KCRS: A Blockchain-Based Key Compromise Resilient Signature System

Lei Xu[1(✉)], Lin Chen[2], Zhimin Gao[3], Xinxin Fan[5], Kimberly Doan[6], Shouhuai Xu[6], and Weidong Shi[4]

[1] University of Texas Rio Grande Valley, Brownsville 78520, USA
lei.xu@utrgv.edu
[2] Texas Tech University, Lubbock 79409, USA
[3] Auburn University at Montgomery, Montgomery 36117, USA
[4] University of Houston, Houston 77004, USA
[5] IoTeX, Menlo Park 94025, USA
[6] University of Texas at San Antonio, San Antonio 78249, USA

Abstract. Digital signatures are widely used to assure authenticity and integrity of messages (including blockchain transactions). This assurance is based on assumption that the private signing key is kept secret, which may be exposed or compromised without being detected in the real world. Many schemes have been proposed to mitigate this problem, but most schemes are not compatible with widely used digital signature standards and do not help detect private key exposures. In this paper, we propose a Key Compromise Resilient Signature (KCRS) system, which leverages blockchain to detect key compromises and mitigate the consequences. Our solution keeps a log of valid certificates and digital signatures that have been issued on the blockchain, which can deter the abuse of compromised private keys. Since the blockchain is an open system, KCRS also provides a privacy protection mechanism to prevent the public from learning the relationship between signatures. We present a theoretical framework for the security of the system and a provably-secure construction. We also implement a prototype of KCRS and conduct experiments to demonstrate its practicability.

Keywords: Digital signature · Key Compromise Resilient · Blockchain · Privacy · Exposure detection

1 Introduction

Digital signatures can assure the authenticity and integrity of messages and play a critical role in many applications, while assuming that the private signing key is kept secret. In the real world, it is difficult to assure the security of private signing keys because the system storing the private signing key can be compromised. This has motivated a sequence of studies on mitigating the damages of key compromises, such as [1,4,6,7,9–11,13,16,18–20,26–30]. Even if a private signing key is not compromised, an attacker still can exploit its service to obtain

Z. Zheng et al. (Eds.): BlockSys 2019, CCIS 1156, pp. 226–239, 2020.
https://doi.org/10.1007/978-981-15-2777-7_19

legitimate digital signatures [24,31]. Advanced cryptographic mechanisms, such as forward secure digital signatures [1,4,20], key-insulated public key cryptosystems [12–14], and intrusion-resilient schemes [19], can mitigate the damages of private signing key compromises. However, they are not compatible with existing digital signature standards and cannot detect key compromises.

On the other hand, the Certificate Transparency (CT) framework [22] has been proposed for monitoring and auditing TLS/SSL certificates with a cryptographically assured, publicly auditable, append-only certificate log. Although this approach is compatible to existing standards and can detect key compromises, it lacks the recovery capability for the compromised certificates. Recent work [2,25] has resorted to blockchain for addressing the aforementioned challenges, due to its salient features such as immutability and tamper detection. This approach is reminiscent of earlier studies on managing digital signatures [17,31]. However, these digital signature management systems expose users' behavior to the public, thereby raising privacy concerns.

In this paper, we propose a privacy-enhanced Key Compromise Resilient Signature framework, or KCRS for short, by leveraging a blockchain to enable privacy-preserving key compromise detection, invalid signature revocation, and key update. KCRS supports standardized digital signature schemes and utilizes a dual key strategy with a "signing" key pair for message authentication and a "master" key pair for signature linkage. In KCRS, both valid public keys and generated signatures are stored on the blockchain, which allows a user to easily detect a compromised signing key by monitoring blockchain records and act promptly. One-time signature and encryption are employed to protect users' privacy, such as the messages they signed and the pattern of their signature generations. A prototype implementation of KCRS using Hyperledger Fabric [3] demonstrates its effectiveness and efficiency in practice.

This paper is organized as follows. Section 2 describes the KCRS framework and its security model. Section 3 presents a concrete construction of KCRS and its security analysis. Section 4 describes the integration of KCRS with blockchain. Section 5 discusses the implementation of KCRS and evaluates its performance. Section 6 concludes the paper.

2 The KCRS Framework and Security Model

2.1 The KCRS Framework

A KCRS scheme consists of the following six algorithms:

Initialization. The following algorithm is used to initialize KCRS.

– $Setup\ (\lambda) \rightarrow pp$. The algorithm *Setup* takes the security parameter λ as input, and outputs the public parameters that are used by other algorithms.

Master Key Initialization. The following algorithm is used by a user (signer or verifier) to get his/her first key pair and register to KCRS.

– *MKGen* $(pp) \rightarrow (pk_M, sk_M)$. *MKGen* takes the public parameters pp as input, and outputs a master public/private key pair (pk_M, sk_M), where pk_M also serves as the identity of its owner.

Signing Key Pair Generation. The following algorithms are used by a signer to select a signing public/private key pair.

– *SPKGen* $(pp, pk_M) \rightarrow (pk_S, aux)$. Given a master public key pk_M and the public parameters pp, the algorithm *SKGen* returns a randomly selected signing public key pk_S and related auxiliary information aux.
– *SSKGen* $(pp, pk_S, aux, sk_M, pk_M) \rightarrow sk_S$. The algorithm is deterministic and takes the public parameters pp, the generated signing public key pk_S with auxiliary information aux, and master public/private key pair (sk_M, pk_M) as inputs. It returns a signing private key sk_S, which is used to generate signatures that can be verified by pk_S.

Signing Key Pair Detection. The following deterministic algorithm is used by a signer to check whether a given signing public key is generated using his/her master public/private key pairing information.

– *SKDetect* $(pp, sk_M, pk_M, pk_S) \rightarrow \delta$. If pk_S is generated using sk_M and pk_M, the algorithm returns 1; otherwise, it returns 0.

Signature Algorithm. The following two algorithms are used to generate/verify digital signatures.

– *SigGen* $(pp, sk_S, m) \rightarrow \sigma_{m,pk_S}$. The function *SigGen* generates signature of m using signing private key sk_S.
– *SigVerify* $(pp, \sigma_{m,pk_S}, m, pk_S) \rightarrow \delta$. The algorithm *SigVerify* returns $\delta = 1$ if the signature sig_{m,pk_S} is valid with respect to message m and public key pk_S; otherwise, it returns $\delta = 0$.

Link Verification. The following algorithm is used for one to establish the connection between a signature and its signer.

– *LNK* $(pp, \sigma, m, pk_M) \rightarrow \delta$. This is a deterministic algorithm that allows one to check whether the signer of σ is the owner of pk_M.

For a signature scheme, we usually assume both the message and the signature are in public. As a result, everyone can use *LNK* to recover the signer's identity. In the concrete construction of KCRS, we demonstrate how to limit this linking capability to the designated signature verifier via encryption.

2.2 Security Definitions of KCRS

Correctness. This means that a KCRS scheme works normally when the signer and verifier are honest. Specifically, (i) the signer can generate signing key pairs and sign messages that can be accepted by a verifier; (ii) if the signer sees a public signing key that is derived from her/his master public key, he/she can

detect it; and (iii) for a targeted signature verifier, he/she can check whether a signature is related to a master public key or not.

Signature Unforgeability. This means that only the one who knows the signing private key can generate a valid signature, dubbed existential unforgeability under adaptive chosen-message attacks (EUF-CMA).

Definition 1 (EUF-CMA [15]**).** *A KCRS scheme is EUF-CMA secure if the probability attacker $\mathcal{A}$ wins the following game is negligible in security parameter λ:*

1. *The challenger $\mathcal{C}$ generates a pair of public/private key pair (pk_S, sk_S) using the public parameter pp, and gives pk to the adversary $\mathcal{A}$.*
2. *$\mathcal{A}$ queries $\mathcal{C}$ for signatures $\sigma_1, \ldots, \sigma_q$ on adaptively chosen messages $m_1, \ldots, m_q$, respectively.*
3. *$\mathcal{A}$ produces a pair of message and signature (m^*, σ^*). If m^* is not queried in a previous step and σ^* is a valid signature of m^*, then $\mathcal{A}$ succeeds.*

Note that KCRS has two pairs of public/private keys, but only the signing key pair is used for digital signature operations.

Signing Private Key Unforgeability. Unforgeability of signing private key (UF-SSK) assures that only the user who has the master private key can generate a signing private key, as formulated in Definition 2.

Definition 2 (UF-SSK) . *A KCRS scheme is UF-SSK secure if the probability that the adversary wins the following game is negligible in security parameter λ:*

1. *The challenger $\mathcal{C}$ generates a pair of master public/private key pair (pk_M, sk_M) using the public parameters pp. The master public key is given to adversary $\mathcal{A}$.*
2. *$\mathcal{A}$ queries $\mathcal{C}$ to obtain a sequence responses, e.g., signing public/private key pairs derived from (pk_M, sk_M) and other related information.*
3. *$\mathcal{A}$ generates a signing public/private key pair (pk_S^*, sk_S^*), which is different from any of the ones obtained in the previous queries.*
4. *If pk_S^* is derived from pk_M and matches with sk_S^*, then $\mathcal{A}$ succeeds.*

Signing Public Key Indistinguishability. Signing public key indistinguishability (IND-SPK) means that one cannot distinguish signing public keys generated from different master keys, thereby protecting the privacy of the signer.

Definition 3 (IND-SPK) . *A KCRS scheme is IND-SPK secure if the probability that an adversary wins the following game is negligibly greater than 1/2 (with respect to security parameter λ).*

1. *The challenger $\mathcal{C}$ generates a master public/private key pair (pk_M, sk_M) using the public parameters pp, and gives pk_M to attacker $\mathcal{A}$.*
2. *$\mathcal{A}$ queries $\mathcal{C}$ with selected auxiliary information for signing public keys derived from pk_M.*

3. *When $\mathcal{A}$ finishes the query phase, $\mathcal{C}$ runs SPKGen on pk_M to generate $pk_S^{(0)}$ and corresponding auxiliary information aux. $\mathcal{C}$ also randomly selects another key $pk_S^{(1)}$*
4. *$\mathcal{C}$ randomly selects $b \in \{0,1\}$, and sends $(pk_S^{(b)}, aux)$ to $\mathcal{A}$.*
5. *$\mathcal{A}$ guesses the value of b and outputs $\hat{b}$.*
6. *If $b = \hat{b}$, $\mathcal{A}$ succeeds and the game returns 1; otherwise $\mathcal{A}$ fails and the game returns 0.*

3 A KCRS Construction and Its Security Analysis

3.1 An ECDSA-Based KCRS Construction

Suppose two parties (i.e., the signer and the targeted verifier) share a secrete key dk securely. This can be achieved through an offline channel (e.g., using public key encryption/key encapsulation [8]).

- *Setup* $(\lambda) \rightarrow pp$, where λ is the security parameter and $pp = (E(\mathbb{F}), P)$, where $E(\mathbb{F})$ is a selected elliptic curve on finite field $\mathbb{F}$, and P is a point on $E(\mathbb{F})$ with order about λ bits.
- *MKGen* $(pp) \rightarrow (pk_M, sk_M)$, where $pp = (E(\mathbb{F}), P)$. This algorithm randomly selects a positive integer $s \xleftarrow{\$} (0, |P|)$ and sets $(pk_M, sk_M) = (sP, s)$.
- *SPKGen* $(pp, pk_M) \rightarrow (pk_S, aux)$. This algorithm randomly selects an integer $r \xleftarrow{\$} (0, |P|)$ and calculates $(pk_S, aux) = (h(r \cdot pk_M)P + pk_M, rP)$, where $h() : E(\mathbb{F}) \rightarrow \mathbb{Z}_{ord(P)}$ is a hash function which works as a random oracle. In practice, $h(\cdot)$ can be implemented using SHA512 mod $ord(P)$ when $\lambda < 512$.
- *SSKGen* $(pp, pk_S, aux, pk_M, sk_M) \rightarrow sk_S$. This algorithm first checks $pk_S \stackrel{?}{=} h(sk_M \cdot aux)P + pk_M$, where aux is a point on $E(\mathbb{F})$. If it passes the test, the algorithm computes and returns $sk_S = h(sk_M \cdot aux) + sk_M$; otherwise, it returns $sk_S = \perp$.
- *SKDetect* $(pp, sk_M, pk_M, pk_S, aux) \rightarrow \delta$. This algorithm calculates $pk'_S \leftarrow h(sk_M \cdot aux)P + pk_M$. If $pk_S = pk'_S$, it sets $\delta \leftarrow 1$; otherwise, it sets $\delta \leftarrow 0$.
- *SigGen* $(pp, sk_S, m, dk) \rightarrow \sigma_{m', pk_S}$. This algorithm pre-processes the message m before signing. Specifically, it calculates $m' \leftarrow Enc(m||r, dk)$ and generates $\sigma_{m', pk_S} \leftarrow Sign_{ECDSA}(sk_S, m')$, where $Enc()$ is a secure symmetric encryption scheme, dk is the secret key shared between the signer and the targeted verifier, and r is the random number selected by the signer in $SPKGen()$. m' is released to the public together with the signature σ_{m', pk_S}.
- *SigVerify* $(pp, \sigma_{m', pk_S}, m', pk_S) \rightarrow \delta$. This algorithm is the same as the ECDSA signature verification algorithm $Verify_{ECDSA}$. If the signature σ_{m', pk_S} matches m' and pk_S, it returns $\delta \leftarrow 1$; otherwise, it returns $\delta \leftarrow 0$.
- *LNK* $(pp, \sigma_{m', pk_S}, m', pk_M, dk)$. This algorithm first checks that σ_{m', pk_S} is a valid signature of m', and runs $Dec(m', dk)$ to recover r. The algorithm then computes $pk'_S \leftarrow h(r \cdot pk_M)P + pk_M$. If $pk'_S = pk_S$, it sets $\delta \leftarrow 1$ (i.e., signature σ_{m', pk_S} is linked to pk_M); otherwise, it sets $\delta \leftarrow 0$.

3.2 Security Analysis

Correctness of the KCRS construction can be verified by observation. Signature unforgeability is based on the security of ECDSA, which has been proved to be UF-CMA secure [23].

Signing Private Key Unforgeability (UF-SSK). This property is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP): Given a generator P of an elliptic curve E, and a random point $Q \in \langle P \rangle$, find r such that $Q = rP$. In what follows, we first describe a simulation algorithm $\mathcal{S}$ that leverages $\mathcal{A}$ to solve an ECDLP instance and then prove the success probability of $\mathcal{S}$. In order for $\mathcal{A}$ to produce a signing public/private key pair, s/he may need to see a sequence of signing public/private key pairs derived from the same master public/private key pair. During this procedure, we allow $\mathcal{A}$ to learn $h(\cdot) \cdot P$ based on his/her selection of random number r. However, we do not allow $\mathcal{A}$ to query hash function $h(\cdot)$ directly because it will disclose the master private key when $\mathcal{A}$ can query both the signing private key and the hash value. $\mathcal{S}$ maintains four tables corresponding to $\mathcal{A}$'s queries with selected random numbers: $R[i]$, which stores the random number $\mathcal{A}$ selected for the ith query of signing key pair; $H[i]$, which stores the scalar multiplication of the hash value and the base point for the ith query; $P[i]$, which stores the signing public key for the ith query; and $S[i]$, which stores the signing private key for the ith query. Because $\mathcal{A}$ is a probabilistic polynomial-time algorithm, it can make at most N queries before it outputs the fake signing key pair, where N is bounded by a polynomial of λ. $\mathcal{S}$ randomly selects an opportunity to feed the ECDLP instance to $\mathcal{A}$ and hopes it will generate the fake signing key pair based on the input. Algorithm 1 describes the simulator $\mathcal{S}$.

Theorem 1. *The KCRS construction is UF-SSK secure under the ECDLP assumption in the random oracle model.*

Proof (sketch). In the random oracle model, $\mathcal{A}$ cannot distinguish the values $\mathcal{S}$ provided from the values in the real system. Thus, $\mathcal{A}$ will produce the fake signing key pair in Algorithm 1. Since $\mathcal{A}$ makes at most N queries and $\mathcal{S}$ randomly picks an opportunity in this process to leverage $\mathcal{A}$ to solve the target ECDLP instance, the probability that $\mathcal{A}$ decides to produce a fake signing key pair at the same time is at least $1/N$. If the target instance has been queried before, $\mathcal{D}$ will terminate and fail. However, the probability of such an event is negligible in λ because the space of r_i's is exponential in λ. Denote by E_0 the event that $\mathcal{A}$ successfully fakes a signing key pair after querying the oracle, and E_1 the event that $\mathcal{S}$ successfully solves the ECDLP instance. We have

$$\Pr[E_1] \geq \frac{1}{N} \Pr[E_0] - negl,$$

where *negl* is the negligible probability that $\mathcal{A}$ queries the target ECDLP instance. If $\mathcal{A}$ can compromise the UF-SSK feature with a non-negligible probability, then $\frac{1}{N} \Pr[E_0] - negl$ is non-negligible and *negl* is negligible. Therefore, $\Pr[E_1]$ is non-negligible, which contradicts with the ECDLP assumption. □

Algorithm 1. ECDLP solver $\mathcal{S}$ using UF-SSK adversary $\mathcal{A}$.

Input: The base point P on $E(\mathbb{F})$; a random point $Q \leftarrow sP$ on $E(\mathbb{F})$, which is the ECDLP instance $\mathcal{S}$ wants to solve; the master public key pk_M, a point on $E(\mathbb{F})$; the maximum number of oracle queries N.

Output: An integer s or $\perp$.

$R[\,] \leftarrow \emptyset$; $T[\,] \leftarrow \emptyset$; $H[\,] \leftarrow \emptyset$; $P[\,] \leftarrow \emptyset$; $S[\,] \leftarrow \emptyset$; $j \xleftarrow{\$} [1, N]$;
for $i = 1$ **to** N **do**
  $\mathcal{A}$ selects a random number $r_i \in Z_{|P|}$; $R[i] \leftarrow r_i$; $idx \leftarrow \texttt{FIND}(R[\,], r_i)$;
  if $i = j$ **and** $\mathcal{A}$ decides to generate the fake signing key pair **then**
    if $idx = 0$ **then**
      $H[i] \leftarrow Q - pk_M$, $P[i] \leftarrow Q$, which are shared with $\mathcal{A}$;
      $\mathcal{A}$ outputs a fake s;
      return s;
    else
      return $\perp$;
    end if
  end if
  if $idx = 0$ **then**
    $S[i] \xleftarrow{\$} (0, |P|)$; $P[i] \leftarrow S[i] \cdot P$; $H[i] \leftarrow P[i] - pk_M$;
  else
    $S[i] \leftarrow S[idx]$; $P[i] \leftarrow P[idx]$; $H[i] \leftarrow P[idx]$;
  end if
  If $\mathcal{A}$ queries the scalar multiplication of the hash value and the base point, return $H[i]$;
  If $\mathcal{A}$ queries the derived signing public/private key pair, return $(P[i], S[i])$;
end for

Signing Public Key Indistinguishability (IND-SPK). The IND-SPK security of the KCRS construction is based on the hardness of the following variant hashed decision Diffie-Hellman (VH-DDH) assumption, which has not been studied in the literature.

Definition 4 (VH-DDH on elliptic curve). *Given a generator P of an elliptic curve E, and three random integers $u, v, z \in (0, ord(P))$, VH-DDH assumption says that $(uP, vP, H(uvP)P)$ and $(uP, vP, H(zP)P)$ are computationally indistinguishable; i.e., for any probabilistic polynomial-time algorithm $\mathcal{D}$, $|\Pr[\mathcal{D}(uP, vP, H(uvP)P) = 1] - \Pr[\mathcal{D}(uP, vP, H(zP)P)] = 1|$ is negligible in the size of $ord(P)$, where H is a cryptography hash function works as a random oracle.*

The standard DDH assumption assumes (uP, vP, uvP) and (uP, vP, zP) are computationally indistinguishable. If f is a one-to-one mapping, it is easy to see that distinguishing $(uP, vP, f(uvP)P)$ and $(uP, vP, f(zP)P)$ is at least as hard as corresponding standard DDH problem; otherwise, an attacker can apply f to the DDH instance to solve it. For the variant hashed decisional Diffie-Hellman problem, the mapping function is a composition of a hash function and a scalar multiplication on the elliptic curve. The scalar multiplication is a one-to-one

mapping when the scalar belongs to $(0, ord(P))$, which is true when the output size of the hash function is less than $ord(P)$. When the scalar multiplication is composed with the cryptography hash function that works as a random oracle, the result function is not one-to-one any more as the hash function can have collisions. However, the likelihood of collision is negligible, and one can still apply the composed function to a DDH instance and solve it with high probability if the variant hashed DDH is easy. Therefore, the variant hashed DDH problem is at least as hard as the standard DDH problem.

Theorem 2. *The KCRS construction is IND-SPK secure under the VH-DDH assumption in the random oracle model.*

Proof (sketch). The proof is to show that given an IND-SPK adversary $\mathcal{A}$, one can build an algorithm $\mathcal{D}$ to solve the VH-DDH problem as described in Algorithm 2.

Algorithm 2. VH-DDH solver $\mathcal{D}$ leveraging IND-SPK adversary $\mathcal{A}$.

Input: Point P on $E(\mathbb{F})$; VH-DDH instance $(uP, vP, H(zP)P)$; master public key $pk_M = uP$; the maximum number of oracle queries N
Output: A bit b where $b = 1$ if $uvP = H(zP)P$ and $b = 0$ otherwise
$i \leftarrow 1$; $R[\,] \leftarrow \emptyset$; $H[\,] \leftarrow \emptyset$;
while $\mathcal{A}$ wants to query for new signing public keys **do**
  $\mathcal{A}$ selects and send a random point r_iP to $\mathcal{D}$; $R[i] \leftarrow r_iP$; $idx \leftarrow \texttt{FIND}(R[\,], r_i)$;
  if $R[i] = vP$ **then**
    return $\perp$;
  end if
  if $idx = 0$ **then**
    $H[i] \xleftarrow{\$} (0, ord(P))$; Sending $H[i]P + uP$ to $\mathcal{A}$;
  else
    Sending $H[idx]P + uP$ to $\mathcal{A}$;
  end if
  $i \leftarrow i + 1$;
end while
$Q \xleftarrow{\$} \langle P \rangle$; $\hat{b} \leftarrow \mathcal{C}(uP, (H(zP)P + uP, H(Q)P + uP), vP)$;
if $b = \hat{b}$ **then**
  return 1;
else
  return 0;
end if

Denote by E_0 the event that the game given in Definition 3 returns 1 and E_1 the event that the game returns 1. If the input to Algorithm 2 is a valid VH-DDH instance, we have

$$\Pr[\mathcal{D}(uP, vP, H(uvP)P) = 1] = \Pr[E_0] - negl,$$

where *negl* is the probability that $\mathcal{A}$ queries with vP in the challenging phase. If the input instance is randomly generated, we have

$$\Pr[\mathcal{D}(uP, vP, H(zP)P) = 1] = \Pr[E_1] - negl.$$

We observe that $\Pr[E_1] = 1/2$ because the two signing public keys are independently and uniformly generated and follow the same distribution. Therefore, we have $Adv(\mathcal{D}) = |(\Pr[E_1] - negl) - (\Pr[E_0] - negl)| = |\Pr[S_0] - 1/2|$, which is equivalent to the advantage of $\mathcal{A}$ against the IND-SPK. □

4 KCRS on Blockchain

KCRS uses the blockchain as a unified information sharing and storage platform for transactions related to digital signatures, messages/signatures, and other kinds of relevant information. Assuming the blockchain is not controlled by the attacker, the use of blockchain prevents an adversary from altering existing transactions while allowing all participants to verify transaction validity. KCRS on blockchain has four types of participants:

- *Signer*: A signer owns a private key and uses the private key to produce digital signatures.
- *Verifier*: A verifier receives and verifies digital signatures generated by a signer. The verifier needs to be convinced that a certain signer has signed a specific message to determine his/her next step.
- *Certificate Authority* (CA): A CA is responsible for setting up the initial public key certificates for the signers and verifiers.
- *Miner*: Miners participate in transaction verification and blockchain maintenance by producing/verifying blocks.

A signer needs to generate a public/private key pair and obtain a certificate of the public key from the CA when he/she joins the system. This key pair serves as the master key pair and identity of the signer, but is not used for daily signature generation and verification. When a verifier needs a signer to sign a message, he/she first contacts the signer through an off-chain channel to exchange information including a symmetric key that will be used to encrypt the message in question. The signer then generates a signing public/private key pair and signs the encrypted message as described in the algorithm *SigGen* and submits the signature to the blockchain as transactions. All miners verify the validity of the signature before embedding it into a block and storing on the blockchain. Figure 1 summarizes the workflow of KCRS.

4.1 Data Management for KCRS on Blockchain

The blockchain stores four types of transactions: (i) certificate of master public key pk_M, which is generated by the CA and will not change frequently; (ii) signing public key pk_S, which is generated by the signer based on the master private

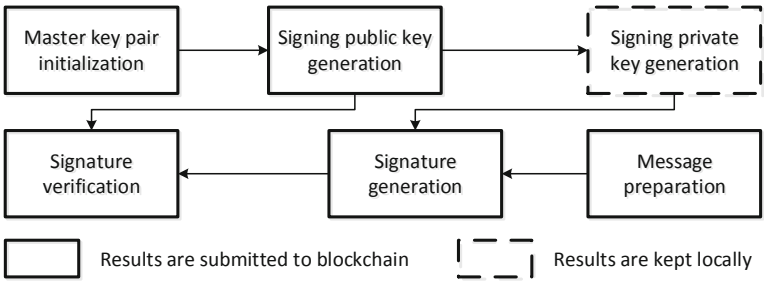

Fig. 1. The workflow of KCRS on blockchain. The output of each step is stored on the blockchain except the signing private key, and miners are responsible for validating outputs and maintaining the blockchain.

key; (iii) transformed message, which is the ciphertext of the original message plus the random number used to derive the signing key pair; (iv) signature, which is generated by the signer using signing private key.

Master public key certificate, signing public key, and signature have fixed sizes and are easy to be embedded into a block and included in the blockchain. However, the size of a message can vary greatly, so we let KCRS on blockchain stores the hash value of the message in a block while the message can be kept on another storage system that is more efficient and flexible. Logically, blocks are organized in a linear structure with a total order. In practice, a node can use a database with a dedicated field of order information to organize all transactions, and a user can search signatures or public keys by querying the database.

4.2 Operations of KCRS on Blockchain

Master Key Pair Initialization. We use permissioned blockchain, where each miner knows the CA's public key in advance. When a new user registers with the system, he/she submits his/her master public key together with a certificate of the public key. Each miner checks the certificate and the master public key, and runs a consensus protocol to include the registration information in the blockchain if it is valid.

Signing Key Pair Generation. According to the design of KCRS, a signing key pair is only used for a single message. The signer can either pre-generate a set of signing key pairs or wait until there is a need to sign a message. The signing public key is submitted to miners, and they run a consensus protocol to include it in the blockchain. The signer does not submit the random value used in the signing key pair generation to miners but keeps it secret.

Message Preparation. The verifier and signer communicate off-chain to exchange the message that needs to be signed and a symmetric key for encrypting the message. The signer adds information about the public key to the message and encrypts the result using the symmetric key. The prepared message does not need to be sent to the miners immediately.

Signature Verification. The signer sends the signature and message to the miners, who run the *SigVerify* algorithm to check the validity of the signature. If the signature is valid, the miners work together to include the pair to the blockchain. Note that the signature is not for the plaintext message, but the encrypted one, meaning that the miners do not need to see the plaintext message in order to verify the signature. If a signature is included in the blockchain, the verifier does not need to check the signature again. Instead, he/she only needs to decrypt the corresponding message stored on the blockchain with the signature, and then check whether the content is correct and the public key used to verify the signature is derived from the correct master public key using *LNK* algorithm.

5 Implementation and Performance Evaluation of KCRS

We implement KCRS using Hyperledger Fabric [3]. Since Fabric has its own PKI system, we use it to issue certificates to signers. Fabric divides the block construction into two steps: *endorsing* and *ordering*. In the process of endorsing, a group of endorsers check each transaction and endorse the valid ones by attaching their signatures. In the process of ordering, a group of orderers work together to determine the order of the endorsed transactions and put them on the blockchain. The default ordering in Fabric is implemented using Kafka [21], which is very efficient when the number of orderers is relatively small. To simply the implementation, we use a single node for the ordering service, and focus on the effect of different endorser configurations when measuring performance. Verifiers passively listen to KCRS and can retrieve signatures from the blockchain.

We deploy a KCRS prototype in Amazon Web Service (AWS). Nodes, including endorser nodes, are configured to utilize the t2.medium instance type, which has 2 processing cores and 4 GB memory. We distribute the nodes in different instances in order to reduce the bottleneck of computing resource. In this deployment, all endorser nodes join in a same channel. We also use different endorsement policies to manage the total number of signatures required by a transaction. For instance, policy *"AND('Org1.peer', 'Org2.peer')"* indicates that a transaction requires signatures from both organizations *Org1* and *Org2*.

Latency and Throughput. We evaluate the latency and throughput of the prototype with different parameters. Latency and throughput are mainly affected by to factors, the performance of the underlying blockchain system itself and the performance of the cryptographic operations. We fix the number of orderer node to one and measure the latency of per transaction. Figure 2 shows the results of changing the number of endorser nodes. We observe that the latency increases significantly when the number of endorsers changes from 1 to 3. This is because compared to a single-endorsement transaction, the orderer node need more time to process multiple endorsements. Both latency and throughput tend to level off when the number of endorsers is greater than 5.

Storage Cost. One of the major concerns of blockchain-based applications is the storage cost because the system has multiple copies of the blockchain and

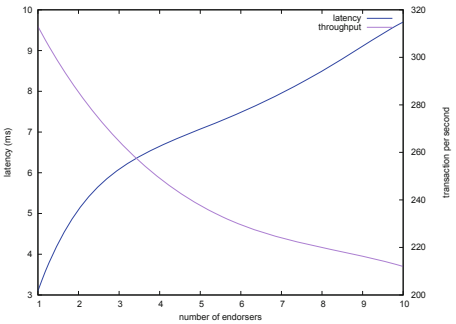

Fig. 2. KCRS performance: Latency is computed between the start time and the finish time with respect to a submitting a transaction; throughput is calculated by sending 1,000 transaction simultaneously and then collecting the start time and the finish time of the last block (if there are multiple blocks).

each of them keeps increasing. For blockchain-based KCRS, the storage cost of master public key certificates is negligible since they are relatively stable. Most transactions come from signatures and the generation of their signing public keys. When KCRS uses ECDSA with 256-bit keys, the size of a signature is 512 bits. The message size varies but we can keep its hash value (instead of the message itself) on the blockchain, the size of which is 256 bits when SHA256 is used. The auxiliary information attached to each signing public key is also an elliptic curve point, which is 512 bits. In summary, each signature request needs a storage of 192 bytes, and a modern computer can easily store hundreds of billions of such transactions. When elliptic curve compressing technologies are applied [5], the storage cost can be further reduced.

6 Conclusion

Private key exposure is one of the most devastating attacks against digital signatures. Although a variety of digital signature schemes have been proposed to mitigate the consequences of private key exposure, the detection of private signing key exposure has not been paid the due amount of attention. We have presented a new digital signature management framework, dubbed KCRS, which incorporates the capability of key exposure detection by leveraging the blockchain technology. We have described the formal security definition of KCRS: (i) only the legitimate user can update key information when key exposure is detected; (ii) only the relevant users can discover the relation between a signature and the signer. We have evaluated the performance of KCRS on blockchain and conducted experiments on Hyperledger Fabric. Experimental results show that KCRS on blockchain can handle a large number of users at a reasonable cost.

Acknowledgment. This work is supported in part by AFRL Grant #FA8750-19-1-0019 and NSF CREST Grant #1736209.

References

1. Abdalla, M., Reyzin, L.: A new forward-secure digital signature scheme. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 116–129. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_10
2. Al-Bassam, M.: Scpki: a smart contract-based PKI and identity system. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, pp. 35–40. ACM (2017)
3. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, p. 30. ACM (2018)
4. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_28
5. Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Möller, B.: Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls). Technical report (2006)
6. Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., Rosenblum, M.: Understanding data lifetime via whole system simulation. In: Proceedings of Usenix Security Symposium 2004 (2004)
7. Chow, J., Pfaff, B., Garfinkel, T., Rosenblum, M.: Shredding your garbage: reducing data lifetime. In: Proceedings 14th USENIX Security Symposium, August 2005
8. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. **33**(1), 167–226 (2003)
9. Dai, W., Parker, T.P., Jin, H., Xu, S.: Enhancing data trustworthiness via assured digital signing. IEEE Trans. Dependable Secure Comput. **9**(6), 838–851 (2012)
10. Ding, X., Tsudik, G., Xu, S.: Leak-free group signatures with immediate revocation. In: 24th International Conference on Distributed Computing Systems (ICDCS 2004), pp. 608–615. IEEE Computer Society (2004)
11. Ding, X., Tsudik, G., Xu, S.: Leak-free mediated group signatures. J. Comput. Secur. **17**(4), 489–514 (2009)
12. Dodis, Y., Katz, J., Xu, S., Yung, M.: Key-insulated public key cryptosystems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 65–82. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_5
13. Dodis, Y., Katz, J., Xu, S., Yung, M.: Strong key-insulated signature schemes. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 130–144. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_10
14. Dodis, Y., Luo, W., Xu, S., Yung, M.: Key-insulated symmetric key cryptography and mitigating attacks against cryptographic cloud software. In: Proceedings ASIACCS 2012, pp. 57–58 (2012)
15. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)
16. Guan, L., Lin, J., Luo, B., Jing, J., Wang, J.: Protecting private keys against memory disclosure attacks using hardware transactional memory. In: Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP 2015, pp. 3–19 (2015)
17. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 437–455. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_32

18. Harrison, K., Xu, S.: Protecting cryptographic keys from memory disclosure attacks. In: The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25–28 June 2007, Edinburgh, UK, Proceedings, pp. 137–143 (2007)
19. Itkis, G., Reyzin, L.: SiBIR: signer-base intrusion-resilient signatures. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 499–514. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_32
20. Krawczyk, H.: Simple forward-secure signatures from any signature scheme. In: ACM Conference on Computer and Communications Security, pp. 108–115 (2000)
21. Kreps, J., Narkhede, N., Rao, J., et al.: Kafka: a distributed messaging system for log processing. In: Proceedings of the NetDB, pp. 1–7 (2011)
22. Laurie, B., Langley, A., Kasper, E.: Certificate transparency. Technical report (2013)
23. Locke, G., Gallagher, P.: Fips pub 186–3: digital signature standard (dss). Federal Information Processing Standards Publication 3, 186–3 (2009)
24. Loscocco, P., Smalley, S., Muckelbauer, P., Taylor, R., Turner, S., Farrell, J.: The inevitability of failure: the flawed assumption of security in modern computing environments. In: Proceedings 21st National Information Systems Security Conference (NISSC 1998) (1998)
25. Orman, H.: Blockchain: the emperors new PKI? IEEE Internet Comput. **22**(2), 23–28 (2018)
26. Parker, T.P., Xu, S.: A method for safekeeping cryptographic keys from memory disclosure attacks. In: First International Conference on Trusted Systems (INTRUST 2009), pp. 39–59 (2009)
27. Shamir, A., van Someren, N.: Playing 'Hide and Seek' with stored keys. In: Franklin, M. (ed.) FC 1999. LNCS, vol. 1648, pp. 118–124. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48390-X_9
28. Xu, S., Li, X., Parker, T.P.: Exploiting social networks for threshold signing: attack-resilience vs. availability. In: Proceedings of ASIACCS 2008, pp. 325–336 (2008)
29. Xu, S., Li, X., Parker, T.P., Wang, X.: Exploiting trust-based social networks for distributed protection of sensitive data. IEEE Trans. Inf. Forensics Secur. **6**(1), 39–52 (2011)
30. Xu, S., Sandhu, R.: A scalable and secure cryptographic service. In: Barker, S., Ahn, G.-J. (eds.) DBSec 2007. LNCS, vol. 4602, pp. 144–160. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73538-0_12
31. Xu, S., Yung, M.: Expecting the unexpected: towards robust credential infrastructure. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 201–221. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03549-4_12