# Zel ID - decentralized two Factor Authentication (d2FA)

Tadeas Kmenta, 25th of September 2018, Zel Technologies GmbH, Zug

Abstract: This paper describes a way to enable a second layer of account authentication in a decentralized way, effectively negating the need for any third-party interaction.

**Historical overview**

In the beginning of authentication systems, user credentials were stored in a centralized database, mostly in a plain text form. The entire security of an account was then dependent on the security of such a third-party centralized database. It is clear that such a solution has many flaws and is a perfect candidate for a trivial replacement using a hash produced by a cryptographic function such as SHA2, instead of being stored in plain text. This vastly improves the security and potential identity theft of accounts itself, but a lot of flaws due to the centralized nature remain, such as:

1. If a weak cryptographic function is used such as MD5 or SHA1, hash collisions are very well-known and password generation that will result in the same password hash is possible in a matter of seconds
2. As with all cryptographic hashing function, there are plenty of databases (rainbow tables) already containing a pair of a common password and its hash, although this can be neglected by producing salted hashes.
3. Classic brute forcing, social engineering and other practices

Some of those flaws are nowadays solved by utilizing two factor authentication (2FA). An attacker now must know the credentials and another secret key, such as having access to a physical device, being a piece of information only the owner of identity knows.

However, this commonly used system has its biggest weakness in centralization, trust in third parties and of course the biggest weakness of it all – the user itself; the authentication system cannot replace the user nor force him to choose unique password long password and unique 2FA for each and every site, we can try to remove other weaknesses.

**Zero storing**

Thanks to the power of blockchain a truly decentralized and anonymous identity can be created. There is no need to store anything on a centralized database, but all data can be stored in a user's computer and in the blockchain itself utilizing Hierarchical Deterministic wallets (HD wallets).

A way of creating a HD wallet and user identity is to utilize user input such as username and password and perform some cryptographic function on top of them. The output is used as a seed phrase for HD wallet generation as that is the way the ZelCore platform operates, or generate a random seed phrase for the user and force the user to remember a long seed phrase, which is the way most HD wallets operates. The first address (public key) in the chain then can be a user identity and only the user possesses the private key to that address (identity) and is able to edit some data tied to that identity. With this system, there is no need to store user login credentials anywhere.

The first system used in ZelCore has a security flaw primarily in that the user itself not choosing a strong username and password and therefore collision can happen, and it may result in accidental identity

theft by other people guessing or accidentally using the same credentials. The system of generating a random seed phrase for a user has the biggest issue in the form of needing to store such a seed phrase on a secret location, as remembering a 12-24 random word phrase is just impossible. This extends to losing access to the entire wallet and troubles while porting the wallet from one device to another. This paper is proposing a solution, especially for the account creation system featured in ZelCore.

**d2FA**

In a previous step we have generated an HD wallet and we are using our first address generated as the main identity. We can now utilize the second address generated in the chain and send a specific transaction to our identity first identity address.

A specific transaction requires the following:

a. Our second address has some funds in it (Assuming it is a ZelCash address, some ZEL has to be present)
b. Generation of third address in a seed not even known to the user

Suppose that our 2nd address is funded and broadcast a following transaction to the network:

1. Sender: My 2nd address
2. Receivers: my 1st address and potentially my 2nd address containing a given UTXO change
3. Amount to send: At least the minimum amount required by the network. Usually 600 satoshis. This amount should be low so the price of generating 2FA is negligible and also should be random as further discussed in Problems section
4. Fee: Suitable for a transaction to be mined in the upcoming block. As of current ZelCash network state, a 0 fee can be used but should be avoided as further discussed in Problems section
5. OP_RETURN field: Encrypted secret word that will be used as our d2FA. The encryption method can be for example aes-256 with an encryption key of our 3rd unknown address that can even be salted

Now that we have broadcast such a transaction, it now lives on a blockchain and upon starting software which utilizes this d2FA system, a knowledge of this d2FA secret is required in order to proceed to the account itself.

This effectively mitigates the problem of a third person generating the same account. Moreover, d2FA secret can be changed by simply broadcasting a new transaction repeating the steps or, completely disabled by broadcasting a transaction with for example some hard-coded wallet software salt. From the timestamp of a transaction, we know which secret is the correct and effective one and from the character of a transaction where both addresses are owned solely by the user there is no way to broadcast this specific transaction by a third person.

**Problems**

The following issues should be taken into consideration while implementing d2FA

1. User needs to have funds in the 2nd address.
   This issue can be solved by service (wallet) providers automatically sending a negligible amount of an asset (ZEL) to users account. However, this inclines that user will create multiple accounts

just to obtain a very small portion of money. This is an unwanted scenario for the ZelCash blockchain. For other blockchains this may not be a problem but is in contradiction in flooding attack described in the following point. Therefore, it is up to the user to pay a negligible amount of ZEL to secure the account utilizing d2FA.

2. Transactions flooding attack. If an address used for identity is known to be an identity address, it can be flooded with many small transactions by third parties resulting in long transaction history. This long transaction history can slow down the service loading while the software checks all the transactions in address history to fulfill a particular transaction possess necessary checks described. Therefore, it is a good practice that while setting up a d2FA a random send amount and standard network fee is used. For an attacker, this attack is very costly, even resulting in making the victim richer and will be ineffective once victim logs back into account and simply sends new d2FA phrase. Since we are checking from the newest transaction to the oldest, an attacker would need to perform the flooding process again.
3. Forgetting the d2FA secret will occur. However, it is possible for a service to implement expiration of a secret based on number of blocks. A service may even require from a user to change the d2FA secret at specific time intervals to further extend account security. Both of those options (expiration, automatic change) should be opted for only upon user request and user should have total control over these functions. These options are also stored in OP_RETURN transaction field
4. OP_RETURN field is usually size-limited (80bytes in ZelCash case), therefore a d2FA secret needs to be short so with the expiration number of blocks, automatic change flag and other data automatically embedded into OP_RETURN. It is expected from a user to create a short d2FA secret and so the size limit should not be exceeded at any given case, however during implementation it must be taken into account and user must be prevented to set such a long secret as the transaction will get refused.
5. Availability of a service. While blockchain itself will not go anywhere, the hard-coded nodes a service is connected to might. The service should therefore be connected and communicate with many nodes in order to achieve 100% operational time and in order to verify that those multiple nodes possess the same transaction history and are indeed on a correct blockchain. This can be further extended by giving the user a choice to back up d2FA transaction into local storage.

**d2FA in a wallet creation**

Above described d2FA process generates seed phrase and the entire wallet prior checking with blockchain and thus only wallet software enhances the security. This can be solved by using a random generated phrase together with d2FA in wallet creation process itself.

Let's suppose that a "prior" HD wallet is generated from users input (eg. username and password). Wallet software then generates a random salt for a user. This random salt is not known to the user but lives in the OP_RETURN field of a transaction broadcasted to a "prior" HD wallet d2FA address encrypted with users d2FA phrase. Attaching this random salt to users input will then obtain an actual users HD wallet. This makes a strong HD wallet with simple knowledge of username, password and changeable d2FA passphrase (PIN) which contains a randomly generated salt for each and every user making the entropy of a seed used for HD wallet very high.

**Summary**

d2FA solves the account conflicts in HD wallets and further extends the wallet's security capability, even giving a changeable wallet parameter. This second authentication layer is effectively stored in a blockchain and no other trusted third party is required to authenticate into the account with d2FA.