/tree/master/experiments

[4] https://www.trufflesuite.com/ganache

[5] Using https://etherscan.io/gasTracker at 2020.abril.09

[6] Using https://coinmarketcap.com/

The results from the experiment assessing the cost for different message length are shown in Figure 2, in gas and USD. It can be seen that the gas used increases almost linearly with the length of the message, and at a certain point (around 370 characters) the increase continues close to linear, but with a slightly less steep inclination. This change in inclination is due to the compression step, as it is not capable of reducing the size when the message is short. The line has a jagged pattern, which is due to the fact that the Ethereum Virtual Machine processes some operations using 32-byte words. Consequently, a more significant increase is experienced every 32 additional bytes. The costs of processing messages (in gas and USD) with different numbers of recipients are shown in Figure 3. The price increases linearly with the number of users addressed. The cost for including a receiver is not dependent on the length of the message, because to do so, it is only necessary to append a ciphered symmetric key to the message block.

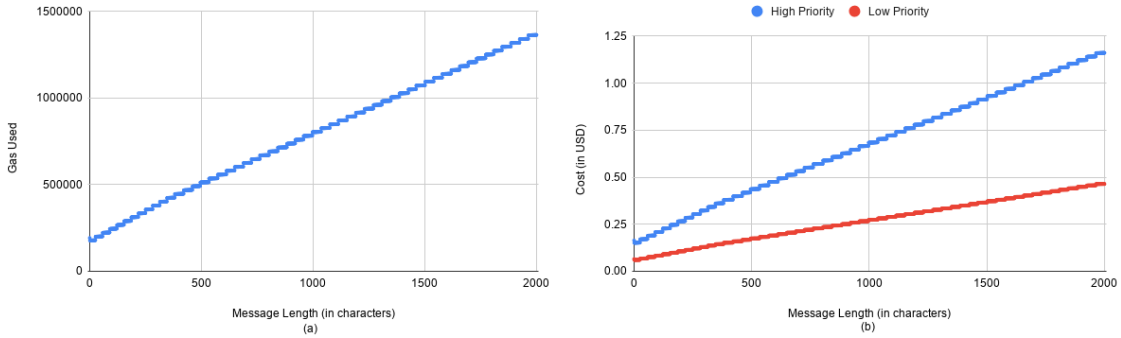

Figure 2. (a) Gas x Message Length; (b) Cost x Message Length

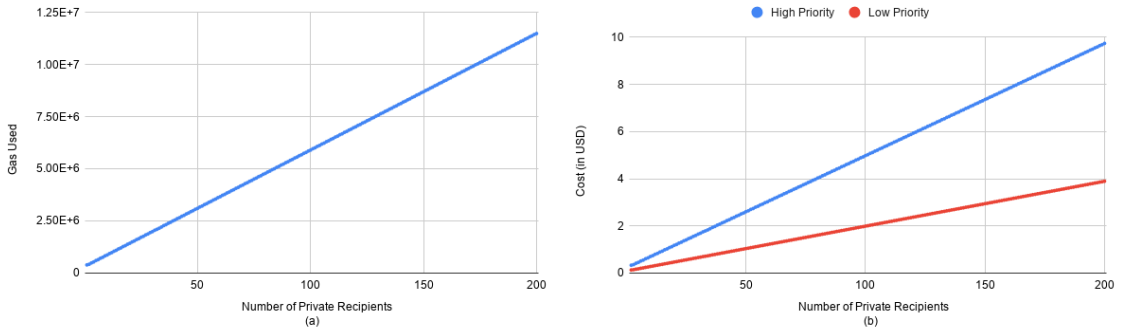

Figure 3. (a) Gas x Number of Private Recipients; (b) Cost x Number of Private Recipients

With the figures collected in both experiments, we defined equations to calculate the approximate cost of submitting a message m. Equation 6 returns the cost in gas, and Equation 7 returns the cost in USD cents (¢) by the market price evaluated at 2020.abril.09. $l(m)$ is the length of m, and n is the number of private recipients (note that $n = 0$ for a public message). $p \in \{0, 1\}$ is the priority with which m is to be to processed, with $p = 0$ being low and $p = 1$ being high. The estimated cost for the most simple message (a single-character public text that takes around 1 minute to be processed) is 6.5654¢. Each additional character costs 0.0214¢, while embedding privacy, limiting the access to a set of users, costs 1.8987¢ per user. Making a message closer to instant (around 13 seconds to be processed) multiplies the price by around 2.5.

$$Cost_{gas}(m,n) = \begin{cases} 192,029 + 628\,l(m) + 55,717\,n & \text{for } 1 \leq l(m) \leq 370 \\ 211,544 + 576\,l(m) + 55,717\,n & \text{for } l(m) > 370 \end{cases} \quad (6)$$

$$Cost_{¢}(m,n,p) = 3.4078\,Cost_{gas}(m,n)\,(1 + 1.5p)\,10^{-5} \quad (7)$$

4. Conclusions

Experiments were conducted to assess the viability of the proposed system in terms of costs. Since retrieving messages is free, and there is no cost to keep the system available, all expenses are related to message submission. It was shown that the approximate costs of sending a message are composed of a constant value, a term proportional to the message length and a term proportional to the number of private recipients, if any, as depicted in Equations 6 and 7. The price for a public message that has the same length as the maximum allowed in Twitter was estimated in 12.1973 USD cents when it is delivered in about 1 minute. Celerity also has its price, so the faster a user requires their message to be delivered, the higher the costs. For that same message to be delivered in about 13 seconds, the price is multiplied by 2.5. The system has its strength in scenarios where a few people generate public content that is consumed by many, as is the YouTube environment, or in situations when censorship is a concern.

Acknowledgments

Prof. Jó Ueyama would like to thank FAPESP (São Paulo Research Foundation) for funding his research project (Grant ID 2018/17335-9).

References

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *White paper. URL: https://github.com/ethereum/wiki/wiki/White-Paper. Accessed 10.apr.2020*.

Chakravorty, A. and Rong, C. (2017). Ushare: user controlled social media based on blockchain. In *Proceedings of the 11th international conference on ubiquitous information management and communication*, page 99. ACM.

Li, C. and Palanisamy, B. (2019). Incentivized blockchain-based social media platforms: A case study of steemit. In *Proceedings of the 10th ACM Conference on Web Science*, WebSci '19, pages 145–154, New York, NY, USA. Association for Computing Machinery.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *White paper. https://bitcoin.org/bitcoin.pdf. Accessed 10.apr.2020*.

Steem (2017). Steem: An incentivized, blockchain-based, public content platform. *White paper. URL: https://steem.io/SteemWhitePaper.pdf. Accessed 10.apr.2020*.

Yi, H. (2019). Securing instant messaging based on blockchain with machine learning. *SAFETY SCIENCE*, 120:6–13.