KeyShield: Leakage-and-Loss-Resilient Private Key Protection for Web3

Ziyang Ji[1], Jie Zhang[1,*], Yuji Dong[2], Ka Lok Man[1], Steven Guan[1] and Mucheol Kim[3]

[1]*School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, China*
[2]*School of Internet of Things, Xi'an Jiaotong-Liverpool University, Suzhou, China*
[3]*School of Computer Science and Engineering, Chung-Ang University, South Korea*
E-mail: Ziyang.Ji18@student.xjtlu.edu.cn; Jie.Zhang01@xjtlu.edu.cn; Yuji.Dong02@xjtlu.edu.cn; Ka.Man@xjtlu.edu.cn; Steven.Guan@xjtlu.edu.cn; mucheol.kim@gmail.com
[*]*Corresponding Author*

Received 14 December 2025; Accepted 06 January 2026

Abstract

Effective management of private keys is crucial to ensure the security and ownership of users' data and digital assets in the Web3 environment. However, existing solutions often fail to adequately address private key management from the user's perspective. Private key leakage and loss incidents occur frequently, resulting in significant losses of digital assets. Moreover, the conventional approach of revoking both the private and public keys after a leakage or loss accident is inconvenient in Web3, where the public key serves as the user's wallet address or digital identity.

To tackle the issue of user-side private key management in Web3, this paper presents KeyShield which is a leakage-and-loss-resilient private key protection scheme. KeyShield divides the user's private key into three shares, securely stored across a primary device and a secondary device owned by the user, and a third storage module owned by the user or a semi-trusted service provider. For daily use of the private key, the user only needs to connect

Journal of Web Engineering, Vol. 25_1, 67–102.
doi: 10.13052/jwe1540-9589.2515

the primary and secondary devices. In the event of a leakage or loss, such as device theft or attack, an update process will be triggered to update the three shares, immediately invalidating the leaked or lost share while causing no changes to the public key. As a demonstration of KeyShield, we developed KeyShieldECC accessible on both Android and iOS platforms for managing Elliptic Curve Cryptography (ECC) private keys. The testing results show that for a 256-bit ECC private key, the daily use only needs 0.05 seconds and update needs 0.25 to 0.3 seconds on an ordinary smart phone.

Keywords: Crypto wallet, key management, threshold cryptography, proxy re-encryption, Web3.

1 Introduction

The third era of the Internet (Web3) [1] stresses people's property rights, that is, the ability to own a piece of the Internet. This is realized via ensuring users' ownership and control of their cryptographic private keys. For example, in the most mature Web3 application, i.e., cryptocurrency, the ownership of digital currency is represented by an address derived from the public key, and all transactions have to be signed using the corresponding private key. Only the person keeping the private key can claim the corresponding digital currency ownership. Similarly, assets ownership in other Web3 applications such as NFT (Non-Fungible Token) [2] and DeFi (Decentralized Finance) [3] are also represented via public keys and determined by the corresponding private keys. Therefore, the security and safety of private keys are vitally important in Web3.

Private key management has been studied for many years since the birth of public-key cryptography. However, it is still not well addressed on the **user side** in the real world as majority of users have very limited knowledge about private keys. Private key leakage and loss still happen frequently and have caused serious damages. For example, in June 2022, hackers attacked the Harmony Horizon Bridge which is one of the most famous cross-chain bridges. This attack resulted in the stolen of 100 million dollars worth of cryptocurrencies due to the leakage of private keys [4]. In August 2022, CloudSEK's Attack Surface Monitoring Platform published a report which uncovered more than 3,200 applications leaking Twitter API keys [5], among which 230 serious leakages can be used to fully take over user's Twitter account.

Specifically in Web3, private key management is a hot topic which has been attracting significant attention from both industry and academia. A variety of decentralized crypto wallets [6–10] for private key management are developed. There are also a number of centralized crypto wallets (aka crypto exchanges) [11, 12] which involve a trusted server for key management on behalf of Web3 users. In addition to the above application-level approaches, private key management for Web3 is also studied in academic papers, including schemes based on secret sharing [13–17], threshold digital signature [18, 19], and proxy digital signature [20].

1.1 Problem Statement

Although existing approaches are helpful to alleviate **private key leakage and loss problems** in Web3, no method can absolutely guarantee private key from been leaked or lost. Therefore, **key update and retrieval** are as important as key protection in Web3, which however are not well addressed by the aforementioned approaches. In most existing private key management solutions mentioned above, both the private and public keys will be revoked once the private key is leaked or lost. This kind of after-leakage or after-loss measure is inconvenient in Web3 scenarios where the private key determines a certain amount of digital assets and the public key stands for user's digital identity. For example, in cryptocurrency systems, before revoking a public-private key pair, the balance should be transferred from the current address (represented by the public key) to a new address of the user. There are two limitations: first, a certain amount of transaction fees will be charged, and much worse, attacker who compromised the private key probably may have already transferred the balance to his/her address before the user discovers the attack. Even though some schemes [13, 14, 18, 19] imply the possibility of updating private key shares without changing the public key, none of them presents how to update the shares, and all of them are impractical for daily use due to heavy communicating and computing costs.

1.2 Our Work

To comprehensively tackle the issue of user-side private key management in Web3, this paper proposes KeyShield which provides leakage-and-loss-resilient private key protection for Web3 users. The basic idea behind KeyShield is originated from leakage-resilient public-key cryptography (LR-PKC) which protects long-term private keys against side-channel attacks. While typical constructions of LR-PKC schemes split the stored

secret into two components that leak independently and update them immediately after use, KeyShield is designed in a different way to better suit the requirements of real-world Web3 application scenario. It splits Web3 users' private key into three shares using an information-theoretical secure secret sharing scheme. The first two shares are kept by the user in two devices for daily use of the private key requested by Web3 applications. The last share can be kept either by the user himself/herself or or by a semi-trusted service provider for updating the private key shares in case of leakage or loss incidents. Once any of the three shares is in danger of leakage or is lost, for example, user device is attacked or stolen, an update process will be invoked to update the three shares, immediately invalidating the potentially leaked share or implicitly retrieving the lost share, and meanwhile causing no changes to the public key. Existing schemes such as [13, 14, 18, 19] can also be extended to include an update process as KeyShield, however, they requires the participation of a certain number of distributed servers for daily use of the private key, which means users' private key can be fully controlled by a certain number of servers, and the communication and computation cost will be too high for daily use.

To demonstrate KeyShield, a private key management system named KeyShieldECC is designed and realized for the management of Elliptic Curve Cryptography (ECC) private keys. Specifically, following the framework of KeyShield, we first present a basic KeyShieldECC which uses Shamir's threshold secret sharing scheme to split the user's ECC private keys into three shares. Among them, one is kept by user's primary device which is a smart phone, one is stored into user's secondary device which is an Near Field Communication (NFC) card, and the last share is kept by the KeyShieldECC server. For daily use of the private key, the user just needs to put the NFC card close to the smart phone. In case the smart phone is stolen or attacked, or the NFC card is lost, or at the very worst, the server is attacked, an update process will be initiated to update all the three shares. To further enhance the security of private key, we then proposed an advanced KeyShieldECC which introduces an encryption scheme constructed based on Proxy Re-Encryption (RPE) [21, 22] and a distributed signing protocol for Elliptic Curve Schnorr (EC-Schnorr) digital signature algorithm. The encryption scheme encrypts the third share before uploading it to the server, which eliminates the needs for secure channels between the user and the server. The distributed signing protocol outputs EC-Schnorr signature of the private key on a message without explicitly reconstructing the private key.

Further, we developed the Beta version of the basic KeyShieldECC Application (App) for both the Android and iOS platforms. A suit of experiments have been carried out to test the functionality and performance, in particular, in various cases of leakage or loss incidents. For a 256-bit ECC private key, the average time of initial split is 0.2 seconds, daily use is 0.05 seconds, and update is 0.25 to 0.3 seconds on an ordinary smart phone (Operating System: Android 13, CPU: MediaTek Dimensity 8200-Ultra). The storage cost is 43.9MB for the App client in the mobile phone. These figures show that KeyShieldECC Beta is efficient and lightweight for ordinary devices in the real world.

1.3 Organization

The remaining parts are organized as follows. Section 2 reviews representative key management products and theoretical schemes in Web3. Section 3 presents cryptographic preliminaries and definitions. Section 4 presents the detailed design of KeyShield. Section 5, 6 and 7 presents the detailed design, evaluation, and implementation of KeyShieldECC respectively. Finally, Section 8 concludes this paper and presents future work.

2 Related Work

This section reviews application-level private key management approaches (i.e., crypto wallets) and recent academic research achievements on private key protection or recovery for Web3.

2.1 Crypto Wallets

In the context of Web3, the system used to manage users' private keys is known as crypto wallet as cryptocurrency is the most widely known and mature Web3 application. Although it is named as 'wallet', it does not store cryptocurrencies. Instead, it stores users' public and private keys and provides an interface to manage cryptocurrencies or to interact with DApps in Web3. This section reviews some popular and representative crypto wallets.

The most famous crypto wallet is Bitcoin Core Wallet [6] for Bitcoin [23] blockchain. It uses AES-256-CBC to encrypt a user's private keys under a random master key. The master key is also encrypted via AES-256-CBC under a key derived from the user's password. Another famous crypto wallet is MetaMask [7] which handles key management for Ethereum [24] users

Table 1 Comparison of private key protection products in Web3

Scheme	Control on Private Key	Supported Web3 Applications	After-Leakage-or-Loss Measure
Bitcoin Core Wallet [6]	User	Only cryptocurrencies	Public key changed
MetaMask [7]	User	DApps	Public key changed
Cryptonator [8]	User	Only cryptocurrencies	Public key changed
KeepKey [9]	User	Only cryptocurrencies	Public key changed
OneKey [10]	User	Only cryptocurrencies	Public key changed
Coinbase [11]	User and trusted server	Only cryptocurrencies	Public key changed
GateHub [12]	User and trusted server	Only cryptocurrencies	Public key changed
KeyShield	User	DApps	Public key unchanged

and connects the users to the blockchain. It generates users' private key from a seed phrase (known as Secret Backup Phrase) and encrypts the seed phrase and all accounts data under a key derived from users' password. In addition, there are a large number of all-in-one crypto wallets that support a wide types of cryptocurrencies, such as software crypto wallets Cryptonator [8], and hardware crypto wallets KeepKey [9] and OneKey [10], etc.

All of the above crypto wallets are known as decentralized wallets in which only users themselves hold their private keys. There are also a few numbers of centralized crypto wallets (aka crypto exchanges) which involve a trusted server for key management. For example, in Coinbase [11], GateHub [12], etc., users are not in full control of their private keys. These crypto exchanges keep the private keys for users and implement strong security measures to prevent unauthorized access.

A comparison of the crypto wallets and exchanges mentioned above is provided in Table 1. In summary, crypto wallets are more secure than crypto exchanges as users hold their private keys and thus can save their private keys offline, while crypto exchanges are more user-friendly. However, in both approaches, compromising the private key of a user means taking full control of the account and cryptocurrencies of the user. The KeyShield approach proposed in this paper can address this problem.

2.2 Private Key Management in Web3

Most existing schemes focus on protecting the confidentiality of private key. Li and You [13] proposed a consortium blockchain wallet using two secret sharing schemes of different thresholds to split the private key and a consortium blockchain to store the shares. Xiong et al. [14] proposed a secret sharing based scheme in blockchains to protect private keys with key

recovery. Soltani et al. [15] roughly described how to use secret sharing to realize key backup and recovery. Zheng et al. [16] used the Generative Adversarial Network (GAN) to realize secret sharing for keys but did not explain who kept the shares. Above schemes can be used as either key backup solution or key management solution, but both have problems. If they are used as a key backup solution, that is, the original private key is still kept and used by the users, then attacks to users' device can lead to the leakage of whole private key, and thereby the updating process has no difference as crypto wallets in Section 2.1. However, if they are used as a key management solution replacing the original private key storage at user side, then daily use of the private key will require the honest participation of at least a threshold number of online servers, which leads to high communication and computation cost. Moreover, in both application patterns, a combination of threshold-number servers fully controls the private key of users.

Threshold and proxy digital signature are also regarded as a promising approach to private key protection in Web3. Boneh et al. [18] proposed a threshold signature wallet scheme. Dikshit and Singh [19] designed a weighted threshold ECDSA scheme. Wei et al. [20] proposed an online wallet keeping the encrypted private key and participating in the signature generation process. The above schemes are designed for specific digital signature algorithms; therefore, their applicability is limited. Another shortcoming is that the daily use of private key requires the honest participation of one or threshold-number of online servers. Besides, in [18, 19] a combination of threshold-number of servers can fully control users' private key.

In addition to confidentiality of private keys, some schemes focus on the backup and recovery of private key after lost. Dai et al. [25] designed a cryptocurrency recovery scheme based on hidden assisting relationships. Instead of recovering the lost private key, their scheme transfers the asset from the old address to the new address, which means the change of public key. Soltani et al. [15] roughly described how to use secret sharing to realize key backup and recovery, but did not provide details about how and where to store the shares. Singh et al. [17] proposed a key recovery scheme through the use of personal security questions and secret sharing. Above schemes are useful for the private key lost problem but cannot address the leakage issue.

3 Preliminaries and Definition

This section reviews cryptographic foundations. Symbols used throughout this paper are explained in Table 2.

Table 2 Notation

Symbol	Meaning
F_p	A finite field with prime order p.
$\mathbf{E}$	An elliptic curve defined over F_p.
E	The elliptic curve group consists of all points on $\mathbf{E}$ plus the point $\mathcal{O}$ at infinity.
G	A subgroup of E with a prime order q and a generator P.
q	The order of G.
P	The generator of G.
Z_q	The set of integers $\{1, \ldots, q-1\}$.
sk, pk	The private and public keys.
(t, n)	A threshold secret sharing scheme with n shareholders and threshold number t.
$\hat{e}$	A bilinear map.
G^{T}	A multiplicative group generated by $\hat{e}(P, P)$.
$H_1()$	A hash function mapping a string of arbitrary length $\{0,1\}^*$ to Z_q^*
$H_2()$	A hash function mapping elements in G^{T} to Z_q^*.

3.1 Mathematical Foundations

3.1.1 Elliptic curve groups

An elliptic curve group is a mathematical structure that consists of a set of points defined by an elliptic curve equation and an operation defined on these points. Its definition is reviewed as follows.

Definition 1 (Elliptic Curve Groups) *Let* $\mathbf{E}$ *be an elliptic curve defined over a finite field* F_p *of prime order* p*. The elliptic curve group* E *is the set of all points on* $\mathbf{E}$ *plus the point* $\mathcal{O}$ *at infinity along with the point addition operation* $+$*.*

3.1.2 Bilinear map

A bilinear map is a mathematical operation that takes two elements from two algebraic structures and produces a result in a third structure [26]. Its definition is reviewed as the following.

Definition 2 (Bilinear Map) *Let* G_1, G_2*, be two additive groups of the same large prime order* q*,* G^T *be a multiplicative group of order* q*, and* P_1, P_2 *be the generator of* G_1 *and* G_2 *respectively. A map* $\hat{e}$*:* $G_1 \times G_2 \rightarrow G^T$ *is a bilinear map if it satisfies the following properties:*

- $\hat{e}(aP_1, bP_2) = \hat{e}(P_1, P_2)^{ab}$ *for any* $a, b \in Z_q$*;*
- $\hat{e}(P_1, P_2)$ *is a generator of* G^T*.*
- *There is a polynomial time algorithm to compute* $\hat{e}(Q, R)$ *for any* $Q \in G_1, R \in G_2$*.*

G_1 and G_2 can be the same group or two distinct groups. In this paper, we treat them as the same group and let them be a subgroup of an elliptic curve group E defined in Definition 1 with a generator P. That is, the bilinear map used in our design is $\hat{e} : G \times G \rightarrow G^{\mathrm{T}}$ where G^{T} is a multiplicative group generated by $\hat{e}(P, P)$.

3.1.3 Cryptographic hardness assumptions

The security of our scheme and its underlying cryptographic primitives are based on the hardness of solving the following problems.

Definition 3 (ECDL Problem) *Fix an elliptic curve group G with a prime order q. Let P denote the generator of G. The Elliptic Curve Discrete Logarithm (ECDL) problem in G is to find $x \in Z_q$ for an element $X \in G$ chosen uniformly at random such that $X = xP$, denoted by $\mathbf{ECDL}(P, X)$.*

Definition 4 (WDH Problem) *Let $\hat{e} : G \times G \rightarrow G^T$ be a bilinear map, P be a generator of G, and q be the order of G. The Weil Diffie-Hellman (WDH) problem is to output 1 if $W = \hat{e}(P, P)^{abc}$ and 0 otherwise on input (P, aP, bP, cP, W), where (a, b, c) and W are chosen uniformly at random from Z_q^3 and G^T respectively.*

3.2 Threshold Secret Sharing

An information-theoretical secure (t, n)-threshold secret sharing scheme splits a secret into n shares such that the secret can be reconstructed by combining any t or more shares, while a combination of any less than t shares can leak no information about the secret. The famous Shamir's (t, n) secret sharing scheme [27] is reviewed as follows.

Let sk be the secret, q be a large prime which is bigger than both sk and n, and $a_1, \ldots, a_{t-1}$ are randomly chosen from Z_q. The (t, n) secret sharing scheme of Shamir consists of the following algorithms.

- $Share(sk) \rightarrow (sk_1, \ldots, sk_n)$: takes sk as input and outputs n shares: $sk_1 = f(1), \ldots, sk_n = f(n)$ where
$$f(x) = sk + a_1 x + \cdots + a_{t-1} x^{t-1} \mod q$$
- $Reconstruct(sk_1, \ldots, sk_t) \rightarrow sk$: takes any t shares ($sk_1, \ldots, sk_t$ without loss of generality) as input and outputs
$$sk = \sum_{i=1}^{t} sk_i \prod_{j=1, j \neq i}^{t} \frac{i}{i - j} \mod q.$$

3.3 Digital Signatures

3.3.1 Digital signature schemes

A digital signature scheme is defined as a tuple of PPT algorithms $(Gen, Sign, Verify)$ such that:

- The Gen algorithm takes a security parameter $\mathbf{k}$ as input and outputs a pair of private and public keys: $(sk, pk) \leftarrow Gen(1^{\mathbf{k}})$
- The $Sign$ algorithm takes a private key sk and a message m as input and outputs a signature $\sigma \leftarrow Sign(sk, m)$
- The $Verify$ algorithm takes a message m, a signature σ, and a public key pk as input, and outputs a bit b indicating whether the signature is valid or invalid: $b \leftarrow Verify(pk, m, \sigma)$

A digital signature scheme should satisfy the following two requirements:

- Correctness. For any $(sk, pk) \leftarrow Gen(1^{\mathbf{k}})$, any message m, if $\sigma \leftarrow Sign(sk, m)$, then $1 \leftarrow Verify(pk, m, \sigma)$.
- Security. Without the private key, it is hard for any PPT adversary to forge a valid signature on a new message that can pass the signature verification.

3.3.2 Threshold Schnorr signature

The Schnorr signature algorithm is promising digital signature algorithm in both current Internet and the Web3. One of its variant, the Edwards-Curve Digital Signature Algorithm (EdDSA) [28], has been included by the National Institute of Standards and Technology (NIST) in Federal Information Processing Standard (FIPS) in 2019. Below we review the (t, n) threshold Schnorr signature scheme $(Keygen, Dsign, Verify)$ over elliptic curve group G [29].

- The $Keygen$ algorithm can be designed in centralized manner with a dealer or decentralized manner without a dealer. The centralized $Keygen$ where the dealer is Web3 user suits our scenario. It takes a private key sk as input and outputs n shares $(sk_1, \ldots, sk_n)$ such that sk can be reconstructed with t or more shares.
- The $Dsign$ protocol is run collaboratively by $\geq t$ honest participants:
  1. Each participant computes a random nonce share k_i and $K_i = k_i \cdot P$, and then publish K_i to all other participants and the dealer.
  2. After receive t or more K_i, a party can compute $K = k \cdot P$ where k is not computed by any party. Then the party can compute and

submit to the dealer the signature share:

$$s_i = k_i - e \cdot sk_i$$

where $e = H_1(m||K)$ and m is the message.

3. After receive t or more K_i and S_i, the dealer can compute

$$K = k \cdot P,$$
$$e = H_1(m||K),$$
$$s = k - e \cdot sk \mod q,$$
$$\sigma = (s, e)$$

where k, sk are not computed explicit.

- The $Verify$ algorithm takes message m, signature $\sigma = (s, e)$ and public key pk as input, computes

$$K_v = s \cdot P + e \cdot pk$$
$$e_v = H_1(m||K_v)$$

and output 1 indicating "accept" if

$$e_v = e.$$

4 KeyShield

Assume Alice is a Web3 user who owns a (or more) pair of public and private keys, denoted by (pk, sk). The public key is published in the Web3 network. The private key is known only by Alice to prove and guarantee her ownership to her identity and digital assists, usually through digital signature and public-key encryption. It is vital but challenging that Alice as a non-expert of Cryptography can guarantee the security (not be leaked) and safety (not be lost) of her private key, particularly in a hostile network environment where attackers have strong incentive to steal private keys to gain financial reward. This section presents KeyShield which provides a framework of designing and developing private key protection schemes for ordinary Web3 users.

4.1 System Model

4.1.1 System overview

The architecture of KeyShield and its relationship with DApps in Web3 are shown in Fig. 1. As shown in the left part of Fig. 1, KeyShield involves three separated devices:

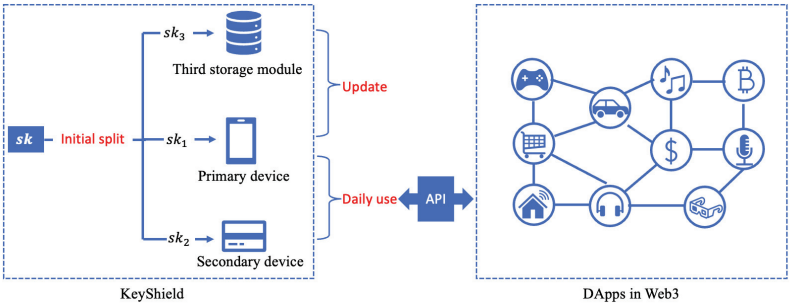


Figure 1 System model.

- Primary device: This device is owned by the user. It maintains one private key share and interacts with DApp for daily use of the private key. In the real world, this device can be user's smart phone.
- Secondary device: This device is also owned by the user. It keeps one private key share and interacts with the primary device for daily use of the private key. For example, it can be an NFC card or a smart bracelet of the user.
- Third storage module: This module can be managed by the user or a semi-trusted service provider. It is responsible for the management of the last share of users' private key. It is use for update only and is not involved in daily use of the private key.

The original private key should be destroyed after the three shares are distributed. For daily use of the private key requested by DApps in Web3, users only need to connect the primary and secondary devices. Once any one of the three shares is in danger of leakage or is lost, an update process will be executed to update all shares.

4.1.2 Threats and assumptions

KeyShield assumes the user is honest but devices could be attacked, damaged, or lost, which can lead to the leakage or loss of the private key share stored there.

It also assumes human user is able to judge whether attacks are happening when they are using the device. This assumption is reasonable in the real world thanks to various intrusion detection approaches. The most serious and common attacks usually are launched when the user is not using the system, for example, during the midnight.

Finally, it assumes secure (authenticated and confidential) channels can be established among the three kinds of devices. This can be realized by

human-assisted Out-Of-Band (OOB) channels or Transport Layer Security (TLS) in the real world.

4.1.3 Design goals

Under the above threats and assumptions, KeyShield aims to achieve the following five goals related to security or performance:

- **Goal 1: Leakage resilience**. Leakage of one share should not influence the confidentiality of the private key.
- **Goal 2: Loss resilience**. Loss of one share should not influence the integrity of the private key.
- **Goal 3: Preserved ownership**. Only the user himself/herself has fully control on his/her private key.
- **Goal 4: Efficiency of daily use**. The user should be able to reconstruct the private key directly or implicitly within a reasonable time.
- **Goal 5: Broad applicability**. The framework should be able to be applied in various Web3 application scenarios, in particular, it should not limit the adoption of cryptography algorithms and implementations.

4.2 Core Algorithms

KeyShield manages and protects the user's private key through three algorithms based on information-theoretical secure $(2, 3)$-threshold secret sharing:

- $InitialSplit(sk) \rightarrow (sk_1, sk_2, sk_3)$: This algorithm is used to split the user's private key. It takes a private key sk as input, outputs three shares (sk_1, sk_2, sk_3), and stores them into a primary device, a secondary device, and a third storage module. It is executed on the primary devices and involves the secondary device and the third storage module for receiving and storing sk_2 and sk_3.
- $DailyUse(sk_1, sk_2, m) \rightarrow f(sk, m)$: This algorithm takes two shares and a message m as input, and outputs the private key (when m is null), a digital signature (when f is a digital signature algorithm), or a decryption (when f is a public-key decryption algorithm). It is executed on the primary device and involves the secondary devices.
- $Update(sk_i, sk_j) \rightarrow (sk'_1, sk'_2, sk'_3)$ for $1 \leq i, j \leq 3$ and $i \neq j$: This algorithm is used to update the private key shares. It takes any two shares as input and outputs three new shares. It is executed on the primary devices and involves the secondary device and the third storage module.

4.3 Analysis

Based on the security of information-theoretical secure $(2, 3)$-threshold secret sharing schemes, the KeyShield framework achieves the five design goals:

- **Leakage resilience**. The $InitialSplit$ algorithm splits the private key into three shares kept in three different places, and the $Update$ function invalidates any share leaked to attackers, therefore, attackers cannot get threshold number of related shares even in a continuous leaking scenario, which means attackers cannot leak any information of the private key.
- **Loss resilience**. The $Update$ algorithm outputs three new shares of the original private key, thus any share lost is replaced by a new one.
- **Preserved ownership**. The user owners at least a threshold number of shares and can reconstruct the private key either implicitly or explicitly. Even though the third share may be kept a semi-trusted server, the full ownership on the private key is preserved only by the user as the third share can leak no information about the private key.
- **Efficiency of daily use**. The user can run the $DailyUse$ algorithm locally among the primary and secondary devices. This avoids high communication cost over the internet and therefore can achieve high efficiency.
- **Broad applicability**. KeyShield acts on the private key and its shares. It neither makes any changes to nor specifies the use of the digital signature or public-key encryption algorithms, and thus it can support various public-key settings.

5 KeyShieldECC

ECC-based schemes are widely used in real-world applications. In these schemes, the private key is an element in $Z_q = [1, q - 1]$, and the public key is a point in ECC group. This section presents KeyShieldECC as a case study of KeyShield to manage private keys in Z_q. We first introduce the basic KeyShieldECC, and then describe an advanced version which further enhances the security of private key.

5.1 Basic KeyShieldECC

The basic KeyShieldECC instantiates the primary devices as a smart phone of the user, the secondary device as an NFC card owned by the user, and

the third storage module as a server managed by a semi-structured service provider.

5.1.1 Initial split

The user first registers an account through the KeyShieldECC App client using his or her phone number. After login the system, the user inputs an ECC private key $sk \in Z_q$ to be protected. KeyShieldECC then executes the $InitialSplit(sk)$ algorithm which takes the sk as input and outputs three shares (sk_1, sk_2, sk_3) through the following steps:

1. The KeyShieldECC client on the smart phone:
   (1) Generate a random value $r \in Z_q$ to construct the following equation:
   $$f(x) = sk + r \cdot x \mod q.$$
   (2) Compute three shares as follows:
   $$sk_1 = sk + r \mod q,$$
   $$sk_2 = sk + 2r \mod q,$$
   $$sk_3 = sk + 3r \mod q.$$
   (3) Save sk_1 in the smart phone and store sk_2 into the NFC card.
   (4) Send sk_3 to the server via secure channels.
   (5) Destroy sk and r.
2. The KeyShieldECC server receives and securely stores sk_3.

5.1.2 Daily use

When the user needs sk for a computing task (such as generating a signature for message m, or decrypting a message m) requested by Web3 applications, he/she puts the NFC card close to the mobile phone and then receives the computing result in KeyShieldECC client by executing the $DailyUse(sk_1, sk_2, m)$ algorithm:

$$f(sk, m) = g(g_1(sk_1, m), g_2(sk_2, m)).$$

The function f can be configured as digital signature generation or public key decryption, and functions g, g_1, g_2 together construct a threshold signature or threshold decryption scheme in which the private key is implied but not recovered. Specifically, when m is null:

$$sk_1 \leftarrow g_1(sk_1, null),$$
$$sk_2 \leftarrow g_2(sk_2, null),$$

and

$$sk \leftarrow (f(sk, null) = g(sk_1, sk_1)).$$

Notably, the server need not be connected for daily use, and network connection is also not needed for using the private key in implied manner.

5.1.3 Update

There are various situations that the private key shares need be updated. We classify them into three update cases and introduce the secure update process for each of them:

Case 1 $Update\_1(sk_1, sk_3)$: the NFC card is lost or broken. In this case, sk_2 is not available. KeyShieldECC updates all shares through the following steps:

1. The user initiates the Update Case 1 in the client, which will send an update request to the server.
2. The server generates a verification code and sends to the user's phone number.
3. The user inputs the verification code in the client. If the code is valid, the client will receive sk_3 via secure channel from the server and then initiate the $Update\_1(sk_1, sk_3)$ algorithm to output updated shares:
   (1) Generate a random value $r' \in Z_q$.
   (2) Compute three updated shares:

$$\begin{aligned} sk_1' &= (3sk_1 - sk_3) \cdot 2^{-1} + r' \mod q, \\ sk_2' &= (3sk_1 - sk_3) \cdot 2^{-1} + 2r' \mod q, \\ sk_3' &= (3sk_1 - sk_3) \cdot 2^{-1} + 3r' \mod q. \end{aligned}$$

   (3) Updates sk_1 into sk_1' in the mobile phone, and save sk_2' in a new NFC card.
   (4) Send sk_3' to the server via secure channels.
   (5) Destroy r' and erase all values produced during above computation.
4. The server receives sk_3' and uses it to update the storage of sk_3.

Case 2 $Update\_2(sk_2, sk_3)$: the mobile phone is lost, broken or attacked. In this case, sk_1 is lost and/or in danger of leakage. The user uses the KeyShieldECC client in a different mobile phone to request secure update service.

1. The user initiates the Update Case 2 in the client, which will send an update request to the server.
2. The server generates a verification code and sends to the user's phone number.
3. The user inputs the verification code in the client. If the code is valid, the client will receive sk_3 via secure channel from the server and then initiate the $Update\_2(sk_2, sk_3)$ algorithm:
   (1) Generate a random value $r' \in Z_q$.
   (2) Compute three updated shares:

$$\begin{aligned} sk_1' &= 3sk_2 - 2sk_3 + r' \mod q, \\ sk_2' &= 3sk_2 - 2sk_3 + 2r' \mod q, \\ sk_3' &= 3sk_2 - 2sk_3 + 3r' \mod q. \end{aligned}$$

   (3) Save sk_1' in the new mobile phone, and send sk_2' to the NFC card to update the storage of sk_2.
   (4) Send sk_3' to the server via secure channels.
   (5) Destroy r' and erase all values produced during above computation.
4. The server receives sk_3' and uses it to update the storage of sk_3.

Case 3 $Update\_3(sk_1, sk_2)$: the server is attacked. In this case, sk_3 should be invalidated. The server pushes an update request to the user. The user who accepts the update request puts the NFC card close to the phone and initiates the $Update\_3(sk_1, sk_2)$ algorithm as follows:

1. The client:
   (1) Generate a random value $r' \in Z_q$.
   (2) Compute three updated shares:

$$\begin{aligned} sk_1' &= (2sk_1 - sk_2) \cdot 2^{-1} + r' \mod q, \\ sk_2' &= (2sk_1 - sk_2) \cdot 2^{-1} + 2r' \mod q, \\ sk_3' &= (2sk_1 - sk_2) \cdot 2^{-1} + 3r' \mod q. \end{aligned}$$

   (3) Update sk_1 into sk_1' in the mobile phone, and send sk_2' to the NFC card to update sk_2.
   (4) Send sk_3' to the server via secure channel.
2. The server receives sk_3' and uses it to update the storage of sk_3.

5.2 Advanced KeyShieldECC

The advanced KeyShieldECC instantiates the primary devices as a smart phone of the user, the secondary device as an smart bracelet worn by the user, and the third storage module as a server managed by a semi-structured service provider. In this version, the secondary device should have not only storing ability like NFC card but also computing abilities to execute some cryptographic computations.

5.2.1 Initial split

As in the basic KeyShieldECC, the user needs to register and login the system. After successfully login the system, the user inputs a private key $sk \in Z_q$ for protection. The $AdInitialSplit(sk)$ algorithm splits sk and outputs $(sk_1, uk_1, sk_2, uk_2, c)$ through the following steps:

1. The client:
   (1) Generate a random value $r \in Z_q$ to construct the following equation:
   $$f(x) = sk + r \cdot x \mod q.$$
   (2) Compute the three shares as follows:
   $$\begin{aligned} sk_1 &= sk + r \mod q, \\ sk_2 &= sk + 2r \mod q, \\ sk_3 &= sk + 3r \mod q. \end{aligned}$$
   (3) Compute two update keys for sk_1 and sk_2:
   $$\begin{aligned} uk_1 &= sk_1 \cdot sk^{-1} \cdot P, \\ uk_2 &= sk_2 \cdot sk^{-1} \cdot P. \end{aligned}$$
   (4) Encrypt sk_3:
   $$c = (sk \cdot v \cdot P, sk_3 \cdot H_2(\hat{e}(P, P)^v)) = (\alpha, \beta)$$
   where $H_2 : G^{\mathrm{T}} \to Z_q^*$ is a secure hash function and $v \in Z_q$.
   (5) Save (sk_1, uk_1) in the mobile phone and (sk_2, uk_2) in the smart bracelet.
   (6) Send c to the server via authenticated channels.
   (7) Destroy sk, r and v.
2. The server receives and stores c.

5.2.2 Daily use: Distributed signing

Considering that KeyShieldECC client receives a request of generating a signature for a message m from Web3 DApp. KeyShieldECC client executes the following $DistributedSigning(m, sk_1, sk_2)$ protocol among the smart phone and smart bracelet:

1. The smart phone generates a random $k_1 \in Z_q$, computes

$$K_1 = k_1 \cdot P$$

   and then sends K_1 to the smart bracelet.
2. The smart bracelet generates a random $k_2 \in Z_q$, computes

$$\begin{aligned} K_2 &= k_2 \cdot P, \\ K &= 2K_2 - K_1, \\ e &= H_1(m \| K), \\ s_2 &= k_2 - sk_2 \cdot e \mod q \end{aligned}$$

   and then sends (K_2, s_2) to the smart phone.
3. The smart phone computes

$$\begin{aligned} K &= 2K_2 - K_1, \\ e &= H_1(m \| K), \\ s_1 &= k_1 - sk_1 \cdot e \mod q \\ s &= 2s_2 - s_1 \mod q \end{aligned}$$

   and outputs $\sigma = (s, e)$ to DApp.

The output $\sigma = (k_x, s)$ is a valid signature for m under sk since

$$\begin{aligned} K_v &= s \cdot P + e \cdot pk \\ &= (2s_2 - s_1)P + e \cdot pk \\ &= (2(k_2 - sk_2 \cdot e) - (k_1 - sk_1 \cdot e))P + e \cdot pk \\ &= ((2k_2 - k_1) - (2sk_2 - sk_1)e)P + e \cdot pk \\ &= ((2k_2 - k_1) - sk \cdot e)P + e \cdot pk \\ &= (2k_2 - k_1)P - e \cdot pk + e \cdot pk \\ &= K, \end{aligned}$$

thus

$$e_v = H(m \| K_v) = H(m \| K) = e.$$

5.2.3 Update

The update procedure has three cases as in the basic KeyShieldECC.

Case 1 $AdUpdate\_1(sk_1, uk_1, c)$: When the smart bracelet is lost, damaged, or attacked, that is, sk_2 is lost or in danger of leakage. The user can invoke the update procedure as in Case 1 of Section 5.1.3. Again, authenticated but non-confidential channels are sufficient for communications among the server and the client.

1. The user initiates the Update Case 1 in the client on the smart phone, which will send an update request to the server.
2. The server generates a verification code and sends to the user's phone number.
3. The user inputs the verification code in the client. If the code is valid, the server response the client with $c = (\alpha, \beta)$.
4. When receive c, the smart phone
   (1) Compute sk_3 from
   $$sk_3 = \frac{\beta}{H_2(\hat{e}(\alpha, uk_1)^{\frac{1}{sk_1}})}$$
   where $H_2 : G^{\mathrm{T}} \to Z_q^*$ is a secure hash function.
   (2) Generate random values $r', v' \in Z_q$
   (3) Compute
   $$\begin{aligned} sk_1' &= (3sk_1 - sk_3) \cdot 2^{-1} + r' \mod q \\ sk_2' &= (3sk_1 - sk_3) \cdot 2^{-1} + 2r' \mod q \\ sk_3' &= (3sk_1 - sk_3) \cdot 2^{-1} + 3r' \mod q \\ uk_1' &= 2sk_1' \cdot (3sk_1 - sk_3)^{-1} \mod q \\ uk_2' &= 2sk_2' \cdot (3sk_1 - sk_3)^{-1} \mod q \\ c' &= (v' \cdot pk, sk_3' \cdot H_2(\hat{e}(P, P)^{k'})) = (\alpha', \beta') \end{aligned}$$
   (4) Update (sk_1, uk_1) into (sk_1', uk_1') in the mobile phone, and save (sk_2', uk_2') to a new smart bracelet.
   (5) Send c' to the server via authenticated channel.
   (6) Destroy r', v' and erase all values produced during above computation.
5. The server receives c' and uses it to update the storage of c.

Case 2 $AdUpdate\_2(sk_2, uk_2, c)$**:** When the smart phone is attacked, stolen, or lost, which means sk_1 is lost or in danger of leakage, the user can invoke the update procedure as in Case 2 of Section 5.1.3.

1. The user initiates the Update Case 2 in the client on a new smart phone, which will send an update request to the KeyShield server.
2. The server generates a verification code and sends to the user's phone number.
3. The user inputs the verification code in the client. If the code is valid, the server will response with $c = (\alpha, \beta)$ via authenticated channel.
4. When receive c, the new smart phone
   (1) Compute sk_3 from
   $$sk_3 = \frac{\beta}{H_2(\hat{e}(\alpha, uk_2)^{\frac{1}{sk_2}})}$$
   where $H_2 : G^{\mathrm{T}} \to Z_q^*$ is a secure hash function.
   (2) Generate random values $r', v' \in Z_q$.
   (3) Compute
   $$\begin{aligned} sk_1' &= 3sk_2 - 2sk_3 + r' \mod q, \\ sk_2' &= 3sk_2 - 2sk_3 + 2r' \mod q, \\ sk_3' &= 3sk_2 - 2sk_3 + 3r' \mod q. \\ uk_1' &= sk_1' \cdot (3sk_2 - 2sk_3)^{-1} \cdot P \\ uk_2' &= sk_2' \cdot (3sk_2 - 2sk_3)^{-1} \cdot P \\ c' &= (v' \cdot pk, sk_3' \cdot H_2(\hat{e}(P, P)^{k'})) = (\alpha', \beta') \end{aligned}$$
   (4) Save (sk_1', uk_1') in the new mobile phone, and send (sk_2', uk_2') to the smart bracelet to update (sk_2, uk_2).
   (5) Send c' to the server via authenticated channel.
   (6) Destroy r', v' and erase all values produced during above computation.
5. The server receives c' and uses it to update the storage of c.

Case 3 $AdUpdate\_3(sk_1, sk_2)$: When the server loses c_3 due to attacks or system failure, it pushes an update request to the user. The user who accepts the update request puts the NFC card close to the phone and initiates the $AdUpdate\_3(sk_1, sk_2)$ algorithm as follows:

1. The client:
   (1) Generate random values $r', v' \in Z_q$.
   (2) Compute

$$\begin{aligned} sk_1' &= (2sk_1 - sk_2) \cdot 2^{-1} + r' \mod q \\ sk_2' &= (2sk_1 - sk_2) \cdot 2^{-1} + 2r' \mod q \\ sk_3' &= (2sk_1 - sk_2) \cdot 2^{-1} + 3r' \mod q \\ uk_1' &= 2sk_1' \cdot (2sk_1 - sk_2)^{-1} \cdot P \mod q \\ uk_2' &= 2sk_2' \cdot (2sk_1 - sk_2)^{-1} \cdot P \mod q \\ c' &= (v' \cdot pk, sk_3' \cdot H_2(\hat{e}(P,P)^{v'})) = (\alpha', \beta') \end{aligned}$$

   (3) Update (sk_1, uk_1) into $(sk_1', uk_1)'$ in the mobile phone, and send (sk_2', uk_2') to the smart bracelet to update the storage of (sk_2, uk_2).
   (4) Send c' to the server via authenticated channel.
   (5) Destroy r', v' and erase all values produced during above computation.
2. The server receives c' and uses it to update the storage of c.

6 Evaluation

This section presents the cost evaluation and security analysis for KeyShieldECC.

6.1 Cost Evaluation

We evaluate the computing cost of the basic and advanced KeyShieldECC through time-consuming operations, including exponentiation in G and G^{T}, denoted by $\mathcal{E}$ and $\mathcal{E}^{\mathrm{T}}$, and bilinear pairing from G to G^{T}, denoted by $\mathcal{P}$.

The communicating cost is evaluated through the channels used for transmitting messages. Denote the local channel assisted by human user between smart phone and NFC card (or smart bracelet) by $\mathcal{M}_{\mathcal{L}}$, the authenticated but non-confidential channel between smart phone and server by $\mathcal{M}_{\mathcal{A}}$, and the secure channel between smart phone and server by $\mathcal{M}_{\mathcal{S}}$. Among these three kinds of channel, the communication cost $\mathcal{M}_{\mathcal{L}} < \mathcal{M}_{\mathcal{A}} < \mathcal{M}_{\mathcal{S}}$.

The results are listed in Table 3 and 4. Notably, the costs of the Daily Use function of basic KeyShieldECC is the smallest, involving 0 time-consuming operations and 0 remote communication. The costs of Daily Use in advanced

Table 3 Evaluation of cost in basic KeyShieldECC

Functionality	Algorithm / Protocol	Computing Cost	Communication Cost
Initial Split	$InitialSplit$	0	$\mathcal{M}_{\mathcal{L}} + \mathcal{M}_{\mathcal{S}}$
Daily Use	$DailyUse$	0	$\mathcal{M}_{\mathcal{L}}$
Update Case 1	$Update\_1$	0	$\mathcal{M}_{\mathcal{L}} + 2\mathcal{M}_{\mathcal{S}}$
Update Case 2	$Update\_2$	0	$2\mathcal{M}_{\mathcal{L}} + 2\mathcal{M}_{\mathcal{S}}$
Update Case 3	$Update\_3$	0	$2\mathcal{M}_{\mathcal{L}} + \mathcal{M}_{\mathcal{S}}$

Table 4 Evaluation of cost in advanced KeyShieldECC

Functionality	Algorithm/Protocol	Computing Cost	Communicating Cost
Initial Split	$AdInitialSplit$	$2\mathcal{E} + \mathcal{E}^{\mathcal{T}}$	$\mathcal{M}_{\mathcal{L}} + \mathcal{M}_{\mathcal{A}}$
Daily Use	$DistributedSigning$	$2\mathcal{E}$	$2\mathcal{M}_{\mathcal{L}}$
Update Case 1	$AdUpdate\_1$	$2\mathcal{E} + \mathcal{P} + 2\mathcal{E}^{\mathcal{T}}$	$\mathcal{M}_{\mathcal{L}} + 2\mathcal{M}_{\mathcal{A}}$
Update Case 2	$AdUpdate\_2$	$2\mathcal{M}_{\mathcal{L}} + 2\mathcal{M}_{\mathcal{S}}$	$2\mathcal{M}_{\mathcal{L}} + 2\mathcal{M}_{\mathcal{A}}$
Update Case 3	$AdUpdate\_3$	$2\mathcal{M}_{\mathcal{L}} + \mathcal{M}_{\mathcal{S}}$	$2\mathcal{M}_{\mathcal{L}} + \mathcal{M}_{\mathcal{A}}$

KeyShieldECC is also not high, involving only 2 exponentiation in G and 0 remote communication. These figures show that both the basic and advanced KeyShieldECC will be efficient for daily use.

6.2 Security Evaluation

We first prove the security of the distributed signing protocol introduced in the advanced KeyShieldECC, then analyze how the basic and advanced KeyShieldECC achieve the security goals specified in KeyShield.

Section 4.1.3 presents five goals and the first three are related to the security of private key. Below we analyze how these goals are achieved in the basic and advanced KeyShieldECC:

- **Goal 1: Leakage resilience**. First, the $InitialSplit$ and $AdInitialSplit$ algorithms split the private key using a $(2, 3)$ secret sharing scheme, which means compromising one share cannot leak any information about the private key. Second, the $Update$ and $AdUpdate$ functions invalidate the share leaked to attackers, therefore, attackers cannot get threshold number of related shares even in a continuous leaking scenario, which means attackers cannot leak any information of the private key.
- **Goal 2: Loss resilience**. The $Update$ and $AdUpdate$ functions output three new shares of the original private key, thus any share lost is replaced by a new one and becomes be useless anymore.

- **Goal 3: Preserved ownership**. The user owners a threshold number of shares and can reconstruct the private key either implicitly or explicitly. The server keeps only one share of the private key, so it can leak no information about the private key. Therefore, the user has full control on the private key.

7 Implementation and Testing

7.1 Development and Execution Environments

We have developed the Beta version (Fig. 2) of basic KeyShieldECC for both Android and iOS platforms using Java and Uniapp programming languages. Details of the development specification are listed in Table 5.

To test the functionality and performance of KeyShieldECC Beta, we install its App client in an Android mobile phone specified in Table 6.

7.2 Testing and Results

The storage requirement for the App client in the mobile phone is 43.9 MB which shows acceptable storage costs given that an ordinary smart phone usually has 256 GB or larger storage.

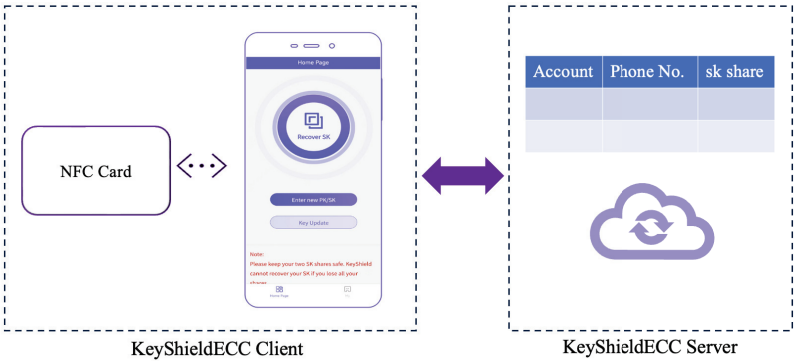


Figure 2 KeyShieldECC Beta system.

Table 5 Development specification

Parameter	Specification
Programming language	Back end: Java; Front end: Uniapp; Database: Mysql 5.7
Development environment	Back end: Java Development Kit 1.8; Front end: HBuilder X v3.8.12
ECC curve **E**	FIPS approved standard curve P-256
q	0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551

Table 6 Execution environment

Item	Specification
Operating System	Android 13
CPU	MediaTek Dimensity 8200-Ultra
Memory	16GB
Storage	512GB

To test the functionalities and performance, we specify five test cases and run the App 10 times to acquire average results:

- Initial Split: The user inputs a private key in the KeyShield App client in the smart phone, then connects the NFC card to the smart phone, and finally initiates the $InitialSplit$ function to split the private key into three shares and store them into the mobile phone, the NFC card, and the server.
- Daily Use: The user connects the NFC card to the smart phone and initiates the $DailyUse$ function to reconstruct the private key.
- Update Case 1: The user's NFC card is lost or broken, thus sk_2 is lost and/or in danger of leakage. The user connects a new NFC card to the mobile phone and initiates the $Update\_1$ function to output three new shares and store them into the mobile phone, the new NFC card, and the server. The previous shares in the smart phone and the server are overwritten by the new shares.
- Update Case 2: The user's smart phone is lost, broken or attacked, thus sk_1 is lost and/or in danger of leakage. The user installs the KeyShieldECC client in a new mobile phone, connects the NFC card to the new mobile phone and initiates the $Update\_2$ function to output three new shares and store them into the new mobile phone, the NFC card, and the server. The previous shares in the NFC card and the server are overwritten by the new shares.
- Update Case 3: The server is attacked or broken, thus sk_3 is lost and/or in danger of leakage. After recovered from the incident, the server pushes an update request to the user. The user who accepts the request connects the NFC card to the smart phone, and initiates the $Update\_3$ function to output three new shares and store them to the mobile phone, NFC card, and the server. The previous shares if still exist will be overwritten by the new shares.

Note that in all of the above cases except the Daily Use case, the user needs to authenticate to the server through SMG verification code using his/her phone number.

Table 7 Execution time in second

Functionality	Execution time
Initial Split	0.20
Daily Use	0.05
Update Case 1	0.26
Update Case 2	0.30
Update Case 3	0.25

The testing results are summarized in Table 7. The time of Daily Use function is merely 0.05 seconds, which is consistent with the evaluation result in Section 6.1.

7.3 Analysis

Based on the testing results, we now conclude that KeyShieldECC has achieved the two performance-related goals listed in Section 4.1.3.

- **Goal 4: Efficiency of daily use**. The experimental results show that the Daily Use function is lightweight as it can be executed within 0.05 seconds for an ordinary mobile phone.
- **Goal 5: Broad applicability**. In the Daily Use test case, the original ECC private key is output thus KeyShieldECC can either be directly used by ordinary users to manage their private keys or work as a key management module by providing an Application Program Interface (API) to DApps.

8 Conclusion

This paper has successfully addressed the critical issues of private key leakage and loss in the context of Web3. It introduced KeyShield, a practical and user-oriented framework specifically designed to be resilient against private key leakage and loss. Additionally, the paper presented KeyShieldECC as a case study of KeyShield, demonstrating its effectiveness in protecting ECC private keys. The thorough testing and analysis conducted on KeyShieldECC confirm its ability to meet the intended design goals.

Moving forward, several avenues for future work have been identified. Firstly, building upon the foundation established by KeyShield, the development of key management frameworks to address Web3's supervision problems is planned. Secondly, the current version of KeyShieldECC focuses

on protecting ECC private keys of curve P-256. To broaden its applicability, continuous upgrades will be implemented to support the protection of additional types of keys. Lastly, expanding the support for various types of secondary storage equipment, such as smart bands, will be explored to provide diverse options for storing the second local share of the private key.

Acknowledgement

This work is partially supported by the XJTLU AI University Research Centre and Jiangsu Province Engineering Research Centre of Data Science and Cognitive Computation at XJTLU and SIP AI innovation platform (YZCXPT2022103). Also, it is partially funded by the Suzhou Municipal Key Laboratory for Intelligent Virtual Engineering (SZS2022004). This work is also supported by 2024 Suzhou Innovation Consortium Construction Project (LHT202406), Suzhou Data Innovation Application Laboratory and SIP High level innovation platform.

References

[1] Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, B. Wen, Q. Li, and Y.-C. Hu, "Make web3.0 connected," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 2965–2981, 2022.

[2] Q. Wang, R. Li, Q. Wang, and S. Chen, "Non-fungible token (nft): Overview, evaluation, opportunities and challenges," *arXiv preprint arXiv:2105.07447*, 2021.

[3] W. Li, J. Bu, X. Li, and X. Chen, "Security analysis of defi: Vulnerabilities, attacks and advances," in *2022 IEEE International Conference on Blockchain (Blockchain)*, pp. 488–493, 2022.

[4] hackernoon, "Harmony's horizon bridge attack: How $100m was siphoned off by a hacker," https://hackernoon.com/harmonys-horizon-bridge-attack-how-dollar100m-was-siphoned-by-a-hacker, 2022.

[5] CloudSEK, "How leaked twitter api keys can be used to build a bot army," https://cloudsek.com/whitepapers_reports/how-leaked-twitter-api-keys-can-be-used-to-build-a-bot-army/, 2022.

[6] "Bitcoin core," https://bitcoin.org/en/bitcoin-core/, 2025.

[7] "Metamask," https://metamask.io, 2025.

[8] "Cryptonator," https://www.cryptonator.com, 2025.

[9] "Keepkey," https://shapeshift.com/keepkey, 2025.

[10] "Onekey," https://www.onekey.so, 2025.
[11] "Coinbase exchange: Institutional trading platform," https://exchange.coinbase.com, 2025.
[12] "Gatehub," https://gatehub.net, 2025.
[13] G. Li and L. You, "A consortium blockchain wallet scheme based on dual-threshold key sharing," *Symmetry*, vol. 13, no. 8, p. 1444, 2021.
[14] F. Xiong, R. Xiao, W. Ren, R. Zheng, and J. Jiang, "A key protection scheme based on secret sharing for blockchain-based construction supply chain system," *IEEE access*, vol. 7, pp. 126773–126786, 2019.
[15] R. Soltani, U. T. Nguyen, and A. An, "Practical key recovery model for self-sovereign identity based digital wallets," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 320–325, IEEE, 2019.
[16] W. Zheng, K. Wang, and F.-Y. Wang, "Gan-based key secret-sharing scheme in blockchain," *IEEE transactions on cybernetics*, vol. 51, no. 1, pp. 393–404, 2020.
[17] H. P. Singh, K. Stefanidis, and F. Kirstein, "A private key recovery scheme using partial knowledge," in *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, IEEE, 2021.
[18] D. Boneh, R. Gennaro, and S. Goldfeder, "Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security," in *International Conference on Cryptology and Information Security in Latin America*, pp. 352–377, Springer, 2017.
[19] P. Dikshit and K. Singh, "Efficient weighted threshold ecdsa for securing bitcoin wallet," in *2017 ISEA Asia Security and Privacy (ISEASP)*, pp. 1–9, IEEE, 2017.
[20] Q. Wei, S. Li, W. Li, H. Li, and M. Wang, "Decentralized hierarchical authorized payment with online wallet for blockchain," in *Wireless Algorithms, Systems, and Applications: 14th International Conference, WASA 2019, Honolulu, HI, USA, June 24–26, 2019, Proceedings 14*, pp. 358–369, Springer, 2019.
[21] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *International conference on the theory and applications of cryptographic techniques*, pp. 127–144, Springer, 1998.
[22] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage,"

ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 1, pp. 1–30, 2006.

[23] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[24] V. Buterin, "Ethereum whitepaper," https://ethereum.org/en/whitepaper/, 2022.

[25] W. Dai, Y. Lv, K.-K. R. Choo, Z. Liu, D. Zou, and H. Jin, "Crsa: A cryptocurrency recovery scheme based on hidden assistance relationships," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4291–4305, 2021.

[26] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*, pp. 213–229, Springer, 2001.

[27] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[28] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," tech. rep., 2017.

[29] L. Brandão and M. Davidson, "Notes on threshold eddsa/schnorr signatures," 2022.

[30] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *International conference on provable security*, pp. 1–16, Springer, 2007.

[31] D. Boneh, X. Ding, G. Tsudik, and C. M. Wong, "A method for fast revocation of public key certificates and security capabilities," in *10th USENIX Security Symposium (USENIX Security 01)*, 2001.

[32] Y. Lindell, "Fast secure two-party ecdsa signing," in *Annual International Cryptology Conference*, pp. 613–644, Springer, 2017.

[33] Y. Lindell, "Fast secure two-party ecdsa signing," *Journal of Cryptology*, vol. 34, no. 4, pp. 1–38, 2021.

[34] Y. Lindell and A. Nof, "Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1837–1854, 2018.

[35] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*, pp. 129–140, Springer, 1991.

[36] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous distributed key generation," in *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 2518–2534, IEEE, 2022.

[37] L. Zhang, F. Qiu, F. Hao, and H. Kan, "1-round distributed key generation with efficient reconstruction using decentralized cp-abe," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 894–907, 2022.

[38] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Annual International Cryptology Conference*, pp. 398–412, Springer, 1999.

[39] E. Trichina and A. Bellezza, "Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 98–113, Springer, 2002.

[40] E. Kiltz and K. Pietrzak, "Leakage resilient elgamal encryption," in *International conference on the theory and application of cryptology and information security*, pp. 595–612, Springer, 2010.

[41] J. Zhang, F. Zhang, X. Huang, and X. Liu, "Leakage-resilient authenticated key exchange for edge artificial intelligence," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2835–2847, 2020.

[42] J. Zhang and F. Zhang, "Identity-based key agreement for blockchain-powered intelligent edge," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6688–6702, 2021.

[43] J. Alawatugoda and T. Okamoto, "Standard model leakage-resilient authenticated key exchange using inner-product extractors," *Designs, Codes and Cryptography*, vol. 90, no. 4, pp. 1059–1079, 2022.

[44] Y. Zhou, Y. Xu, Z. Qiao, B. Yang, and M. Zhang, "Continuous leakage-resilient certificate-based signcryption scheme and application in cloud computing," *Theoretical Computer Science*, vol. 860, pp. 1–22, 2021.

[45] J. B. Nielsen and M. Simkin, "Lower bounds for leakage-resilient secret sharing," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 556–577, Springer, 2020.

[46] H. K. Maji, A. Paskin-Cherniavsky, T. Suad, and M. Wang, "Constructing locally leakage-resilient linear secret-sharing schemes," in *Annual International Cryptology Conference*, pp. 779–808, Springer, 2021.

[47] I. Tjuawinata and C. Xing, "Leakage-resilient secret sharing with constant share size," *IEEE Transactions on Information Theory*, 2022.

[48] "Jaxx," https://www.jaxxwallet.io, 2022.

[49] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Communications magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[50] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*, pp. 213–229, Springer, 2001.

[51] C. Cremers, B. Kiesl, and N. Medinger, "A formal analysis of {IEEE} 802.11's {WPA2}: Countering the kracks caused by cracking the counters," in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1–17, 2020.

[52] N. Kshetri, "Web 3.0 and the metaverse shaping organizations brand and product strategies," *IT Professional*, vol. 24, no. 2, pp. 11–15, 2022.

[53] E. Rescorla, "The transport layer security (tls) protocol version 1.3," tech. rep., 2018.

[54] J. Zhang, *Authenticated Key Exchange Protocols with Unbalanced Computational Requirements*. The University of Liverpool (United Kingdom), 2018.

[55] Y. Wang, Z. Su, N. Zhang, D. Liu, R. Xing, T. H. Luan, and X. Shen, "A survey on metaverse: Fundamentals, security, and privacy," *arXiv preprint arXiv:2203.02662*, 2022.

[56] C. Boyd, A. Mathuria, and D. Stebila, *Protocols for authentication and key establishment*, vol. 1. Springer, 2003.

[57] A. Lei, H. Cruickshank, Y. Cao, P. Asuquo, C. P. A. Ogah, and Z. Sun, "Blockchain-based dynamic key management for heterogeneous intelligent transportation systems," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1832–1843, 2017.

[58] Z. Ma, J. Zhang, Y. Guo, Y. Liu, X. Liu, and W. He, "An efficient decentralized key management mechanism for vanet with blockchain," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 6, pp. 5836–5849, 2020.

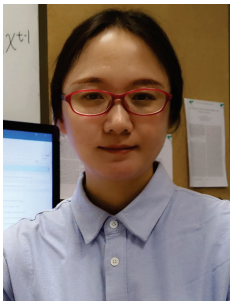
[59] S. S. Panda, D. Jena, B. K. Mohanta, S. Ramasubbareddy, M. Daneshmand, and A. H. Gandomi, "Authentication and key management in distributed iot using blockchain technology," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12947–12954, 2021.

[60] M. Baza, M. M. Fouda, M. Nabil, A. T. Eldien, H. Mansour, and M. Mahmoud, "Blockchain-based distributed key management approach tailored for smart grid," in *Combating Security Challenges in the Age of Big Data*, pp. 237–263, Springer, 2020.

[61] J. Li, J. Wu, L. Chen, J. Li, and S. K. Lam, "Blockchain-based secure key management for mobile edge computing," *IEEE Transactions on Mobile Computing*, 2021.
[62] Y. Tan, J. Liu, and N. Kato, "Blockchain-based key management for heterogeneous flying ad hoc network," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7629–7638, 2020.
[63] V. Ribeiro, R. Holanda, A. Ramos, and J. J. Rodrigues, "Enhancing key management in lorawan with permissioned blockchain," *Sensors*, vol. 20, no. 11, p. 3068, 2020.
[64] T. Chen, L. Zhang, K.-K. R. Choo, R. Zhang, and X. Meng, "Blockchain-based key management scheme in fog-enabled iot systems," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10766–10778, 2021.
[65] J. Wang, L. Wu, K.-K. R. Choo, and D. He, "Blockchain-based anonymous authentication with key management for smart grid edge computing infrastructure," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1984–1992, 2019.
[66] Y. Tian, Z. Wang, J. Xiong, and J. Ma, "A blockchain-based secure key management scheme with trustworthiness in dwsns," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6193–6202, 2020.
[67] W. M. Shbair, E. Gavrilov, and R. State, "Hsm-based key management solution for ethereum blockchain," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–3, IEEE, 2021.
[68] T. Cai, Z. Hong, S. Liu, W. Chen, Z. Zheng, and Y. Yu, "Socialchain: Decoupling social data and applications to return your data ownership," *IEEE Transactions on Services Computing*, 2021.
[69] A. Lei, H. Cruickshank, Y. Cao, P. Asuquo, C. P. A. Ogah, and Z. Sun, "Blockchain-based dynamic key management for heterogeneous intelligent transportation systems," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1832–1843, 2017.
[70] H. Mayer, "Ecdsa security in bitcoin and ethereum: a research survey," *CoinFaabrik, June*, vol. 28, no. 126, p. 50, 2016.
[71] R. Henry, A. Herzberg, and A. Kate, "Blockchain access privacy: Challenges and directions," *IEEE Security & Privacy*, vol. 16, no. 4, pp. 38–45, 2018.

Biographies

Ziyang Ji received his M.S. degree from Xi'an Jiaotong-Liverpool University in 2020. He is currently working towards the Ph.D. degree from the University of Liverpool. His research interests include cryptography and blockchain.



Jie Zhang received her Ph.D. degree from the University of Liverpool in 2018. She is now an Associate Professor at Xi'an Jiaotong-Liverpool University. Her research interests include cryptography and cyberspace security.

Yuji Dong received his B.Eng. in Electronic Science and Technology in 2011 from XJTLU, China, M.Sc. in Advanced Computer Science in 2012 and Ph.D. in Computer Science in 2018 from the University of Liverpool. He is currently working as an Assistant Professor at School of Internet of Things, Xi'an Jiaotong Liverpool University. His research interests primarily focus on the ecological construction of Internet of Things (IoT) systems from software engineering perspective including multiple technologies such as blockchain, human-computer interaction, reinforcement learning, and multi-agent collaboration.

Ka Lok Man received the Dr. Eng. degree in electronic engineering from the Politecnico di Torino, Turin, Italy, in 1998, and the Ph.D. degree in computer science from Technische Universiteit Eindhoven, Eindhoven, The Netherlands, in 2006. He is currently a Professor at Xi'an Jiaotong-Liverpool University, Suzhou, China. His research interests include formal methods and process algebras, embedded system design and testing, AI, and photovoltaics.

Steven Guan received his BSc. from Tsinghua University and M.Sc. & Ph.D. from the University of North Carolina at Chapel Hill. He is currently a Professor at Xi'an Jiaotong-Liverpool University (XJTLU). He served the head of department position at XJTLU for 4.5 years, creating the department from scratch and now in shape. Before joining XJTLU, he was a tenured professor and chair in intelligent systems at Brunel University, UK. Prof. Guan's research interests include: machine learning, computational intelligence, big data analytics, mobile commerce, modeling, networking, personalization, security, coding theory, and pseudorandom number generation.

Mucheol Kim is a professor in the School of Computer Science and Engineering at Chung-Ang University. He received the BS, MS, Ph.D. degrees from the school of Computer Science and Engineering at ChungAng University, Seoul, Korea in 2005, 2007 and '2012, respectively. He was an assistant professor in a department of computer & software engineering at Wonkwang University (2017-2018). In 2014-2016, he was an assistant professor of Department of Media Software at Sungkyul University, Korea. In 2011-2014, he had been working as a Senior Researcher in Korea Institute of Science and Technology Information (KISTI), Daejeon, Korea. His research interests include Graph Neural Networks, Information Retrieval, Web Technology, Social Networks, Disaster Management and Language Models.