About a Decentralized 2FA Mechanism

Marc Jansen
Computer Science Institute
University of Applied Sciences Ruhr West
Bottrop, Germany
marc.jansen@hs-ruhrwest.de

***Abstract*— Web based security applications have become increasingly important in the past years. Especially in times of blockchain based crypto currencies, user authentication is a critical aspect for the overall security, integrity and acceptance of such systems. While blockchain technologies provide a decentralized approach, the client side still largely relies on centralized security approaches. Those centralized approaches are easier to implement, but at the same time bear the risk of usual security flaws. Therefore, this paper presents a decentralized approach for increasing the security by adding a decentralized two-factor authentication mechanism to the execution of operations.**

Keywords—blockchain, multi-factor authentication, decentralization

I. Introduction

Usually, nowadays the security of a certain application (especially blockchain based technologies, like Bitcoin [1]) that deals with security relevant data is protected not only by single passwords but multi-factor authentication mechanisms. Here, the most prominent implementations use two-factor authentications (2FA) in which a certain user of a system has to identify himself with two components, e.g., things only the user knows, possesses or something that is inseparable from the user [2].

Here, a well accepted pattern is that a user first authenticates with his username and a corresponding password and, in a second step, the second authentication mechanism is used in order to finally authenticate the user for the task in question. In common implementations, the second step of authentication needs a central repository, e.g., a central database that stores the necessary information in order to authenticate the user.

Although two-factor authentication already increases the security of a given system, it still encounters the drawback of the centralized database storing a secret only the user who wants to authenticate knows. This database could potentially be corrupted by different means. Therefore, in this paper, we describe a decentralized approach for the implementation of a two-factor authentication mechanism that does not need a centralized repository for storing certain information necessary for user authentication. First, we describe the fundamental idea of the approach, followed by different ways of implementation, depending on different business cases and functionalities that the underlying blockchain provides.

Therefore, the following parts of the paper are organized as follows: Section 2 provides an overview about the current state of the art. Afterwards, Section 3 describes a reference architecture in order to introduce some terms in the context of this paper, followed by a description of the implementation of the approach in general terms in Section 4 in order to motivate the descriptions of the algorithms. Section 5 concludes the paper by a discussion and an outlook for future work.

II. State of the Art

Blockchain based technologies are used in a large number. While digital currencies are still, by far, the most used scenario for blockchains, other scenarios have also become more and more prominent. At the same time, security of such systems needs to be improved in order to increase the broad acceptance. Since the blockchain is in itself a decentralized approach, all other related parts should preferably also be decentralized. Recent hacks in the blockchain community have shown that there is a tremendous need for securing blockchain technologies not only by a usual username / password based approach, but, additionally, to provide at least a two-factor authentication mechanism. Preferably, all parts that are related to the security of the approach should be implemented in a decentralized way. Therefore, in general, the need for decentralized user authentication mechanism, including 2FA is given.

Looking at different approaches currently implemented based on blockchain technologies, there are already implementations far beyond simple currencies, e.g., for securing intellectual property rights, that make use of blockchain technologies, e.g., OriginStamp [3]. This is, to some extent similar, because, for a decentralized 2FA mechanism, someone has to prove that he is the only one (and therefore also the first) who knows a certain secret. Furthermore, decentralized naming service also exist already, e.g., Blockstack [4] and Namecoin [5].

Additionally, there are a couple of 2FA implementations based on the Time-based One-Time Password algorithm (TOTP) [6]. Here, a number of different clients exist that allow to easily integrate 2FA at the client side, e.g., GoogleAuthenticator [7] or 1Password [8].

An approach for 2FA authentication on the Bitcoin protocol is presented in [9]. This approach uses a two party-party signature scheme compatible with ECDSA (Elliptic Curve Digital Signature Algorithm) [10]. Here, a mobile device is used in order to provide the second authentication factor. One problem that occurs with this approach is that the mobile device that generates the second factor needs to directly communicate with the PC that generates the transaction. This is not possible in general, depending on the current network configuration especially of the PC that generates the transaction, e.g., there could be communication problems in usual NAT (Network Address Translation) networks. Therefore, another solution needs to be chosen in order to allow a general approach.

Therefore, currently, an approach does not exist that allows to have a decentralized 2FA process that does not need to store the secret centrally, as it is necessary for the TOTP implementation, for example. Therefore, in this paper, we provide such an approach, that utilizes TOTP and different other approaches (depending on the business case and the functionalities of the underlying blockchain) in order to achieve the goal of a 2FA process without the central storage of the necessary secret. At the same time, we of course ensure that the secret is only known by the user who wants to get authenticated.

III. Architecture

The following two subsections provide an overview of some terms with respect to the architecture that are important to be clarified in order to provide access to the presented approach. On one hand, a description of the reference architecture of a modern blockchain based approach, along with corresponding terms, and on the other hand fundamental aspects of a TOTP architecture are introduced.

A. *Description of the reference architecture for the presented approach*

The remaining part of the section makes some assumptions related to the underlying architecture of the blockchain network and how users interact with this network. This reference architecture is shown in Figure 1.

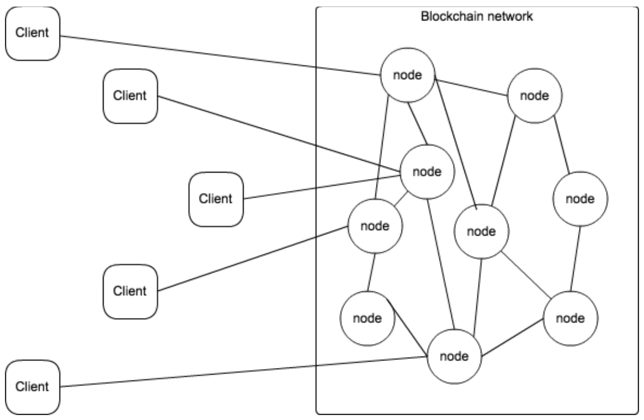

Figure 1: Reference architecture for the communication between clients and the peers of a blockchain network

As it could be seen, the clients are directly communicating with the peers of the blockchain network, which we will denote as nodes in the remaining part of the paper. It is important to state here that this reference architecture is not something special nor a limiting factor for the universality of the presented approach. It is rather a very common architecture for modern blockchain based systems.

B. *Some words on a general TOTP architecture*

The general idea of the TOTP protocol is that a client and an authentication provider agree on a secret. On the client side, this secret could be stored on different devices, e.g., the smartphone of the user, the tablet and/or his PC/Laptop. In order for the user to get authenticated, the agreed secret is used later on for the creation of the time based on-time password, e.g., a passwords that is only valid for a certain period of time. The general architecture for this is shown in figure 2.

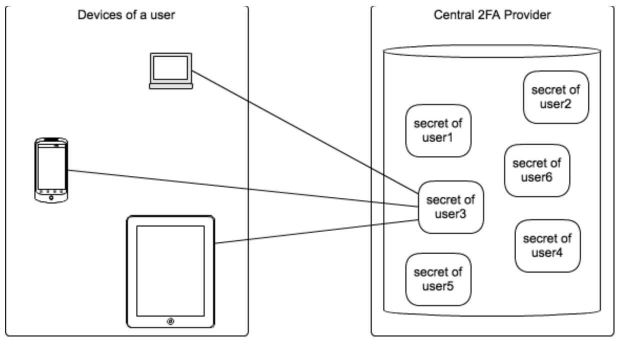

Figure 2: Architectural overview of usual TOTP implementations

Here, it is important to note that the secret shared between different devices of the user and the secret stored for that user in the centralized system are identical. At the same time, the central database is of course capable of storing secrets for other users also. As the database stores secrets for an increasing number of users, it becomes a more interesting target for potential attacker.

IV. Implementation

The following two subsections first describe the idea of the implementation and then the necessary algorithms for the implementation.

A. *Approach*

In usual 2FA implementations that make use of the TOTP protocol, the secret that is used for the creation of the one-time password is stored in some central repository. In order to overcome the necessity of the central repository, while at the same time not making the secret available to everybody, the basic idea we are following here, is to make the secret a one-time pad that is securely stored in a blockchain, allowing someone who knows the secret at a certain point in time to get authenticated towards the decentralized system. With this, in contrast with the usual TOTP implementations, not only is the generated one-time password a one-time pad as well, but also the secret that is used in order to generate the one-time password becomes a one-time pad.

Imagine a user of a certain system wants to perform a certain operation, e.g., a crypto currency transaction, then the user first needs to login to the system (by using his/her credentials). For each security related operation that the user performs in the system, the user sends the necessary information for the operation, including a TOTP based password along with the secret that is necessary in order to generate the TOTP. In order to prevent an attacker from reusing this secret and trying to get authenticated as the original user, the secret needs to be destroyed after the first usage. Additionally, by providing the secret, the user can prove that he/she already knew the secret beforehand. In order to be able to perform additional operations, also secured by the 2FA mechanism, the user has to generate a new secret with the last trusted operation he/she performed in the system, and, by this, creating a chain of trust. Here, different scenarios might be possible depending on the functionality of the underlying blockchain technology:

Scenario 1: If the underlying blockchains supports some means of attachments to an operation, the newly generated secret could be hashed and stored as an attachment to the last accepted operation.

Scenario 2: If we just have a very basic blockchain that does not allow for attachments nor sidechains or assets, the secret could be stored in the blockchain itself, secured by a Distributed Trusted Timestamp (DTT) approach [11]. Therefore, within the last trusted operation, the user generates a new secret, hashes this secret and creates a blockchain address from the hash of the secret. In order to timestamp the secret, the user transfers the minimum number of tokens to this newly created blockchain address and by this registers the secret in the system.

Scenario 3: If the underlying blockchains supports sidechains or additional assets, a security sidechain/asset could be developed in the blockchain, that stores the newly created secret. This approach also has an additional business case: Here, the security tokens could be commercialized in order to allow users to "buy" additional security (in terms of 2FA) for their operations. Basically, this implementation follows the same ideas as scenario 2 for the implementation of a DTT based approach, but without the problem of polluting the main blockchain with a large number of addresses holding just the minimal amounts of tokens, e.g., of a cryptocurrency, just for authentication, without a possibility of getting these tokens back active in the system.

Independent of which of the three scenarios are chosen, in order for the system to trust the next operation of the user, the system receives the secret sent by the user and can verify it. This verification is again a little bit different, depending on which of the above described scenarios is chosen:

Verification in scenario 1: The node that receives the operation extracts the secret of the user from the operation, hashes it and checks if the hash fits the hash of the attachment from the last accepted operation of the user.

Verification in scenarios 2 and 3: Again, the node that receives the operation extracts the secret of the user from the operation and creates the corresponding blockchain/sidechain/asset address for the hash of the secret. If the first transaction to this address came from the user who currently wants to get authenticated, the system will accept the current secret.

After accepting the secret, the node can calculate the TOTP password from the accepted secret and compare it with the TOTP password of the user that was sent within the latest operation, the system can successfully authenticate the user and accept the operation.

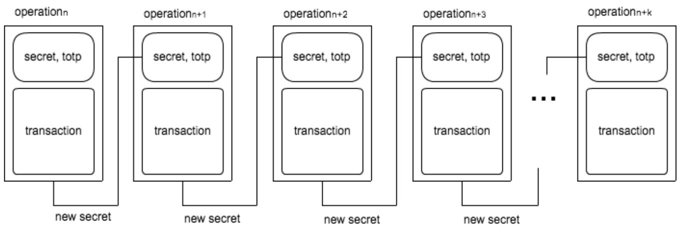

Figure 3: The chain of trust in which every last transaction creates a new secret

As shown in figure 3, by this, a chain of trust for the user will be created that allows to accept the (n+1)th-operation as long as the system has trusted the n-th operation.

B. Algorithms

Basically, the described approach needs two algorithms, one on the client side and one for the system that receives the operations of the client, referred to as a node in the following. First of all, the algorithm (figure 4) for the client side could be described as:

```
performOperation(transactionData) {
        totp = calculateTotp(secret)
        performCentralOperation(secret, totp,
        transactionData)
        secret = generateNewSecret()
        hashedSecret = hash(secret)
        blockchainAddress =
        generateBlockchainAddress(hashedSecret)
        transferToken(blockchainAddress)
}
```

Figure 4: Client side operation in case of scenarios 2 and 3

The following algorithm (figure 5) describes the functionality necessary on a node in order to trust the transaction sent by the client:

```
executeOperation(clientSecret, totp,
transactionData) {
        if (trust(clientSecret, totp)) {
        executeTransaction(transactionData)
        }
}

trust(clientSecret, totp) {
        hashedSecret = hash(clientSecret)
        blockchainAddress =
generateBlockchainAddress(hashedSecret)
        firstTransaction =
getFirstTransaction(blockchainAddress)

        if (firstTransation.sender.equals(client)) {
                generatedTotp = calculateTotp(secret)

                if (generatedTotp.equals(totp)) {
                        return true;
                }
        }

        return false;
}
```

Figure 5: Node operation

By this, the node can trust the client and perform the provided transaction. This trust is built not only on the asynchronously signed transaction data (that still needs to be checked by the above used `executeTransaction()` method, since it is not reflected in the above code), as usual in, e.g., blockchain technologies, but also on the second factor authentication via the TOTP protocol in a decentralized way.

V. CONCLUSION & OUTLOOK

The described approach currently lacks the possibility to check that a certain secret is really only used once. In order to achieve this, the tokens transferred to the newly generated blockchain address could be interpreted as semaphores. By adding a certain functionality to the above presented algorithm 2 that sends the tokens from the address back to the original sender, the characteristic of a one-time-pad could be achieved. It is ensured that the node can actually perform this operation, since it will generate the corresponding public and private key of the blockchain address along with the generation of the address itself from the hashed secret received by the client.

Also, another drawback would be solved by the above mentioned interpretation of the tokens as semaphores: the used blockchain would not be polluted by a large number of address holding the minimal number of tokens.

Possible next steps include the implementation of the proposed approach on top of an existing blockchain technology. Here, the blockchain that is used for the DTT does not necessarily need to be the same as the blockchain that the operations are performed on. Furthermore, a detailled discussion about possible drawbacks of the presented approach are necessary.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, https://Bitcoin.org/Bitcoin.pdf, last visited: 28.11.2016

[2] "How to extract data from an iCloud account with two-factor authentication activated". iphonebackupextractor.com. Retrieved 2016-06-08.

[3] https://www.originstamp.org, last visited: 08th of June 2017

[4] https://www.blockstack.org, last visited: 08th of June 2017

[5] https://www.namecoin.info, last visited: 08th of June 2017

[6] "RFC 6238 - TOTP: Time-Based One-Time Password Algorithm". Retrieved July 04, 2016.

[7] https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&hl=de, last visited: 08th of June 2017

[8] https://1password.com, last visited: 08th of June 2017

[9] C. Mann, D. Loebenberger, “Two-factor authentication for the Bitcoin protocol. In: International Journal of Information Security", 2016, p. 1--14", doi: 10.1007/s10207-016-0325-1

[10] J. López, R. Dahab, “An Overview of Elliptic Curve Cryptography,” Technical Report IC-00-10, State University of Campinas, 2000.

[11] B. Gipp, N. Meuschke, A. Gernandt, “Decentralized trusted timestamping using the crypto currency Bitcoin”, Proceedings of iConference 2015, 2015.