Splintering: A Decentralized Encryption Method to Improve Password Authentication Security & Enable Trustless Wallet Decentralization

José Luis Lobo
Cryptography Engineer
Tide Foundation

Yuval Hertzog
Co-Founder
Tide Foundation

Michael Loewy
Co-Founder
Tide Foundation

Abstract— **End-user familiarity with username / password authentication continues to see it as the most common and accepted form of electronic authentication today. However, vulnerabilities in this authentication method leave organizations, critical systems and end-users exposed. The proposed solution herewith, named 'Splintering', considers a new, decentralized authentication security scheme to address some of the most significant vulnerabilities of username / password authentication. This discussion paper publishes the workings and findings of a study conducted [1] to evaluate the impact of Splintering. A sample set of 60 million LinkedIn user records from a publicly reported data breach was utilized for a comparison between the success of a dictionary attack in determining passwords from the account data when secured conventionally with hashing and salting vs that secured with Splintering. Results concluded that Splintered authentication data measured an average 14,064,094% security improvement over the conventional, centralized alternatives.**

Splintering offers a significant security boost to password authentication, simultaneously removing the need for a reliable central entity to store authentication data or perform authentication functions. The ensuing benefits are not only applicable to password authentication in its many current use cases, but also enables this ubiquitous authentication method to be introduced to new arenas that have historically shunned it – Distributed Ledger Technology (DLT), such as Blockchain.

In a practical application of Splintering, this paper demonstrates how a decentralized authentication mechanism can provide seamless interaction with a decentralized crypto wallet, via password authentication, without compromising on security. In so doing, this paper aims to demonstrate how a typical web end-user can interact with a blockchain-based application, without the typical technical and usability hurdles inhibiting blockchain's mass adoption.

Keywords—Password authentication, Key retrieval, secret sharing, decentralization, dictionary attacks.

I. Why decentralize a wallet?

A. Today's wallets and their limitations

The primary benefit of DLT is the element of 'trustlessness', i.e. I, as an end-user, don't need to trust other parties in a particular transaction in order for the expected outcome of the transaction to occur. Additionally, I maintain complete control over my assets throughout the process of executing a transaction. Essentially, the need for any central or trusted entities to have effective control or custodianship over an end-user's assets is removed. Whoever holds the private key associated with a digital asset has the ultimate and complete control over it. The accepted norm for holding private keys of digital assets is via crypto wallets.

Crypto wallets range from desktop applications, mobile applications, browser extensions, hardware devices, engraved metal [2], a piece of paper, to even subdermal chip implants [3]. In this section, the paper briefly explores the most common ones, with a focus on the major issues a decentralized wallet can improve upon or overcome. Generally speaking, the various wallet solutions trade one benefit for another – for example, security for accessibility – Thus end-users select a wallet based upon their chosen priority, having to accept the significant limitations of that wallet variety in other respects.

Due to significant limitations of current wallets, primarily the practical usability challenges they present, end-users are often forced to entrust their assets with other entities and relinquish control. In fact, the most common method for typical end-users to hold crypto assets is on a centralized exchange, in the form of a server-side web wallet. Users of the exchange relinquish control over their wallets because either their funds or their private key are held by the exchange. Should there be a breach of the exchange, the perpetrator can gain access to the user's crypto assets. With an average exchange theft resulting in $90 million [4] in stolen digital assets and over $1 Billion [5] stolen just in 2018, the need to relinquish control of crypto assets in order to trade them efficiently, not only philosophically contradicts the intended vision behind cryptocurrency and negates the cryptographic virtues of digital assets, but presents a very real threat to a large number of holders of crypto assets. This includes risks to the exchanges themselves.

In contrast, client-side wallets mitigate that risk by allowing consumer's full control of their wallet from their local device. The first drawback is that for their crypto assets to be utilized, the end-user must be available, or their local device must be always online. A more serious downside is that the user now becomes responsible for managing the security of their private keys. The relatively high knowledge and commitment required for a typical mainstream end-user to successfully navigate this responsibility poses a challenge. Particularly when humans are often the weakest link [6] in a security chain, potentially rendering the system ineffective.

Hardware wallets, also referred to as cold wallets, are physical devices that keep the private key secure in offline storage. The wallet is physically connected to a network when access to the private key is required. Although considered one of the more secure methods of storing keys, its downsides, which are derived from the physical nature of the device itself are significant – for example if a device breaks down, is lost or stolen, so is access to the associated assets. Some compensate for this using mnemonics or key phrases that can be run through an algorithm to recover a key, as a backup. These are usually held in a detachable soft copy or printed on paper, each variation carrying its own security vulnerability, either digital or physical.

The concept of Secret Sharing is also widely accepted as a backup for key recovery. It's a cryptographic method that allocates portions of a secret to a number of chosen parties. When a predetermined threshold of those parties brings their portions together, the secret may be reconstructed. This concept relies on a number of relatively trusted third parties for recovery to be successful, with the potential for those parties to lose their portion or collude at the expense of the original key holder.

The absoluteness of who maintains control over a wallet presents other challenges in the event wallets are lost or the individual or entity with sole access is unavailable. On one end of the spectrum, this presents transaction timing and scalability challenges. On the other more extreme end, the disappearance of wallet custodians can result in millions of dollars in crypto assets becoming permanently unrecoverable [7].

In contrast, centralized wallets provide more accessible and practical access to assets. However, they require end-users to rely upon third parties that hold their assets or who authenticate access to those assets to be both trustworthy and to maintain adequate security. With data breaches that expose usernames and passwords occurring with alarming frequency, centralized systems, though widely common are proving less sustainable.

This paper explores a solution where neither a central entity nor the end-user is required to hold keys, but rather a user retains control of those keys and the ability to reassemble them on the fly – A decentralized wallet.

II. Creating a decentralized wallet

A. *A deeper look at Secret Sharing*

Secret Sharing, briefly discussed above, is a well-known technique among cryptographers with Shamir's Secret Sharing [8] method being the most common. Using this method, a "Dealer" (owner) runs his secret through a function converting it into a one-of-a-kind curve. A chosen arbitrary number of n points on the curve (commonly referred to as "shares") are shared with n participants for safekeeping. Later, when called for, the secret can be recovered by combining a predetermined number of shares, t, returned by the participants at their discretion, through a function that recreates the curve and from there interpolate the original secret.

Consider the following example: Alice has a crypto wallet and wants to back up her private key. She appoints five of her trusted friends to hold the shares of her secret and assumes that they all agree to be share custodians. She establishes 3 to be the threshold number of shares to recover her key:

1. Alice runs her private key through the Secret Sharing function creating five shares to be shared.
2. Sends the different shares to each friend through a secure channel.
3. Each friend stores their share to allow Alice to reconstruct the wallet in the future.

Should Alice lose her mobile phone with the wallet app holding the key, it can be reconstructed by:

1. Alice installs the wallet app on a new phone.
2. She initiates the Restore option.
3. Notifies five friends and request the shares.
4. The friends send the shares to Alice via a secure channel.
5. Alice waits for three out of five friends to respond with the shares.

6. Once three shares are acquired, Alice can now recover the lost private key.

The process of both sharing and recovering the secret present challenges and vulnerabilities. Firstly, to a typical web user, the process would present as a significant barrier due to the complexity and cumbersome nature of the scheme. If a minimum of 3 of Alice's friends are unavailable when she needs to obtain access to her lost key, she'll be unable to do so. If 3 of Alice's 5 friends lose their shares, she'll be unable to recover her key. If 3 of the shares are exposed, then an exploiting party can reconstruct the key. If 3 of the friends collude, they can themselves obtain the key.

B. *A more practical approach to authentication using Secret Sharing*

The above shortcomings render Secret Sharing in its raw form unsuitable for a mainstream audience. This section considers an approach, which utilizes the principles of Secret Sharing together with Threshold Encryption, while simultaneously addressing their usability, trust and reliability challenges.

In this high-level example, Alice's secret is converted into 20 shares instead of 5 and her friends used to hold shares are replaced with 20 nodes on a decentralized network.

When Alice wants to create a wallet using a wallet application:

1. Alice generates a wallet account, by entering a username and password to be used for authentication.
2. A unique Private key is generated for Alice.
3. The Private key is run through a Secret Sharing function to produce twenty shares.
4. The shares are then sent to each of the nodes with an encrypted derivative of the username and password.

For Alice to retrieve the private key:

1. Alice uses the wallet application to enter her username and password.
2. An encrypted derivative of the username and password is sent to each node.
3. If the encrypted derivative of the username and password authenticates against those previously stored, the nodes return the correct shares. Otherwise they return false shares that would result in the reconstruction of an incorrect key.

While this approach creates new challenges and vulnerabilities, addressed further in the paper, with just this technique applied, decentralized authentication has been achieved. Security over traditional centralized authentication mechanisms has improved, since the private key is only ever exposed to Alice herself throughout the process. Alice no longer has to store her key locally. Additionally, with the key held securely in the network, Alice has the potential to perform transactions automatically on her behalf.

C. *Hidden partial actioning of a decentralized wallet*

In the previous example, the private key was made available to Alice momentarily in order for her to perform a transaction, representing a momentary potential vulnerability. Let's now explore how to remove the need for Alice to ever hold or reconstruct her private key.

Alice simply sends a transaction along with an encrypted derivative of her username and password to the network and the nodes will individually, partially sign the transaction. Once Alice receives the partially signed responses, they're reconstructed in her wallet into the fully signed transaction and sent to the blockchain network. Alice just performed a transaction on the blockchain with a simple username and password authentication, without holding the private key.

D. *Seamless integration with a decentralized wallet*

Since the major processing requirements of the decentralized wallet are conducted by the distributed nodes, the wallet front-end only requires very light functionality. Which in turn enables seamless integration of a decentralized wallet with any client-side experience, even a website. Alice has portable access to her wallet (account) from any device simply using a browser, without the need to install wallet software.

E. *Practical application of a decentralized wallet*

To provide further context of the advantages and possibilities of a decentralized wallet, let's first consider how it is used within the Tide Protocol [9], an open source protocol allowing for the safe storage, sharing and trading of sensitive consumer data. It does so by enabling an organization to lock sensitive consumer data in a vault and provide consumers with the only key to it. This vault can be utilized and traded with the consumer's permission and reward, when appropriate.

In a typical Tide scenario, a data "Seeker" (e.g. a marketing company) looks to acquire the consumer data of an interesting audience within another business, a "Vendor" (e.g. a medical center) where those records are encrypted with the consumer's key (i.e. the patient's key). Should the consumer consent to the request, the data is unlocked for the Seeker and the consumer and Vendor receive Tide tokens as compensation.

There are two types of data that reside in a Vendor's database; anonymized data that can be held in plain text and data deemed sensitive, e.g. personally identifiable data, which is encrypted by the consumer before being sent to the Vendor. Since the Vendor does not have the ability to decrypt the sensitive data, in the event of a data trade to a third party, a request to the associated consumers is needed every time. The consumers must be available to approve or reject the acquisition request. In a case where the consumer is offline during the transaction, they may miss out on the opportunity presented in the offer from the Seeker. Additionally, the Seeker may have to wait an impractical period of time to reach the necessary threshold of consumers for the trade to be worthwhile.

To address these issues, a system was developed, leveraging a decentralized wallet to enable a consumer to securely delegate the authority to access their data, based on a pre-approved criterion, to a system that is always online to act on their behalf. This system was named a DAT (Delegated Automated Trustee), which utilizes decentralized nodes/services called ORKs (Orchestrated Recluder of Keys).

The ORKs are an evolved concept from the "decentralized nodes" contemplated above. They store shares of consumer private keys securely using Secret Sharing. Additionally, this infrastructure not only enables the use of threshold encryption for the signing of transactions in distributed manner (threshold signature), but using the DAT, also to perform distributed decryption (threshold decryption) without the need for the owner to participate or relinquish control of their Private key.

F. Practical application of a decentralized wallet

Below is a walkthrough of the math behind the system's implementation.

ElGamal [10] cryptosystem is used for the key infrastructure. Like most asymmetric encryption schemes, the keys in ElGamal are simply large numbers: (P, Q, g, pk) for the public key and (sk) for the private key, where there is a relationship between the public and the private ($pk = g^{sk}$ (mod P)). Similarly, ElGamal encryption and decryption actions are achieved with a basic mathematical operation (within a Finite Field):

- $(c_1, c_2) = (g^r \ (mod\ P),\ m\ .\ pk^r \ (mod\ P))$ for encryption
- $m = c_2\ (c_1^{sk})^{-1}$ for decryption

To perform decryption by an external untrusted party, the private key must remain obscured. This is achieved by splitting the private key to m pieces so that the construction of all the pieces results in the private key ($sk = combine(\{y_1, y_2, ..., y_m\})$). When performing an operation on each of those pieces separately with the same constant (i.e. the ciphertext), the construction of all the results is the same as operating the private key with the constant ($c_1^{sk} = combine(\{c_1^{y1}, c_1^{y2}, ..., c_1^{ym}\})$) and Shamir's Secret Sharing allows this with the ElGamal cryptosystem. This is possible in Shamir's Secret Sharing because it uses coordinates on a polynomial for sharing the private key and the Lagrange interpolation [11] to reconstruct the secret. Furthermore, it remains secure (perfectly secure) since the polynomial is created using random coefficients, so the shares are virtually random for a naïve observer. Secondly, the shares are not smaller than the secret, rather of the same size, therefore, the level of security increases by the number of shares that are set to reconstruct it, even with m-1 shares.

The threshold decryption is performed by the ORK using the partial key share (y_i) (originally given by the consumer as a partial key to be stored) with the ciphertext (c_1) for the partial decryption ($s_i = c_1^{yi}$) and forward the result to the seeker.

Lastly, the interpolation made by the seeker to reconstruct the partial decryption must be changed from:

$$L(0) = \sum_{i=1}^{\kappa} y_i \ell_i(x) \pmod{Q}$$

to:

$$c_1^{L(0)} = \prod_{i=1} s_i^{\ell_i(x)} \pmod{P}$$

The reason being that in plain Shamir the secret is reconstructed ($L(0) = sk$), however, it is now also necessary to interpolate parts of the decryption ($c_1^{L(0)}$) and because the equality in the equation must be maintained, and $s_i = c_1^{yi}$ that is the result of rise c_1 to the power of the equation.

The Seeker ends up with c_1^{sk} and all that is left is to compute locally the product of c_2 with its inverse and the Seeker gets the plaintext m $(m = c_2\ (c_1^{L(0)})^{-1})$.

III. 'Splintering' to boost authentication security

A. Poor passwords offer poor security

So far, this paper has explored a decentralized wallet that allows users to authenticate and authorize transactions using a familiar username and password mechanism. Some significant end-user friction points have been removed in the process, making a crypto wallet more practical, accessible and secure to a mainstream audience. However, the necessity to store encrypted derivatives of passwords in a distributed network, where it must be assumed that nodes cannot be fully trusted, creates a potential vulnerability.

Passwords are notoriously easy to hack [12]. Out of convenience, they are often selected by end users on the basis that they are easy to remember. Additionally, end users often employ common passwords across multiple accounts for additional convenience, which mean if they are exposed on one Vendor, their accounts on other Vendors are potentially also compromised.

The remainder of the paper explores the susceptibility of the system to a dictionary attack and a new technique to radically improve password security over conventional centralized methods.

The technique is intended to improve security without placing any further burden on an end-user, however, it is also complementary to existing secondary security techniques, such as Second Factor Authentication.

B. *Introducing the concept of 'Splintering'*

In pursuit of this objective, a technique named 'Splintering' was developed. At a high level, it involves reducing the size of any password fragments shared with ORKs and spreads them across additional ORKs. This in order to achieve a level of fragmentation where conceptually, any individual ORK's fragment is so common that the number of potential false positive matches to password alternatives renders a dictionary attack impractical.

Although it's surprisingly common to find cases of even the largest tech companies storing passwords in their database in clear text format (Facebook [13] / Google [14]), best practice would usually ensure passwords are only stored as secured hashes. Secured hashing is a one-way operation that allows the input of a string of any length to be converted it into a string of characters that is always the same length, regardless of the input string length. There are many algorithms to hash content. SHA256 [15], designed by the National Security Agency, will return a unique 32 byte array for any given string.

For example, the hash of password "123456": "8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92"

Since a Hash is the result of a one-way operation, they cannot be reverse engineered to identify the original string. In the case of a hashed password, the password cannot be identified from its corresponding hash. However, the same original string (or password) will always result in the identical hash. So, if a bad actor is able, through whatever means, to correlate a password hash to an original password, then any other account identified with the same password hash can also be accessed with the same password with 100% certainty.

Generic passwords, such as "123456" selected by many users, only makes the matching process easier to achieve via dictionary attacks.

If, rather than storing a full password hash on each ORK, only the first 16 bits (i.e. a 16 bit 'Splinter') is stored, then only that Splinter is available for comparison and we begin to see collisions, where the Splinter could potentially apply to numerous passwords. For example, a Splinter of "8d96" could apply to both "123456" and "linked92". When an attacker attempts to employ a dictionary attack on these Splinters the result is no longer 100% certain, since different passwords share the same Splinters. If we reduce the size of the Splinter even further to 8 bits, the number of false positives becomes even greater.

If the total hash is only 8 bits long, an attacker can easily perform a brute force attack, as there are only 256 possibilities to guess a value correctly. However, consider a scenario where a user's secret recovery threshold was 32 and their 8 bit Splinters were shared with 40 different ORK nodes. Additionally, each of those Splinters were salted with a different unique identifier for each ORK node. The attacker would have to attempt not just 256 alternatives, but 256 to the power of 32. This corresponds to the same effort of performing a brute force attack [16] for a hash size of 256 bits.

This process of partial verification of the hash made by different nodes during authentication was named: Splintering.

C. *Testing the effectiveness of Splintering against previous breached data*

To validate the concept a focused study was conducted using as a test data set 60 million LinkedIn credentials [17] exposed in a previous breach along with their equivalent brute forced passwords [18].

#	Password	Password frequency	Splinter	Splinter collisions	Certainty%
1	123456	374,825	141	603,443	62.1144002
2	linkedin	58,725	132	282,432	20.79261557
3	123456789	54,712	21	288,614	18.95680736
4	password	49,868	94	285,088	17.49214278
5	12345678	31,335	239	248,601	12.60453498
.	.	.	.	.	.
16	charlie	10,493	185	232,013	4.522591407
17	linked	9,106	34	225,398	4.039964862
18	maggie	9,014	170	233,886	3.854014349
19	zzzzzzzz	8,912	193	225,653	3.949426775
20	121212	8,465	62	225,093	3.760667813
.	.	.	.	.	.
31	abcdef	6,860	190	230,004	2.982556825
32	welcome	6,823	40	229,712	2.970240997
33	777777	6,791	236	232,562	2.920081527
34	baseball	6,772	160	234,040	2.893522475
35	password1	6,750	11	236,367	2.855728592
36	buster	6,737	203	227,460	2.961839444
37	shadow	6,726	11	236,367	2.84557489
38	chocolate	6,601	116	232,879	2.834519214
39	999999	6,557	147	225,752	2.904514689
.	.	.	.	.	.
55	sophie	5,917	94	285,088	2.075499495
56	Password1	5,893	25	234,950	2.508193233
57	william	5,865	208	227,253	2.580824016
58	friends	5,847	46	224,735	2.601730928
59	computer	5,813	170	233,886	2.48539887
60	jordan	5,806	19	230,928	2.514203561
61	joshua	5,792	252	248,828	2.327712315
62	freedom	5,746	19	230,928	2.488221437
63	football	5,726	99	225,281	2.54171457
64	mother	5,710	207	228,250	2.501642935
65	samantha	5,699	197	227,537	2.504647596

#	Ranking of password frequency
Password	The actual password

Password frequency	The number times password appears across entire user base
Splinter	The first 8 bits of the hashed password
Splinter collisions	The number of times the Splinter matches a Splinter of another password
Certainty	The level of certainty that a password guess matching a Splinter also means a match to the correct password

Figure 1 - Analysis of the 65 most repeated passwords from exposed LinkedIn credentials. Common colors visually identify Splinter collisions.

First, the identical passwords were grouped, tallied and sorted by the number of repetitions, as represented in the ***Password frequency*** column. Many commonly used passwords were observed. For example, "123456" was chosen as a password by 374,825 distinct users. The top 65 most commonly used passwords represented 1.84% (1,070,882) of the total user base (57,950,593).

The passwords were hashed using SHA256, with the first 8 bits (i.e. the first 'Splinter') populated into the ***Splinter*** column. There are only 256 possible values in an 8 bit representation, ranging from 0 to 255. Meaning, with only the first 8 bit Splinter of the hash stored by an individual ORK, in place of the full hash or the plaintext password, the repetitiveness of each Splinter measures greater than the repetitiveness of each original password (***Password frequency***). That proved true for all passwords, including the most common "123456" with Splinter "141" almost doubling the potential password matches from an original 374,825 times in ***Password frequency*** to 603,443 times in ***Splinter collisions***. This occurs because different passwords share common Splinters, unlike when the entire hash is stored and a 1 for 1 match is guaranteed. This phenomenon can be immediately observed in the top 65 passwords, where for example, "password" shares the same Splinter value of "94" as "sophie".

In addition, it can be observed that the Splinter "94" is used 285,088 times, compared to 49,868 times that the password "password" is used, which means that the password only represents 17.49% of all passwords that Splinter may be applicable to. In other words, if one were to attempt to determine which users are using the password "password" and comparing its Splinter with the database of Splinters, it would get a high percentage of false positives (82.51%), because there is only a 17.49% certainty of choosing the correct password for an associated user – which is what the column ***Certainty*** represents.

This is a significant improvement over a traditional scenario where a complete hash is used, where a match results in 100% certainly. This distributed nature of Splintering makes the task of identifying user credentials more difficult due of the abundance of false positives and therefore the need to interact with multiple nodes to achieve an effective attack.

It should be noted that there is a strong correlation between the frequency of a password and the certainty when applying this technique: the lower the frequency, the lower the certainty, and the higher the security. On average, for this LinkedIn database, the level of certainty after applying Splintering for all passwords using an 8 bit Splinter measured 0.000711025452294816%. This is an improvement of 14,064,094% compared to the 100% certainty beforehand. The same rule applies for the Splinter sizes; the smaller the size, the lower the certainty.

As noted, the size of a Splinter directly impacts the difficulty of discovering the origin of a hash. The next effort revolved around identifying the optimal Splinter size. An analysis comparing various sizes were modeled in a function that produces a certainty estimation for different Splinter sizes.

$$mean\_pass = num\_pass/num\_unique$$

$$certainty = \left(mean\_pass + \frac{num\_pass - mean\_pass}{2^{bits}}\right) * 100$$

Using this formula, the variation in the of certainty based on different Splinter sizes was plotted using the passwords in the LinkedIn database.

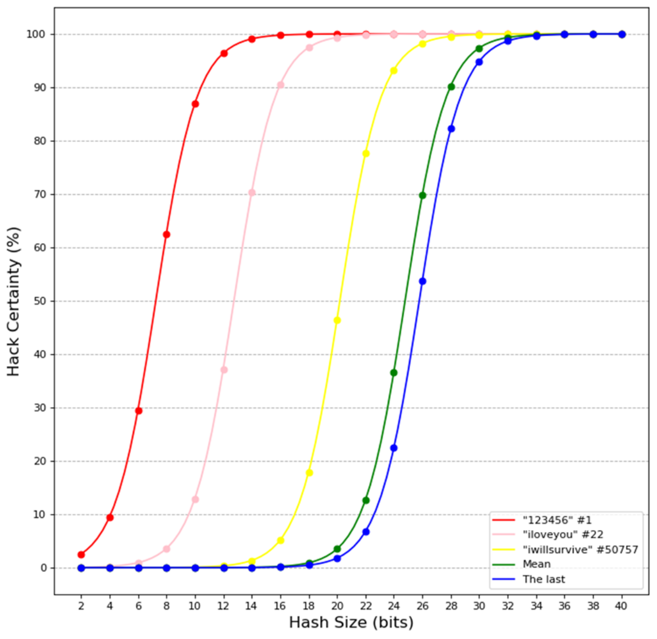

Figure 2 - Hash size vs Certainty for the LinkedIn hacked passwords

Figure 2 shows a mapped dispersion of the most common passwords from the mean and least common. The more common passwords prove more challenging to protect. For passwords that fall within the mean and below, a 16 bit Splinter is sufficient protection. However, for more common passwords, such as "iloveyou", 16 bit Splinters leaves the certainty to an undesired level. 8 bit Splinters protect virtually all passwords, other than the most common, such as “123456”, who’s Certainty is still improved by 160%.

IV. Conclusion

Although password authentication is inherently flawed and prone to various attack vectors, its prevalence as a mainstream authentication method remains high due to its ease of use and by extension its acceptance and familiarity with end users.

This paper demonstrates that concepts like decentralization can be harnessed to significantly improve password authentication security, which together with additional techniques can extend the viability of the most ubiquitous method of online authentication.

While Splintering is a breakthrough in password authentication security over traditional environments, using decentralized environments with zero-trust will likely present additional opportunities for improvement. Further study in this field may reduce the footprint in the distribution of the most common passwords, such as “123456”, reduce the number of nodes necessary to maintain the level of security offered by a 256 bit hash and perhaps even maintain the level of security I the event attackers access more nodes without reaching a threshold.

V. Additional security factors

In order to focus on the technique of Splintering and its beneficial security outcomes as a stand-alone measure, the paper does not address the impact of additional peripheral security measures that significantly bolster security even further within the Tide. However, it’s important to consider their additional impact in order to appreciate the robustness of a decentralized wallet and authentication system. For consideration a sample have been included below:

- In the Splintering example and study, a worst-case scenario was assumed, whereby an attacker knows the position of the Splinter, however, in reality, this information would be difficult to come by, significantly magnifying the complexity of matching via a brute force attack.
- To avoid malicious behavior by an ORK, the user individually salts the password with their username and the unique identifier of the ORK (such as the domain name).
- To hinder a brute-force testing of multiple passwords, the hashing function is slowed down with a key derivation function like Argon2 or PBKDF2.
- If the username and password does not match with the Splinter stored, the service does not disclose that as a failed attempt, as this information reduces the size of the search. Instead, it returns a valid but fake deterministic share based on the input such as:

  fake_share = HMAC_SHA256(ork_key, real_share || input_password)

- Each ORK throttles the user when suspicious behavior is detected (e.g. DDoS attacks) such as blocking the IP/port for an increasing period of time. To further reduce likely success of an attacker, the password Splinter size could be further reduced, which means increasing the number of hosts that must retain the Splinters. For 6 bit hashes, 43 ORK hosts are needed to allow 256 bits hash level of security.

VI. References

[1] "Splintering GitHub Repository," [Online]. Available: https://github.com/tide-foundation/decentralized-password-authentication-analysis.

[2] "Metal Plate Wallets for Bitcoin Recovery Seed Key," [Online]. Available: https://www.toughgadget.com/bitcoin-crypto-metal-recovery-seed-wallets/.

[3] "Installing a Subdermal Bitcoin Wallet Is Only for the Brave," Jamie Redman, [Online]. Available: https://news.bitcoin.com/installing-a-subdermal-bitcoin-wallet-is-only-for-the-brave/.

[4] "2018 Cryptocurrency Exchanges. User Accounts Leaks Analysis," [Online]. Available: https://www.group-ib.com/resources/threat-research/cryptocurrency-exchanges.html.

[5] "CRYPTO CRIME REPORT: Decoding Hacks, Darknet Markets, and Scams - January 2019," [Online]. Available: https://blog.chainalysis.com/2019-cryptocrime-review.

[6] "Transforming the ‘Weakest Link’ — a Human/Computer Interaction Approach to Usable and Effective Security," M A SasseS BrostoffD Weirich, [Online]. Available: https://link.springer.com/article/10.1023/A:1011902718709.

[7] "Crypto Exchange QuadrigaCX Missing $145 Mln After Death of Founder," Aaron Wood, [Online]. Available: https://cointelegraph.com/news/crypto-exchange-quadrigacx-missing-145-mln-after-death-of-founder.

[8] "Shamir's Secret Sharing," [Online]. Available: https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing.

[9] "Tide Procotol Whitepaper," [Online]. Available: https://tide.org/whitepaper.

[10] "ElGamal encryption," [Online]. Available: https://en.wikipedia.org/wiki/ElGamal_encryption.

[11] "Lagrange polynomial," [Online]. Available: https://en.wikipedia.org/wiki/Lagrange_polynomial.

[12] "Passwords are our greatest security weakness," By Lachlan McKenzie, Centrify Country Manager, ANZ, [Online]. Available:

https://www.cso.com.au/article/603494/passwords-our-greatest-security-weakness/.

[13] "Facebook Stored Millions of Passwords In Plaintext—Change Yours Now," Lily Hay Newman, [Online]. Available: https://www.wired.com/story/facebook-passwords-plaintext-change-yours/.

[14] "Google Has Stored Some Passwords in Plaintext Since 2005," Lily Hay Newman, [Online]. Available: https://www.wired.com/story/google-stored-gsuite-passwords-plaintext/.

[15] "SHA-2," [Online]. Available: https://en.wikipedia.org/wiki/SHA-2.

[16] "Brute-force attack," [Online]. Available: https://en.wikipedia.org/wiki/Brute-force_attack.

[17] "LinkedIn Breached Authentication Credentials," [Online]. Available: https://databases.today/search-nojs.php?for=linkedin.

[18] "Corresponding Password Hashes Leaked from LinkedIn Breach," [Online]. Available: https://hashes.org/leaks.php?id=68.

Please note: This paper is intended as a technical discussion paper.