Silk: Solving the security and UX tradeoff of noncustodial wallets

November 23, 2023

Overview

Wallets have suffered the "impossible" trilemma of simultaneous satisfying three properties: noncustody, security, and web2 indistinguishability. Noncustody refers to the lack of centralized control of the wallet, security refers to the minimization of attack vectors, and web2-parity-UX characterizes a use experience where the end user would not know the difference between a regular web account and the wallet. This whitepaper demonstrates that this trilemma is not actually impossible by demonstrating and constructing certain tightly coupled modules that, while independent, are only useful with each other. Each module solves a different problem related to the wallet trilemma, and overall they form a noncustodial wallet that is still secure and retains largely web2-parity UX.

Dispelling Prevalent Myths

Before describing the contributions of Silk to noncustodial wallets, it could be instructive to explain why these are actually contributions. There are no accepted standard definitions of the terms "secure", "easy to use", or even "noncustodial", so it is common for wallets to claim these features. Unfortunately, this backdrop makes solving the trilemma less exciting as it is to us, as claims from some of the hundreds of existing wallets would have the unsuspecting reader believe the trilemma has already been solved by a certain technology such as trusted execution environments (TEEs), passkeys, account abstraction, or social recovery. Here are common misconceptions organized by term

1. **Noncustodial** The common definition of a noncustodial wallet is one that does not permit a centrally governed entity to have power to spend a user's funds. Ideally, it also cannot prevent the user from transacting, but we do not consider that required for it to be noncustodial. When a centralized web account such as Apple, Google, or Twitter

has spending power, it cannot be noncustodial, even if a threshold of nodes can also access the funds. Also, Trusted execution environments are similarly custodial. Under common setups, they can spend user funds at the discretion of the cloud administrator. In circumnstances where a cloud administrator account has no access, attacks on TEEs are released relatively frequently [1], causing the degree of privacy from the hardware custodian to be questioned. In spite of claims involving nonstandard definitions of noncustodial, virtually no WaaS relying on single signon or TEEs today is noncustodial according to the previous definition that we feel is the standard connotation of the term "noncustodial." And there are no WaaS to our knowledge that do not rely on these.

2. **Security** It is difficult to find a wallet that does not advertise security, making it difficult to define "how should security be defined?" We take a stringent stance on what constitutes security:

   (a) **Resilience to uneducated user behavior** so malware, blind signing, and phishing cannot cause significant loss. Perhaps surprisingly, while phishing is easy to prevent, no popular wallets prevent malware or blind signing, with the limited exception of *some* (not most) hardware wallets and MPC wallets.

   (b) **Lack of single point of failure** if any component of the wallet has a critical security issue in its implementation, the user's funds are not at risk. This is uncommon with the sole exception being *some* (but not most) MPC-based wallets. No account abstraction wallets meet this strict standard as smart contracts are single points of failure. It should be noted that some wallets with single points of failures minimize the risk of these failures, such as hardware wallets and extensively-audited smart contract wallets.

3. **User Experience** User experience can be defined by dropoff rates, time spent on common actions, and frustrations. The following are not desireable:

   - Recovery where one has to do significant work before setting up an account, such as electing guardians, storing a seed phrase, or doing a cloud backup. It is interesting in this light that social recovery is considered a boon to web3 adoption, as it is likely more time consuming than cloud backups.
   - The need to download an app or browser extension
   - The need to import or export seed phrases to use an account across browsers or devices

Now that any misconceptions and definitions are cleared up, we can proceed in demonstrating how Silk works.

Silk Modules

1 Identity Commit-Prove with Asymmetric Burden

The core of Silk wallet recovery involves a sort of "asymmetric" commit-and-prove for user identities. Here, "asymmetric" refers to user effort. Committing is very easy; a user can commit to their email, first and last name. Users typically do this when creating accounts, and Google OAuth makes this data available. But proving does not just mean opening the commitment but rather proving that they are the owner of the account, and the owner of that name. Furthermore, proving can be done without a custodian. Proof of email can happen via zero-knowledge proofs [9] in a manner where, assuming a DNS oracle, nobody except the email provider can forge a proof. Proof of legal name can happen via NFC-enabled passports which only a government can forge, or by traditional ID verfification. Email and name are just examples of data that can be easily committed upon account creation but rigorously proven to recover an account. Security and often decentralization increase as the number of identifiers does. This asymmetric burden enabled by cryptographic identity is optimal for noncustodial account recovery: accounts should be easy to make recoverable but require a high burden of evidence to recover.

2 Hedging security assumptions of threshold decryption networks

Various networks such as Lit Protocol [6], TACo [7], and Internet Computer Protocol (ICP) [8] enable access control primitives. However, security of shared secret when collusion is undetectable is still an open problem in practice. Each network takes a different approach, either with economic incentives, trusted nodes, or trusted hardware. Silk diversifies across these security approaches by storing secret shares of a password across multiple of such networks, so that even if one is compromised, the password cannot be retrieved.

3 Cross-device authentication via Verifiable Oblivious Pseudorandom Function (VOPRF)

Oblivious pseudorandom function (OPRF) between two parties is a pseudorandom function (PRF) that can be computed, where one party learns a pseudorandom function PRF and the other learns a particular point along PRF [4]. This enables deriving high-entropy pseudo-randomness from a low-entropy but memorable value such as a password. OPRF need not be limited to two parties. Even in the two-party situation, OPRF cannot leak an input such as a password the other party, but OPRF can be efficiently done with unlimited parties with relatively simple threshold cryptography such as Shamir Secret Sharing. In a verifiable OPRF (VOPRF), the correctness of other party (or parties) contribution is verifiable. We construct an OPRF using the 2HashDH protocol [10] and a zero-knowledge discrete log equality proof so that when we decentralize the network, the contributions of nodes can be verified.

OPRF can scale with $O(1)$ computation per node. Computations take in the tens of milliseconds and can be parallelized. A decentralized threshold version adds minimal overhead since no communication between nodes is necessary. The only overhead is that the client must send HTTP requests to n nodes and perform $O(n)$ efficient local computations in the tens of milliseconds benchmarked on a 2022 13' Macbook pro.

Unlike current single signon approaches to wallet generation that involve DKG, the user's private information isn't revealed to the nodes computing an OPRF. While collusion is a significant concern in some DKG-based wallets because it can reveal the user's private key, it is not a concern when generating the share using an OPRF; the OPRF requires client input to generate the private key share.

OPRF can be used in addition to local key derivation such as Argon2 [2] or Bcrypt [**?**]

2HashDH is implemented as:

$$PRF_k(x) = H(H_c(x)^k)$$

where H is a hash function that maps a scalar to a scalar and H_2 is a hash function that maps a scalar to a point on an elliptic curve. For a verifiable OPRF, it is critical that the client does not know discrete log of H_c's output.

It can be evaluated by C, a client who only knows x and S, a "server" who only knows k.

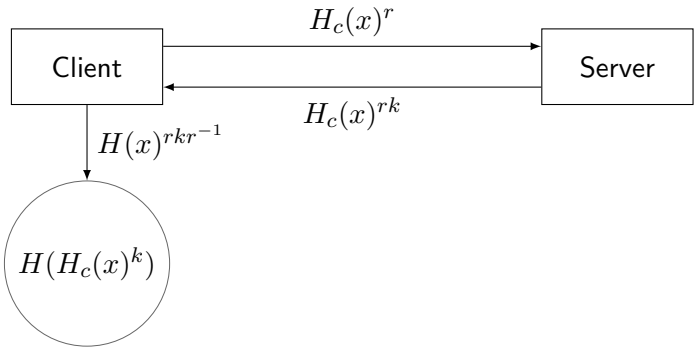

Figure 1: Jointly computing $H(H_c(x)^k)$ without the client or server discovering the other party's secret

1. C computes $h_1 = H_c(x)$, selects random element r, and sends $m = h_1(x)^r$ to S
2. S sends h_1^{rk} back to C
3. C gets h_1^k by computing $h_1^{rk*r^{-1}}$
4. C computes the final result $h_2 = H(h_1^k)$

Neither C nor S learn each other's secret value because the exponentiation occurs over a group where the discrete logarithm problem is thought to be difficult.

3.1 Computing the server response via MPC

The server may be split into many nodes i who know individual shares k_i in a Shamir Secret Sharing scheme [cite]: for any threshold-size set of nodes S, $k = \sum_{i \in Q} k_i \lambda_{i,S}(0)$
where $\lambda_{i,Q}(x)$ is the Lagrange basis for node i in set S. I.e.,
$\lambda_{i,Q}(x) = \prod_{j \in Q \setminus \{i\}} \frac{x-j}{i-j}$

A decentralized computation of the "server"'s Diffie Hellman exponentiation can be done via a method similar to threshold ElGamal. The server's exponentiation h_1^{rk} on client's input h_1^r can be constructed client-side after receiving shares of the result from a threshold-size quorum Q of MPC nodes. Node i returns share
$s_i = h_1^{rk_i\lambda_{i,S}(0)}$

The client combines these shares to compute the decentralized server's exponentiation: $h_1^{rk} = \prod_{i \in Q} s_i$.

This reconstructs the exponentiation since $k = \sum_{i \in Q} k_i \lambda_{i,S}(0)$. So $h_1^{rk} = h_1^{r \sum_{i \in Q} k_i \lambda_{i,S}(0)} = \prod_{i \in Q} s_i$

3.2 Zero-knowledge Proof of the correct computation

The server first commits to k or k_i by publishing g^k or all g^{k_i} respectively, depending whether it is a single party or not. It then computes a zero knowledge discrete log equality proof showing g^k or g^{k_i} has the same discrete log as m^k or m^{k_i} where m is the value the client sends the server in the OPRF.

4 Multifactor Authentication

$mfaShare$, a random element of the group, is the other secret share of the private key. This is used in two-party ECDSA protocol between the User Agent and an entity dubbed the Protector. We currently employ Lindell17 [**?**]. The user authenticates to the Protector via MFA. The protector is either another of the user's devices, such as a mobile phone device owned by the user for resistance to censorship, or a server for convenience. Either way, the Protector does not learn the user's private key. The mitigates risk of stolen seed phrases by splitting the private key across different devices. It further guards against blind signing, malware, and phishing by checking transactions for malicious output on both devices before signing.

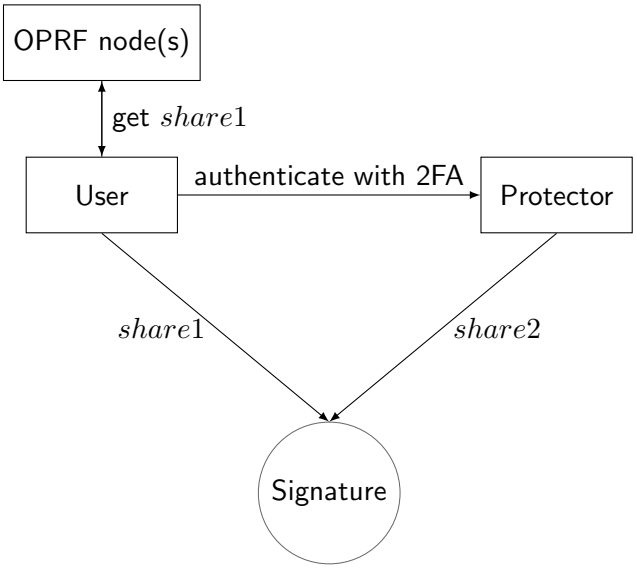

Figure 2: How signatures are constructed in the multiparty signing version. The Protector can be a server or the user's other device, as long as the 2FA is resistant to phishing. It can warn or ask for 2FA again for any risky transactions. It never learns the user's private key.

2FA itself must be phishing-resistant. To sign, both factors are needed, in the following process:

1. User enters username, password in an (untrusted) iframe to get $share1$ of OPRF(username, password).

2. User begins authentication with the Protector. The protector creates a session token which will be retrievable via a 2FA method. The 2FA must resist phishing attacks.

3. If anyone provides the session token and requests a signature share for a low-risk transaction, the Protector will provide the share. If anyone provides the session token and requests a high-risk transaction, the protector will require further authentication. This way, tokens stolen from browser storage cannot be used to act on behalf of the user for transactions categorized as high-risk. Risk analysis can be a combination of objective spending limits with more complex risk analysis algorithms.

4. User combines both shares to construct the signature

5 Clickjacking

Embedded wallets today are vulnerable to clickjacking, and we have reported these vulnerabilities and received a number of corresponding bug bounties. To mitigate these in future wallet designs, we introduce a novel method at clickjacking prevention for sensitive actions within iframes. It inolves a random challenge with information that the user must fully see in order to complete. This mandatory information includes messaging that informs the user of potential consequences of their actions. The user cannot successfully complete the random challenge without seeing the relevant messaging.

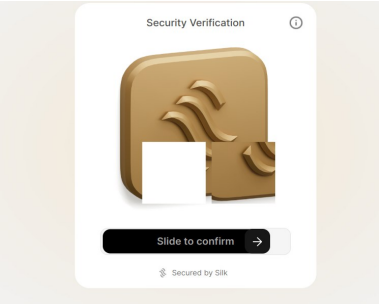

Figure 3: example of clickjacking prevention mechanism with a random challenge the site cannot trick the user into completing without showing the user recognizable graphics

Conclusion

Wallets face a challenging tradeoff: lose convenience and security, or lose the noncustodial fabric of Web3. With Silk, we show this is not a necessary choice, and introduce the first wallet that *seems* centralized due to its convenience and safeguards but is actually noncustodial, while retaining resistance to attacks that even hardware wallets do not prevent.

References

[1] van Schaik, S. et al. *SoK: SGX.Fail: How Stuff Gets eXposed.* From `https://sgx.fail/files/sgx.fail.pdf`

[2] Biryukov, A., Dinu, D., & Khovratovich, D. (201s6, March). *Argon2: new generation of memory-hard functions for password hashing and other applications.* In 2016 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 292-302). IEEE

[3] Provos, N., & Mazieres, D. (1999). *Bcrypt algorithm.* In USENIX.

[4] Casacuberta, S., Hesse, J., & Lehmann, A. (2022, June). *SoK: Oblivious pseudorandom functions.* In 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P) (pp. 625-646). IEEE. url https://eprint.iacr.org/2022/302.pdf

[5] Lindell, Y. (2017). *Fast secure two-party ECDSA signing.* In Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II 37 (pp. 613-644). Springer International Publishing.

[6] Lit Protocol. (2021). *Automate & Free the Web.* Lit Protocol. Retrieved October 19, 2022, from `https://litprotocol.com`

[7] Threshold Network (2023). *TACo Access Control.* https://docs.threshold.network/app-development/threshold-access-control-tac

[8] The DFINITY Team (2022). *The Internet Computer for Geeks.* https://internetcomputer.org/whitepaper.pdf

[9] Divide-By-0, saleel, & curryrasul. Proof of email Retrieved from https://github.com/zkemail (2022).

[10] Jarecki, S., Kiayias, A., & Krawczyk, H. (2014). *Round-optimal password-protected secret sharing and T-PAKE in the password-only model.* In Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and

Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014, Proceedings, Part II 20 (pp. 233-253). Springer Berlin Heidelberg.

[11] John R. Douceur. (2002). *The Sybil Attack*. Nakamoto Institute. In: Peer-to-Peer Systems. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260

[12] Goldwasser, S., Micali, S., & Rackoff, C. (1985). *The knowledge complexity of interactive proof-systems.* In Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali (pp. 203-225).

[13] Vitalik Buterin. (2021). *Why we need wide adoption of social recovery wallets*. Vitalik Buterin's Website. Retrieved June 29, 2022, from `https://vitalik.ca/general/2021/01/11/recovery.html`